qpu

May 2, 2023

1 Welcome to the Quantum Parallel Universe

1.1 Initial Setup

1.1.1 Immutable Imports

[1]:
```
import math
import re
from IPython.display import Latex
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.circuit import Qubit
```

1.1.2 Globals

Manually Managed Variables & Imports

[2]:
```
# number of qubits: int
N = 15

# IBMQ Mock Backend (https://qiskit.org/documentation/stable/0.42/apidoc/
↪providers_fake_provider.html#fake-v1-backends)
from qiskit.providers.fake_provider import FakeCairo
backend = { 'device': FakeCairo() }
```

Automatically Managed Variables

[3]:
```
# linear GHZ container
linear = {
  'circuit': { 'draw': None, 'execute': None},
  'transpiled': None,
  'job': None,
  'result': None,
  'time': None,
  'error': { '0': None, '1': None }
}

# logarithmic GHZ container
log = {
```

```
  'circuit': { 'draw': None, 'execute': None},
  'transpiled': None,
  'job': None,
  'result': None,
  'time': None,
  'error': { '0': None, '1': None }
}

# ideal shots per state
isps = 512

# IBMQ Mock Backend
if N > 0:
  backend['name'] = re.sub(r'(_|fake|v\d)', ' ', backend['device'].backend_name.
↪lower()).title()
  backend['num_qubits'] = backend['device'].configuration().num_qubits
  backend['simulator'] = AerSimulator.from_backend(backend['device'])
else:
  raise RuntimeError(msg=f"Invalid N={N}, must be 0 < N <␣
↪{backend['num_qubits']}")
```

1.2 Generate $|\mathrm{GHZ}_N\rangle$ Circuits1

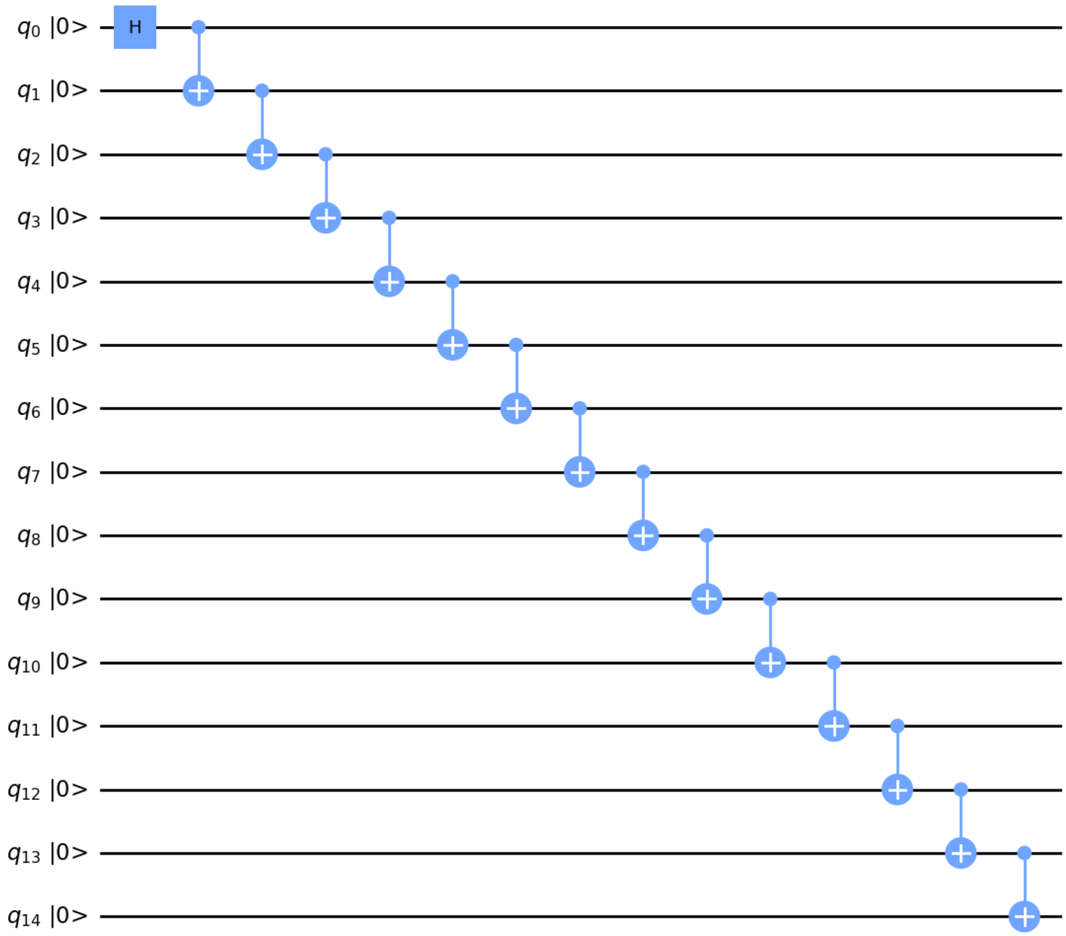
1.2.1 Generate Linear Time Complexity Circuits for $|\mathrm{GHZ}_N\rangle$

[4]:
```
def linear_complexity_GHZ(N: int, measure: bool = True) -> QuantumCircuit:
  if not isinstance(N, int):
    raise TypeError("Only integer arguments accepted.")
  if N < 1:
    raise ValueError("There must be one or more qubits.")

  c = QuantumCircuit(N)
  c.h(0)
  for i in range(1, N):
    c.cx(i-1, i)
  if measure:
    c.measure_active()
  return c
```

[5]:
```
linear['circuit']['execute'] = linear_complexity_GHZ(N)
linear['circuit']['draw'] = linear_complexity_GHZ(N, False)
linear['circuit']['draw'].draw(output='mpl', fold=-1, initial_state=True)
```

[5]:



1.2.2 Generate Logaritmic Complexity Circuits for $|\mathrm{GHZ}_{2^m}\rangle$

[6]:
```
def _log_complexity_GHZ(m: int) -> QuantumCircuit:
  if not isinstance(m, int):
    raise TypeError("Only integer arguments accepted.")
  if m < 0:
    raise ValueError("`m` must be at least 0 (evaluated 2^m).")

  if m == 0:
    c = QuantumCircuit([Qubit()])
    c.h(0)
  else:
    c = _log_complexity_GHZ(m - 1)
    for i in range(c.num_qubits):
      c.add_bits([Qubit()])
      new_qubit_index = c.num_qubits - 1
```

```
    c.cx(i, new_qubit_index)
  return c
```

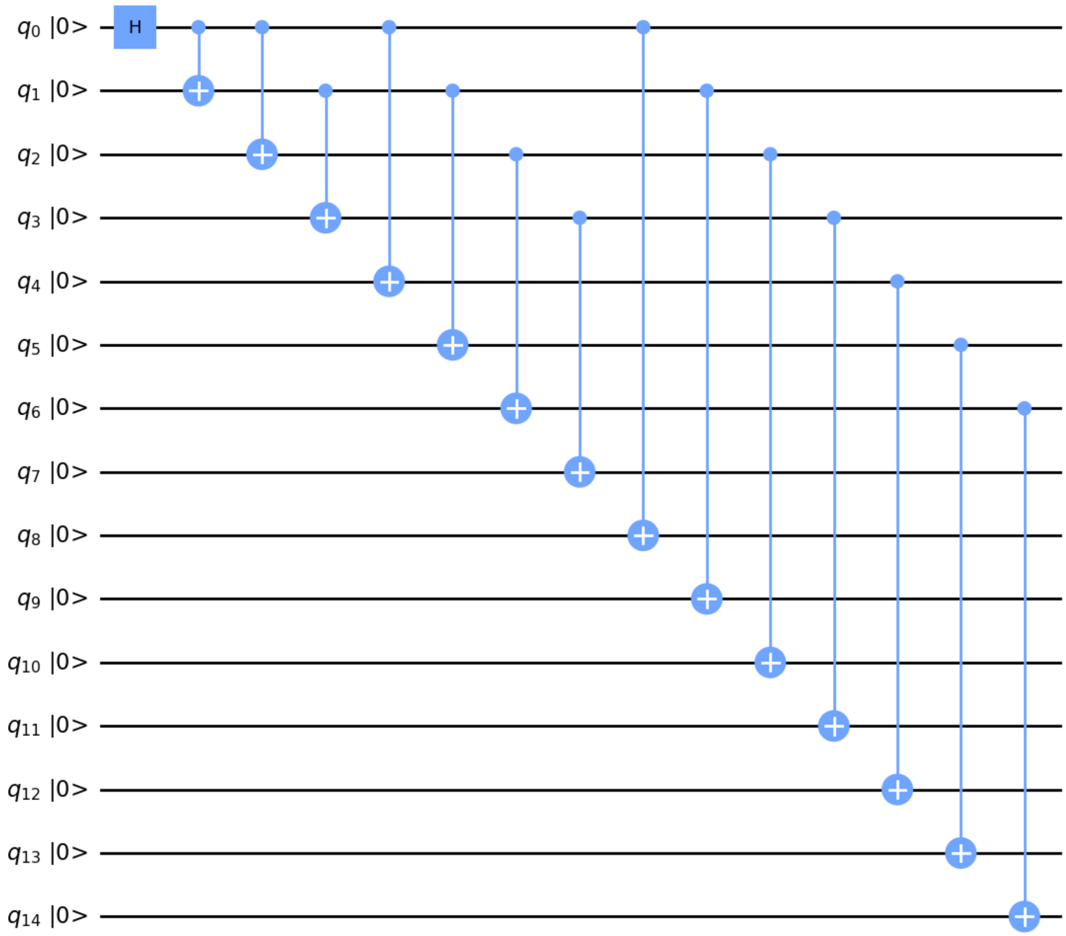
1.2.3 Generate Logaritmic Complexity Circuits for $|\mathrm{GHZ}_N\rangle$

[7]:
```
def log_complexity_GHZ(N: int, measure: bool = True) -> QuantumCircuit:
  if not isinstance(N, int):
    raise TypeError("Only an integer argument is accepted.")
  if N < 1:
    raise ValueError("There must be one or more qubits.")

  m = math.ceil(math.log2(N))
  num_qubits_to_erase = 2**m - N
  old_circuit = _log_complexity_GHZ(m=m)
  new_num_qubits = old_circuit.num_qubits - num_qubits_to_erase
  new_circuit = QuantumCircuit(new_num_qubits)
  for gate in old_circuit.data:
    qubits_affected = gate.qubits
    if all(old_circuit.find_bit(qubit).index < new_num_qubits for qubit in␣
↪qubits_affected):
      new_circuit.append(gate[0], [old_circuit.find_bit(qubit).index for qubit␣
↪in qubits_affected])
  if measure:
    new_circuit.measure_active()
  return new_circuit
```

[8]:
```
log['circuit']['execute'] = log_complexity_GHZ(N)
log['circuit']['draw'] = log_complexity_GHZ(N, False)
log['circuit']['draw'].draw(output='mpl', fold=-1, initial_state=True)
```

[8]:



1.3 Computational Cost Analysis

1.3.1 Cost of Hadamard (H) Gate2

[9]:
```
H_cost = 2
```

1.3.2 Cost of CNOT (CX) Gate2

[10]:
```
CX_cost = 5
```

1.3.3 Program Cost

[11]:
```python
T_cost = ((N - 1) * CX_cost) + H_cost
Latex(f"""\\begin{{equation*}}{T_cost}\\end{{equation*}}""")
```

[11]:

$$72$$

1.3.4 Program Sections (s)

[12]:
```python
def program_sections() -> int:
  init = [ 0, 1, 2 ]
  i = len(init)
  k = 1
  while len(init) <= N:
    init += [i] * 2**k
    i += 1
    k += 1
  return init[N]
```

[13]:
```python
s = program_sections()
Latex(f"""\\begin{{equation*}}{s}\\end{{equation*}}""")
```

[13]:

$$5$$

1.3.5 Cost per Section

[14]:
```python
def cost_per_section() -> list:
  init = [ H_cost ]
  _N = N - 1
  k = 0
  while len(init) < (s - 1):
    init.append(CX_cost * 2**k)
    _N -= 2**k
    k += 1
  if _N > 0:
    init.append(CX_cost * _N)
  return init
```

[15]:
```python
section_cost_list = cost_per_section()
Latex(f"""\\begin{{equation*}}{section_cost_list}\\end{{equation*}}""")
```

[15]:

$$[2, 5, 10, 20, 35]$$

1.4 Parallel v. Sequential Analysis

1.4.1 Gates Per Section

[16]:
```
def gates_per_section() -> list:
  if s == 1:
    return [1]
  elif s == 2:
    return [1, 1]
  else:
    init = [1, 1]
    for i in range(len(init), s):
      init.append(int(section_cost_list[i] / CX_cost))
    return init
```

[17]:
```
num_gates_list = gates_per_section()
Latex(f"""\\begin{{equation*}}{num_gates_list}\\end{{equation*}}""")
```

[17]:

$$[1, 1, 2, 4, 7]$$

1.4.2 Percent Sequential

[18]:
```
def add_sequential_portions() -> float:
  seq = 0
  for cost, gates in zip(section_cost_list, num_gates_list):
    seq += (1 / gates) * (cost / T_cost)
  return seq
```

[19]:
```
sequential_portion = add_sequential_portions()
Latex(f"""\\begin{{equation*}}{sequential_portion * 100}\%\\end{{equation*}}""")
```

[19]:

$$30.555555555555557\%$$

1.4.3 Percent Parallel

[20]:
```
def add_parallel_portions() -> float:
  par = 0
  for cost, gates in zip(section_cost_list, num_gates_list):
    par += ( (gates - 1) / gates ) * (cost / T_cost)
  return par
```

[21]:
```
parallel_portion = add_parallel_portions()
Latex(f"""\\begin{{equation*}}{parallel_portion * 100}\%\\end{{equation*}}""")
```

[21]:

$$69.44444444444444\%$$

1.5 Quantum Simulation

1.5.1 Device

[22]:
```
Latex(f"""\\begin{{equation*}}
        \\text{{{backend['name']} ({backend['num_qubits']} qubits)}}
        \\end{{equation*}}""")
```

[22]:

Cairo (27 qubits)

1.5.2 Transpile Circuits3

[23]:
```
linear['transpiled'] = transpile(linear['circuit']['execute'],
                                 backend['simulator'],
                                 scheduling_method="asap",
                                 optimization_level=0)
```

[24]:
```
log['transpiled'] = transpile(log['circuit']['execute'],
                              backend['simulator'],
                              scheduling_method="asap",
                              optimization_level=0)
```

1.5.3 Run Simulations

[25]:
```
linear['job'] = backend['device'].run(linear['transpiled'])
```

[26]:
```
log['job'] = backend['device'].run(log['transpiled'])
```

1.5.4 Block for Results

[27]:
```
linear['result'] = linear['job'].result()
```

[28]:
```
log['result'] = log['job'].result()
```

1.6 Speed-Up Analysis

1.6.1 Run-Times

Linear

[29]:
```
linear['time'] = linear['result'].time_taken
Latex(f"""\\begin{{equation*}}{linear['time']}\\space\\text{{seconds}}\\end{{equation*}}""")
```

[29]:

24.292431592941284seconds

Log

[30]:
```
log['time'] = log['result'].time_taken
Latex(f"""\\begin{{equation*}}{log['time']}\\space\\text{{seconds}}\\end{{equation*}}""")
```

[30]:

$$9.33341646194458\text{seconds}$$

1.6.2 Theoretical Max Speed-Up (Amdahl's Law4)

$$\lim_{F \to \infty} S_{\text{latency}} = \frac{1}{S_{\text{eq}} + \frac{P}{F}} = \frac{1}{S_{\text{eq}}}$$

- S_{eq} represents the portions of the program running sequentially
- P represents the portions of the program running in parallel
- F represents the level of concurrency (i.e. number of cores in classical computing)

[31]:
```
Latex(f"""\\begin{{equation*}}{1 / sequential_portion}\\end{{equation*}}""")
```

[31]:

$$3.2727272727272725$$

1.6.3 Observed Speed-Up Factor (S_{latency})

[32]:
```
S_latency = linear['time'] / log['time']
Latex(f"""\\begin{{equation*}}{S_latency}\\end{{equation*}}""")
```

[32]:

$$2.602737346178599$$

1.6.4 Approximated Level of Concurrency (F)

$$F = \frac{P \cdot S_{\text{latency}}}{1 - S_{\text{eq}} \cdot S_{\text{latency}}}$$

[33]:
```
F = (parallel_portion * S_latency) / (1 - (sequential_portion * S_latency))
Latex(f"""\\begin{{equation*}}{F}\\end{{equation*}}""")
```

[33]:

$$8.828956848467138$$

1.7 Error Analysis

1.7.1 Linear Error Percentage

State $|0\rangle$

[34]:
```
try:
```

```
    linear['error']['0'] = abs((linear['result'].get_counts()['0' * N] - isps) /␣
  ↪isps)
except KeyError:
    linear['error']['0'] = 1
Latex(f"""\\begin{{equation*}}{linear['error']['0'] *␣
  ↪100}\%\\end{{equation*}}""")
```

[34]:

$$63.8671875\%$$

State $|1\rangle$

[35]:
```
try:
    linear['error']['1'] = abs((linear['result'].get_counts()['1' * N] - isps) /␣
  ↪isps)
except KeyError:
    linear['error']['1'] = 1
Latex(f"""\\begin{{equation*}}{linear['error']['1'] *␣
  ↪100}\%\\end{{equation*}}""")
```

[35]:

$$91.9921875\%$$

1.7.2 Logarithmic Error Percentage

State $|0\rangle$

[36]:
```
try:
    log['error']['0'] = abs((log['result'].get_counts()['0' * N] - isps) / isps)
except KeyError:
    log['error']['0'] = 1
Latex(f"""\\begin{{equation*}}{log['error']['0'] * 100}\%\\end{{equation*}}""")
```

[36]:

$$63.671875\%$$

State $|1\rangle$

[37]:
```
try:
    log['error']['1'] = abs((log['result'].get_counts()['1' * N] - isps) / isps)
except KeyError:
    log['error']['1'] = 1
Latex(f"""\\begin{{equation*}}{log['error']['1'] * 100}\%\\end{{equation*}}""")
```

[37]:

$$89.6484375\%$$

1.8 References

1. Cruz, Diogo, Romain Fournier, Fabien Gremion, Alix Jeannerot, Kenichi Komagata, Tara Tosic, Jarla Thiesbrummel, et al. (2018). Efficient Quantum Algorithms for GHZ and W States, and Implementation on the IBM Quantum Computer. ArXiv. 1-2.

2. Lee, Soonchil & Lee, Seong-Joo & Kim, Taegon & Lee, Jae-Seung & Biamonte, Jacob & Perkowski, Marek. (2006). The cost of quantum gate primitives. Journal of Multiple-Valued Logic and Soft Computing. 12. 571.

3. Scheduling Methods

4. Amdahl's Law Definition