TRUBA Üzerinde Kuantum Algoritma Çalıştırma Hands-on Eğitimi

İçindekiler:

1. Hands-on Bölümünün Amacı

Bu bölümün amacı, yarışmacıların TRUBA üzerinde kuantum algoritma kodlarını çalıştırabilmeleri için gerekli temel iş akışını uygulamalı olarak göstermektir.

Eğitim sonunda katılımcıların aşağıdaki adımları gerçekleştirebilmesi beklenmektedir:

- TRUBA üzerinde çalışma klasörü oluşturma,

- Verilen Python kodunu uygun klasör yapısına yerleştirme,

- SLURM template dosyasındaki gerekli alanları düzenleme,

- GPU node üzerinde iş çalıştırma,

- Job durumunu ve log dosyalarını takip etme,

- Üretilen JSON sonuç dosyasını inceleme,

- Yarışma kapsamında beklenen solution.py input/output formatını anlama.

Bu hands-on çalışmasında örnek olarak GHZ devresi kullanılacaktır. GHZ örneği, yarışma probleminin doğrudan kendisi değildir; ancak TRUBA üzerinde kuantum devresi çalıştırma, GPU destekli simülasyon kullanma, SLURM ile iş gönderme ve çıktı dosyası üretme adımlarını göstermek için bir örnektir.

2. Çalışma Klasörü ve Dosya Yapısı

Yarışma kapsamında tüm dosyalar TRUBA üzerindeki scratch alanında tutulacaktır. Katılımcılar öncelikle kendi kullanıcı alanlarında bir çalışma klasörü oluşturmalıdır.

Örnek klasör hazırlığı:

cd /arf/scratch/$USER

mkdir -p ghz_demo

cd ghz_demo

mkdir -p logs results

Bu klasör yapısında:

- logs klasörü, SLURM tarafından üretilen .out ve .err log dosyalarını tutmak için kullanılacaktır.

- results klasörü, Python kodu tarafından üretilecek JSON sonuç dosyalarını tutmak için kullanılacaktır.

- Python kodları ve SLURM dosyaları ana çalışma klasörü içinde yer alacaktır.

Örnek dosya yapısı aşağıdaki gibi olacaktır:

ghz_hands_on/

├── ghz_aer_gpu.py

├── run_ghz.slurm

├── logs/

└── results/

3. GHZ (Greenberger-Horne-Zeilinger) Devresi Nedir?

GHZ durumu, birden fazla kübit arasında dolaşıklık oluşturan temel kuantum durumlarından biridir. Bu örnekte GHZ devresi, kuantum devre kurulumunu ve ölçüm sonuçlarının yorumlanmasını göstermek için kullanılacaktır.

3 kübitlik standart GHZ durumu şu şekilde ifade edilir:

$$|GHZ_3\rangle = \frac{|000\rangle + |111\rangle}{\sqrt{2}}$$

Genel olarak n kübitlik GHZ durumu:

$$|GHZ_n\rangle = \frac{|00\ldots0\rangle + |11\ldots1\rangle}{\sqrt{2}}$$

Bu ifade, ölçüm yapıldığında ideal durumda sistemin yalnızca iki sonuçtan birini üretmesinin beklendiği anlamına gelir:

- Tüm kübitler 0 ölçülür: 000

- Tüm kübitler 1 ölçülür: 111

Aynı mantık daha fazla kübit için de geçerlidir.

GHZ devresinin temel mantığı şu adımlardan oluşur:

1. İlk kübite Hadamard kapısı uygulanır.

2. İlk kübit süperpozisyon durumuna alınır.

3. CNOT kapıları ardışık kübitler arasında uygulanarak dolaşıklık tüm devreye yayılır.

4. Ölçüm sonucunda tüm kübitlerin birlikte 0 veya birlikte 1 olması beklenir.

4. GHZ Python Kodunun Temel Mantığı

Bu bölümde, GHZ devresini Qiskit kullanarak oluşturan ve TRUBA üzerinde GPU destekli olarak çalıştırılabilecek örnek bir Python kodu verilecektir.

Kodun temel amacı:

- Kullanıcının belirlediği kübit sayısına göre GHZ devresi oluşturmak,

- Devreyi Qiskit Aer GPU simulator üzerinde çalıştırmak,

- Ölçüm sonuçlarını almak,

- Fidelity değerini hesaplamak,

- Sonuçları JSON dosyası olarak kaydetmektir.

Bu örnekte fidelity değeri, ölçüm sonucunda beklenen iki GHZ çıktısının oranı olarak hesaplanmaktadır. Yani 000...0 ve 111...1 sonuçlarının toplam oranı fidelity olarak alınmıştır.

1. Çalışma klasörünü hazırlama
   cd /arf/scratch/USER
   mkdir -p ghz_hands_on
   cd ghz_hands_on
   mkdir -p logs results

2. Python dosyasını oluşturma:
   Dosyayı nano ile açıyoruz:

   “nano ghz.py”

```
import json
import os
import time

from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
```

```
def create_ghz_circuit(n_qubits):
    # Verilen kübit sayısına göre GHZ devresi hazırlanır.
    circuit = QuantumCircuit(n_qubits, n_qubits)

    # İlk kübit süperpozisyon durumuna alınır.
    circuit.h(0)

    # Diğer kübitler CNOT kapıları ile devreye bağlanır.
    for i in range(n_qubits - 1):
        circuit.cx(i, i + 1)

    # Tüm kübitler ölçülür.
    circuit.measure(range(n_qubits), range(n_qubits))

    return circuit


def run_ghz(n_qubits, shots, job_id, job_name, node_list):
    # GHZ devresi oluşturulur.
    circuit = create_ghz_circuit(n_qubits)

    # Devre Qiskit Aer GPU simulator üzerinde çalıştırılır.
    simulator = AerSimulator(
        method="statevector",
        device="GPU",
        precision="double",
        cusvaer_enable=False,
        cuStateVec_enable=False,
        blocking_enable=True,
        blocking_qubits=29
    )

    circuit = transpile(circuit, simulator)

    print("GHZ started")
    print("Qubits:", n_qubits)
    print("Shots:", shots)

    start_time = time.time()

    result = simulator.run(circuit, shots=shots).result()
    counts = result.get_counts()

    elapsed_seconds = time.time() - start_time

    zero_state = "0" * n_qubits
    one_state = "1" * n_qubits

    # GHZ için beklenen iki çıktı: 000...0 ve 111...1
    fidelity = (counts.get(zero_state, 0) + counts.get(one_state, 0)) / shots

    output_data = {
        "job_id": job_id,
        "job_name": job_name,
        "node_list": node_list,
        "n_qubits": n_qubits,
        "shots": shots,
        "counts": counts,
        "fidelity": fidelity,
```

```
        "elapsed_seconds": elapsed_seconds
    }

    # Sonuç dosyaları results klasörüne yazılır.
    os.makedirs("results", exist_ok=True)

    output_file = f"results/ghz_{n_qubits}q_{job_id}.json"

    with open(output_file, "w") as f:
        json.dump(output_data, f, indent=2)

    print("Counts:", counts)
    print("Fidelity:", fidelity)
    print("Result file:", output_file)


def main():
    # SLURM tarafından verilen job bilgileri alınır.
    job_id = os.environ.get("SLURM_JOB_ID", "local")
    job_name = os.environ.get("SLURM_JOB_NAME", "local")
    node_list = os.environ.get("SLURM_NODELIST", "local")

    shots = 1024
    qubit_list = [4, 25, 30]

    print("Job ID:", job_id)
    print("Job name:", job_name)
    print("Node:", node_list)

    # Aynı job içinde farklı kübit sayıları denenir.
    for n_qubits in qubit_list:
        run_ghz(n_qubits, shots, job_id, job_name, node_list)

    print("All GHZ runs finished")


if __name__ == "__main__":
    main()
```

Kod dosya içerisine yapıştırılır. Sonra:
CTRL + O → kaydet
Enter → dosya adını onayla
CTRL + X → çık

5. SLURM İş Dosyasının Hazırlanması

nano run.slurm

```
#!/bin/bash
#SBATCH --job-name=ghz_demo
#SBATCH --partition=kolyoz-cuda
#SBATCH --reservation=qalgtest
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=64
#SBATCH --gres=gpu:4
#SBATCH --time=00:30:00
#SBATCH --output=ghz_demo_%j.out
#SBATCH --error=ghz_demo_%j.err

set -e

# Çalışma klasörü
PROJECT_DIR=/arf/scratch/$USER/ghz_demo

# Kullanılacak container
SIF=/arf/sw/containers/quantumcomp/cuquantum-25.11-cuda13.0.1.sif

cd "$PROJECT_DIR"

echo "GHZ demo job started"
echo "Job ID: $SLURM_JOB_ID"
echo "Job name: $SLURM_JOB_NAME"
echo "Node: $SLURM_NODELIST"

# Job'un GPU node üzerinde çalıştığını görmek için
nvidia-smi

# Python kodu container içinde çalıştırılır.
apptainer exec --nv \
  -B "$PROJECT_DIR:$PROJECT_DIR" \
  "$SIF" \
  python "$PROJECT_DIR/ghz.py"

echo "GHZ demo job finished"
echo "Result files:"
ls -lh "$PROJECT_DIR/results"
```

Not: Bu örnekte çalışma dizini kullanıcı hesabına göre /arf/scratch/$USER/ghz_demo olarak oluşturulmuştur. Yarışma ortamında ortak hackathon hesabı kullanılacağından $USER yerine her gruba özel olarak tanımlanan çalışma klasörü kullanılacaktır.

Kod dosya içerisine yapıştırılır. Sonra:
CTRL + O → kaydet
Enter → dosya adını onayla
CTRL + X → çık

Dosyanın oluştuğunu kontrol etmek için: ls -lh run.slurm

Bu SLURM dosyası, TRUBA üzerinde 1 node ve 4 GPU talep eder. Örnek olarak, 4, 25 ve 30 kübitlik GHZ devreleri sırayla çalıştırılır ve sonuçlar results klasörüne JSON dosyası olarak yazılır.

Job gönderme: sbatch run_ghz.slurm

Ekranda oluşacak çıktı: Submitted batch job 1513000
İş Durumunu kontrol etme: squeue -j 1513000

Log dosyasını izleme:

SLURM dosyasında çıktı ve hata loglarının nereye yazılacağı aşağıdaki satırlarla belirlenir:
#SBATCH --output=ghz_demo_%j.out
#SBATCH --error=ghz_demo_%j.err
Buradaki %j ifadesi, SLURM tarafından otomatik olarak job numarası ile değiştirilir. Örneğin job numarası 1517267 ise oluşacak log dosyaları şu şekilde olur:
ghz_demo_1517267.out
ghz_demo_1517267.err
.out dosyası programın ekrana yazdırdığı normal çıktıları tutar. .err dosyası ise çalışma sırasında oluşan hata mesajlarını tutar.

Logu izlemek için: tail -f ghz_demo_1517267.out

İş bittikten sonra durum kontrolü:

sacct -j 1517267 --format=JobID,JobName,State,Elapsed,NodeList,ExitCode

6. JSON Sonuçlarını İncelme:

ls -lh results
Bu komut results klasörü içinde üretilen sonuç dosyalarını listeler. Örneğin :
ghz_4q.json
ghz_25q.json
ghz_30q.json

JSON dosyasının içeriğini daha okunabilir görmek için:
python -m json.tool results/ghz_4q.json
python -m json.tool results/ghz_25q.json
python -m json.tool results/ghz_30q.json

7. Sık Karşılaşılan Hatalar ve Çözümleri

Yarışmacılara hazır bir SLURM şablonu verileceğinden GPU, container ve simülatör ayarları şablon içerisinde tanımlanmış olacaktır. Bu nedenle yarışmacıların yalnızca değiştirilmesi gerektiği belirtilen alanları düzenlemesi önerilmektedir.

Hata veya durum	Olası neden	Çözüm
`cd: ... No such file or directory`	Grup klasörünün yolu yanlış yazılmıştır.	Size bildirilen grup klasörünün tam yolunu kontrol edin. `pwd` ve `ls` komutlarından yararlanabilirsiniz.
`sbatch: error: Unable to open file`	SLURM dosyasının adı yanlış yazılmış veya komut farklı bir klasörde çalıştırılmıştır.	`ls -lh` ile dosya adını kontrol edin ve `sbatch run_ghz.slurm` komutunu dosyanın bulunduğu klasörde çalıştırın.
Python dosyası bulunamıyor	Python dosyasının adı değiştirilmiş, yanlış yazılmış veya dosya başka bir klasöre kaydedilmiştir.	`ls -lh ghz_aer_gpu.py` komutuyla dosyanın bulunduğunu kontrol edin.
İş `PENDING (Priority)` durumunda	İş, kaynakların uygun hâle gelmesini veya çalışma sırasını beklemektedir.	Bu bir hata değildir. İşin durumunu `squeue -j <JOB_ID>` komutuyla takip edin.
`squeue` çıktı göstermiyor	İş tamamlanmış, iptal edilmiş veya hata alarak sonlanmış olabilir.	İşin son durumunu `sacct -j <JOB_ID>` komutuyla kontrol edin.
`Invalid job id specified`	İş numarası yanlış yazılmış veya tamamlanan iş `squeue` ile görüntülenmeye çalışılmıştır.	Gerçek iş numarasını kontrol edin. Tamamlanan işler için `sacct` komutunu kullanın.
`sacct: error: Unknown arguments`	`--format` alanındaki virgüllerden sonra boşluk bırakılmıştır.	Alanları boşluk bırakmadan yazın: `--format=JobID,JobName,State,Elapsed,NodeList,ExitCode`
Log dosyası bulunamıyor	Yanlış iş numarası veya yanlış dosya	SLURM tarafından verilen gerçek iş numarasını ve şablondaki log dosyası konumunu kontrol edin.

	yolu kullanılmıştır.	
JSON dosyası oluşmadı	İş hata almış veya program sonuç üretme aşamasına ulaşamamıştır.	Önce `sacct` ile iş durumunu, ardından `.err` uzantılı hata dosyasını kontrol edin.
`python -m json.tool` dosyayı bulamıyor	JSON dosyasının adı veya yolu yanlış yazılmıştır.	Önce `ls -lh results` komutuyla oluşan dosyaları listeleyin.
`Permission denied`	Başka bir grubun klasörüne erişilmeye çalışılmış olabilir.	Yalnızca grubunuza tahsis edilen çalışma klasörünü kullanın.
SSH bağlantısı kesildi	İnternet veya VPN bağlantısı kopmuş olabilir.	TRUBA'ya yeniden bağlanın. Gönderilmiş SLURM işi bağlantıdan bağımsız çalışmaya devam eder.

8. Requirements

Çalışma kapsamında kişisel bilgisayarlarınızda çalışma yapabilmek adına ihtiyaç duyacağınız kütüphane ve sürümleri:

qiskit==1.4.5
qiskit-aer==0.15.1
qiskit-optimization==0.6.1
numpy
scipy
networkx
matplotlib

9. Ek Python Paketleri Kullanımı için Dosyalar

TRUBA'da kullanılan Apptainer container, temel çalışma ortamını sağlar. Ancak bazı projelerde container içinde hazır bulunmayan ek Python paketlerine ihtiyaç olabilir. Bu durumda container içeriğini değiştirmek yerine, ek paketler kullanıcı çalışma dizini altında pydeps adlı(klasör adı değiştirilebilir) bir klasöre kurulabilir.
pydeps, proje için gerekli ek Python kütüphanelerinin tutulduğu klasördür. SLURM dosyasında bu klasör PYTHONPATH ile tanıtıldığında Python, çalıştırma sırasında bu paketleri görebilir.

TRUBA'da örnek kullanım:

Önce proje klasörüne geçilir:

cd /arf/scratch/$USER/proje_adi
pydeps klasörü oluşturulur:
mkdir -p pydeps

Gerekli paketler bu klasöre kurulur:

apptainer exec -B "$PWD:$PWD" /arf/sw/containers/quantumcomp/cuquantum-25.11-cuda13.0.1.sif python -m pip install --target "$PWD/pydeps" qiskit-optimization==0.6.1

SLURM dosyasında Python çalıştırılmadan önce şu satır eklenir:

export PYTHONPATH="$PROJECT_DIR/pydeps:$PYTHONPATH"

Böylece container içinde doğrudan bulunmayan paketler, proje klasöründeki pydeps üzerinden kullanılabilir.