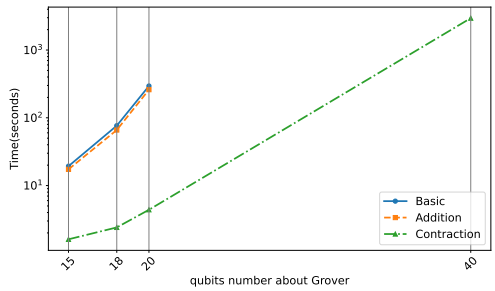
软件测试

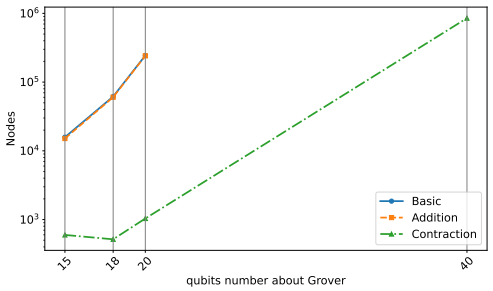
高丁超

1 Python for Image Computation

- 软件的工作目标:
  - 输入电路与系统初始状态，输出下一步的系统状态。
- 运行环境：
  - Python >= 3.9.0
  - Numpy >= 1.20.0
  - Qiskit >= 0.25.0
  - Graphviz >= 0.20.0
- 适应的工作对象:
  - 复杂线路如 Grover，QRW 不超过二十比特。
  - 简单线路如 GHZ，不超过二百比特。
- 测试样例:
  - Grover
  - QFT
  - BV
  - GHZ
  - QRW
- 自我测试实验报告:
  - 图1表示了对 Grover 搜索算法运行不同 image computation 算法的资源对比。图2表示了对 quantum Fourier transform（QFT）算法运行不同 image computation 算法的资源对比。图3表示了对 Bernstein-Vazirani（BV）算法运行不同 image computation 算法的资源对比。图4表示了对 Greenberger-Horne-Zeilinger （GHZ）状态制备电路运行不同 image computation 算法的资源对比。图5表示了对在 2^n 环上的 quantum random walk （QRW）算法运行不同 image computation 算法的资源对比。
  - 表1给出了在不同电路拆分技术下具体的各类算法的计算时间，单位为秒，max node 表示计算过程中 TDD 的节点最大个数，- 表示超时。其中 addition 和 contraction 表示不同的电路拆分技术。

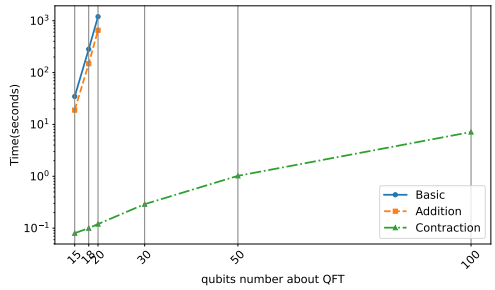


(a) 对 Grover 算法应用不同电路拆分技术的时间对比

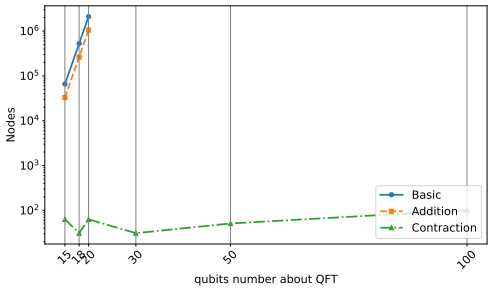


(b) 对 Grover 算法应用不同电路拆分技术的最大节点对比

Figure 1: 对 Grover 算法运行 image computation 时不同电路拆分技术的资源对比

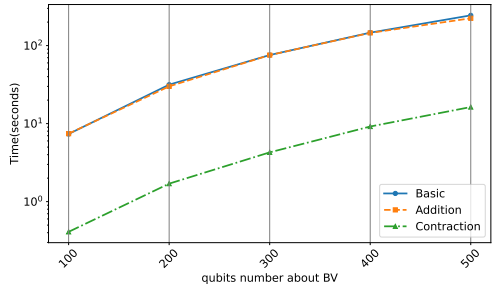


(a) 对 QFT 算法应用不同电路拆分技术的时间对比

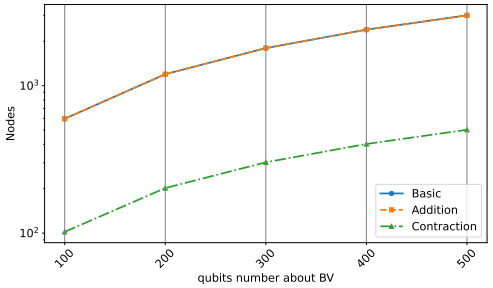


(b) 对 QFT 算法应用不同电路拆分技术的最大节点对比

Figure 2: 对 QFT 算法运行 image computation 时不同电路拆分技术的资源对比



(a) 对 BV 算法应用不同电路拆分技术的时间对比



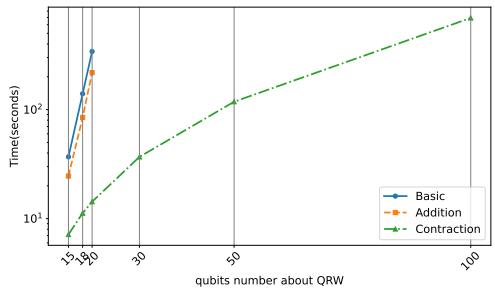
(b) 对 BV 算法应用不同电路拆分技术的最大节点对比

Figure 3: 对 BV 算法运行 image computation 时不同电路拆分技术的资源对比

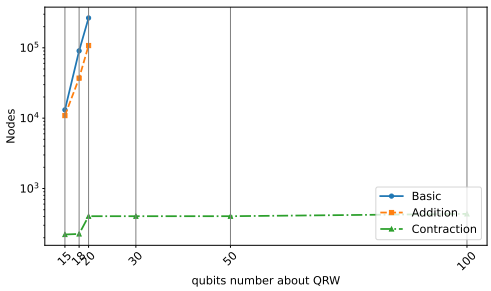
(a) 对 GHZ 算法应用不同电路拆分技术的时间对比

(b) 对 GHZ 算法应用不同电路拆分技术的最大节点对比

Figure 4: 对 GHZ 算法运行 image computation 时不同电路拆分技术的资源对比



(a) 对 QRW 算法应用不同电路拆分技术的时间对比



(b) 对 QRW 算法应用不同电路拆分技术的最大节点对比

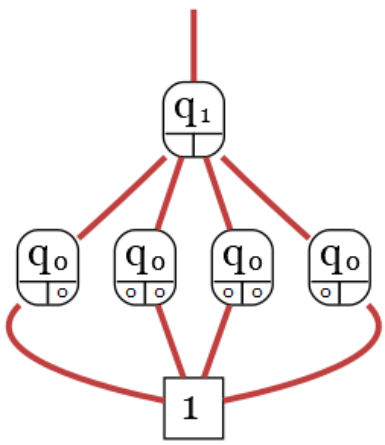
Figure 5: 对 QRW 算法运行 image computation 时不同电路拆分技术的资源对比

Benchmark	basic		addition		contraction	
	time	max #node	time	max #node	time	max #node
Grover_15	19.33	15785	17.35	15099	1.61	597
Grover_18	76.47	61694	66.02	60332	2.41	516
Grover_20	294.65	243946	259.87	241240	4.39	1036
Grover_40	-		-		2953.57	851973
QFT_15	34.64	65536	18.88	32770	0.08	63
QFT_18	282.12	524288	148.13	262146	0.10	31
QFT_20	1199.21	2097152	655.19	1048578	0.12	63
QFT_30	-		-		0.29	31
QFT_50	-		-		1.02	51
QFT_100	-		-		7.14	101
BV_100	7.36	596	7.43	596	0.41	102
BV_200	31.57	1196	30.03	1196	1.70	202
BV_300	75.66	1796	75.56	1796	4.28	302
BV_400	146.47	2396	145.40	2396	9.18	402
BV_500	244.15	2996	223.90	2996	16.31	502
GHZ_100	0.38	595	0.13	301	0.18	200
GHZ_200	0.72	1195	0.37	601	0.48	400
GHZ_300	1.29	1795	0.62	901	0.80	600
GHZ_400	2.03	2395	1.00	1201	1.26	800
GHZ_500	2.96	2995	1.45	1501	1.72	1000
QRW_15	36.86	13122	24.59	10882	7.16	222
QRW_18	139.76	90538	84.69	37064	11.23	226
QRW_20	341.05	265614	218.29	107714	14.31	404
QRW_30	-		-		36.82	404
QRW_50	-		-		118.08	404
QRW_100	-		-		692.08	436

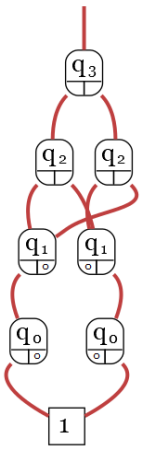
Table 1: 对不同测试实验应用 image computation

2 C++ for TDD

- 软件的工作目标:
  - 输入电路，输出电路的 TDD 表示。
- 运行环境：
  - C++ Standard 17
  - CMake >= 3.20.0
  - XTL >= 0.7.5
  - XTENSOR >= 0.24.0
  - Graphviz >= 2.43.0
  - XTENSOR-Python >= 0.26.0
  - Pybind11 >= 2.12.0
  - Numpy >= 1.20.0
- 适应的工作对象:
  - 小型电路
- 测试样例:
  - 基础门，如 CNOT, H, S, Swap
  - 随机生成的小规模电路
- 自我测试实验报告:
  - 所有基础门正确转化为 TDD 表示
  - 大部分小规模电路正确转换，部分出现了内存占用过大的问题。
  - 支持任意维度张量，例如对双比特 CNOT 门，既可以按图6a中的张量维度为 4，即按照索引为 q_1, q_0 进行表示。也可以按图6b中的张量维度为 4，即按照索引为 q_3, q_2, q_1, q_0 进行表示。这样的设计大大提高了 TDD 的表示能力，为更复杂系统的验证提供了基础。



(a) 张量维度为 4 的 CNOT 门的 TDD 表示



(b) 张量维度为 2 的 CNOT 门的 TDD 表示

Figure 6: C 语言版的 TDD 支持任意维度的例子