Automating NISQ Application Design w/ Meta Quantum Circuits with Constraints

Xiaodi Wu
QuICS & UMD

Joint work with Haowei Deng, Yuxiang Peng, and Mike Hicks

UNIVERSITY OF MARYLAND

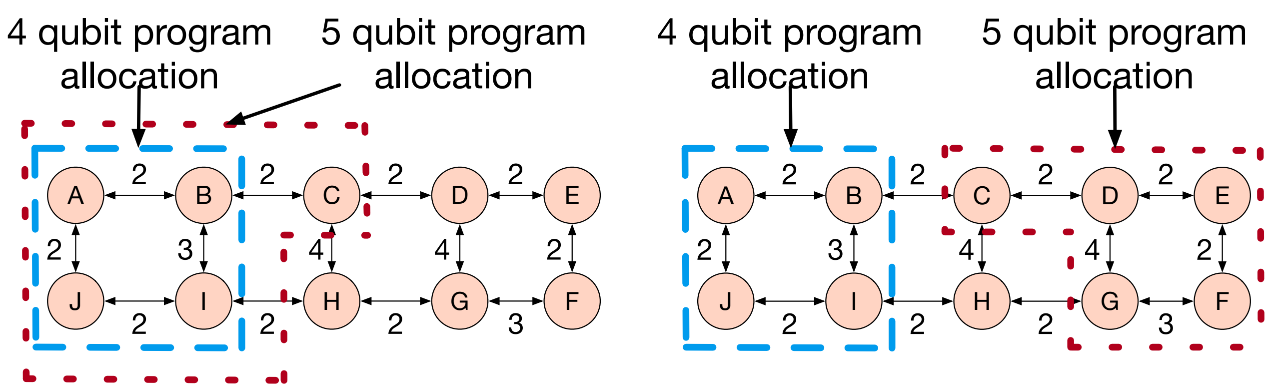
JOINT CENTER FOR QUANTUM INFORMATION AND COMPUTER SCIENCE

Features of NISQ Application Design

NISQ machines: very *restricted* hardware resources, where precisely controllable qubits are *expensive, error-prone, and scarce*.

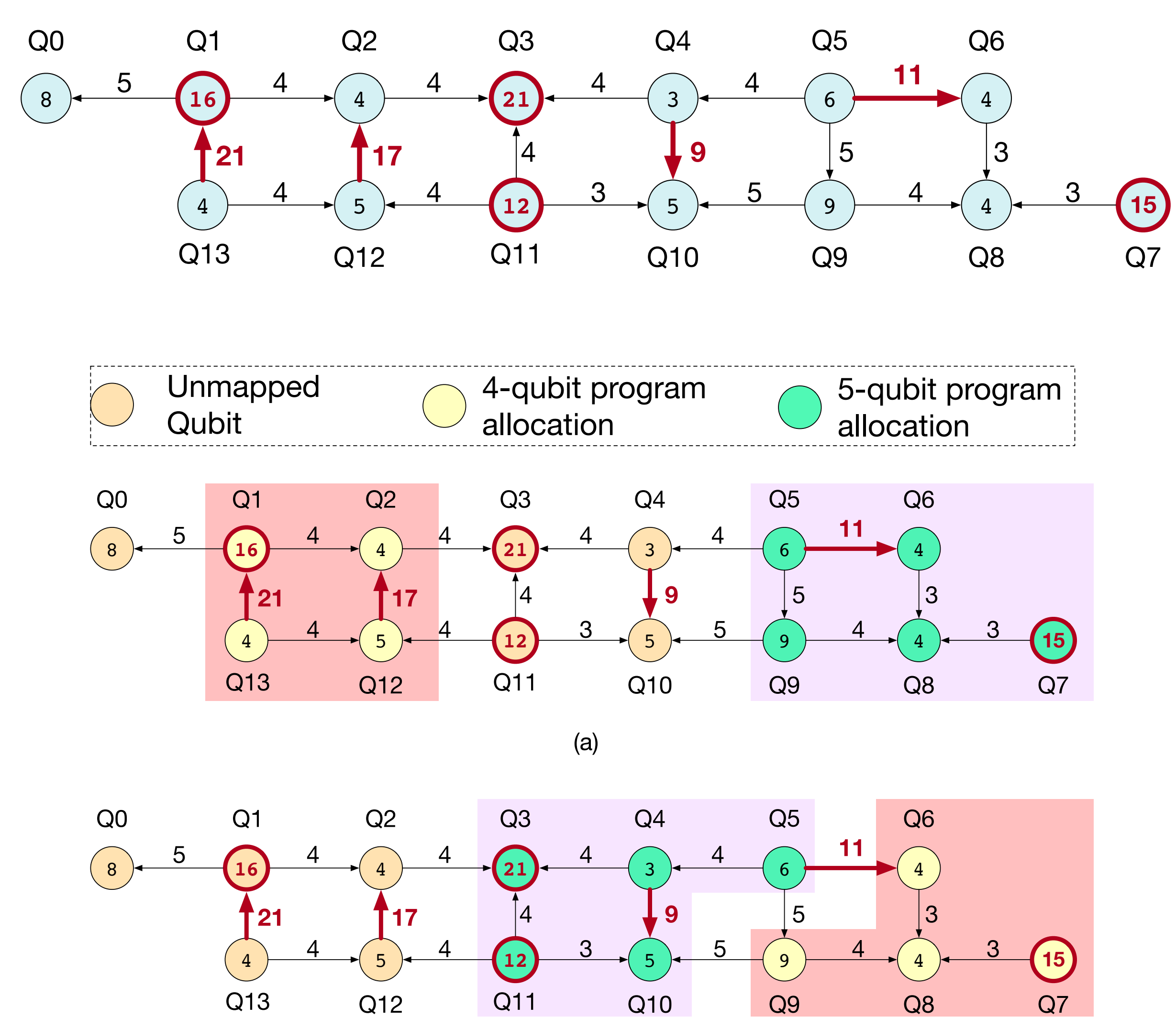
NISQ application design: investigate the best balance of **trade-offs** among a large number of (potentially heterogeneous) factors specific to the **targeted application** and **quantum hardware.**

Multi-Programming (MICRO 2019) :



(a) (b)

IBM Q16



Competing Goals:

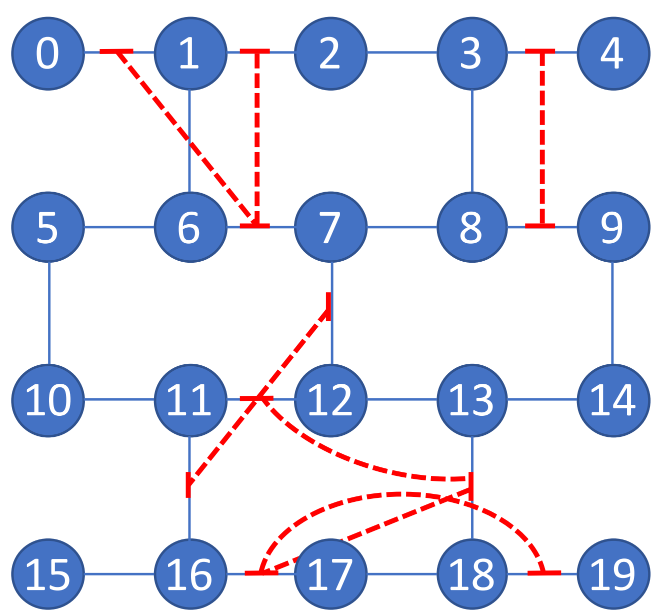
(1) Fully leverage qubits & Shorten the total execution
=> Multi-Programming

(2) High Reliability => Use the best qubits
=> Sequentially Allocate Programs

Solution: A run-time trade-off between these competing goals.

Features of NISQ Application Design

Cross-talk:



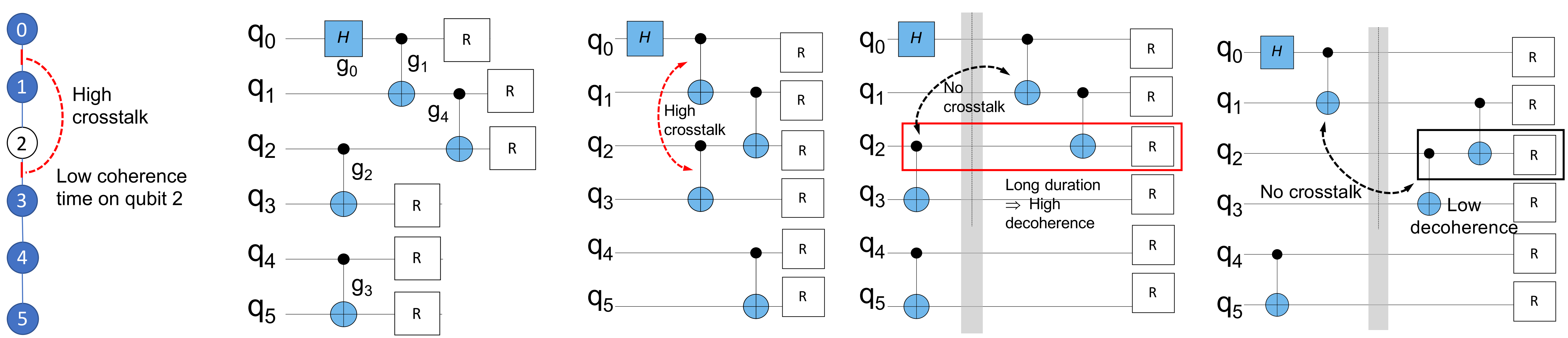
Cross-Talk: **Red Pairs** of gates when executed simultaneously will cause much larger errors.

IBMQ Boeblingen

Competing Goals:
Circuit Depth (decoherence) vs Cross-Talk

Software Solutions:
(1) Circuit Reschedule - Xtalk - (ASPLOS 2020)
(2) Frequency-Aware Compilation (MICRO 2020)

Xtalk



(a) Machine **(b)** Program IR **(c)** Original Default Schedule **(d)** High decoherence schedule **(e)** Desired Schedule

Automating NISQ Application Design

Current implementation of NISQ application design are CASE by CASE.

A unified and automatic framework for productivity?

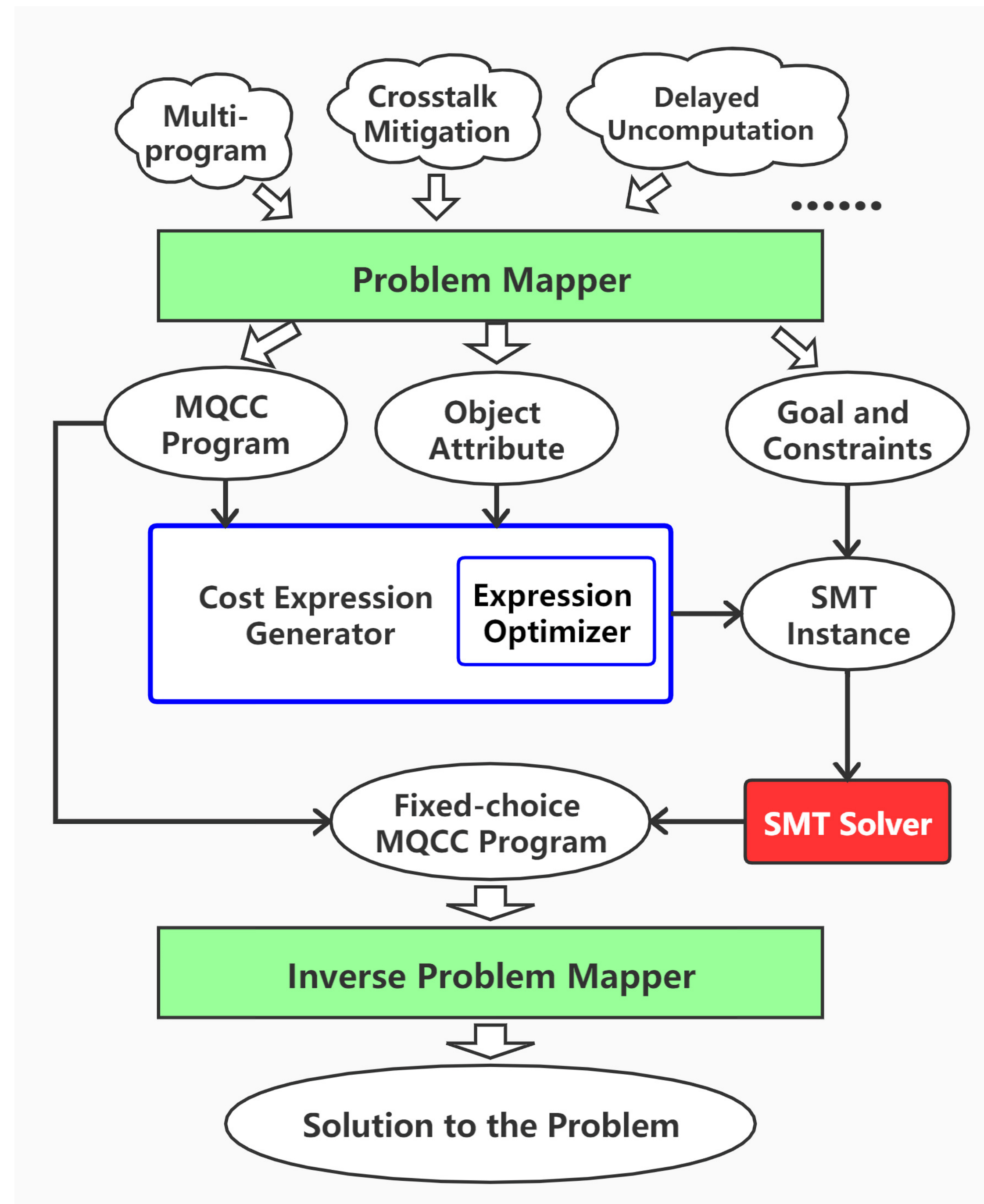
Desiderata:

Succinct Expression
of different design choices

Flexible Expression
of different optimization goals

Automation of Trade-offs
of competing optimization goals

High Reusability & Productivity
of balancing different trade-offs



Meta Quantum Circuits with Constraints (MQCC)

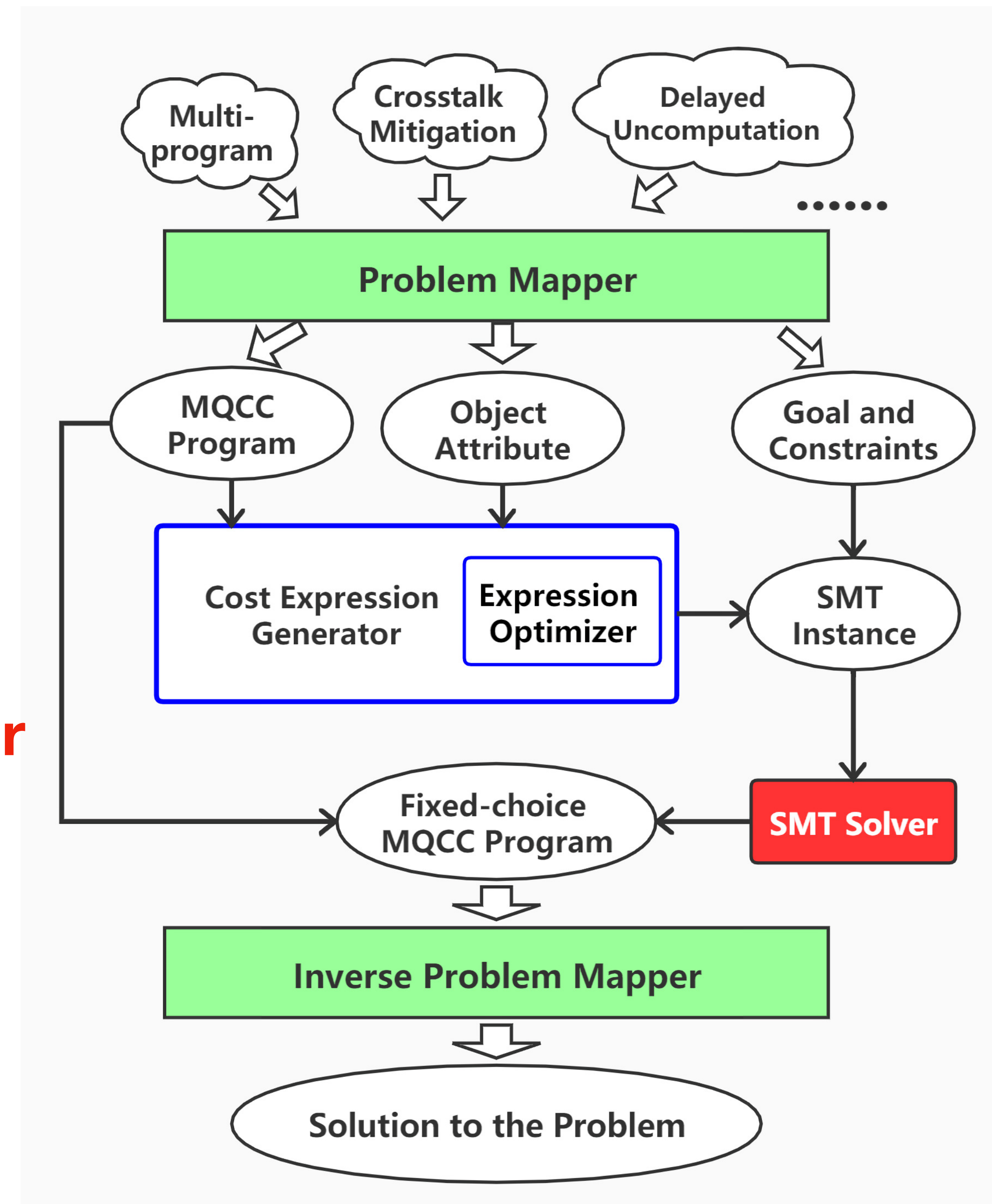
Desiderata:

Succinct Expression
of different design choices
MQCC with choice variables

Flexible Expression
of different optimization goals
Flexible Attributes Expression

Automation of Trade-offs
of competing optimization goals
Satisfiability Modulo Theories (SMT) Solver

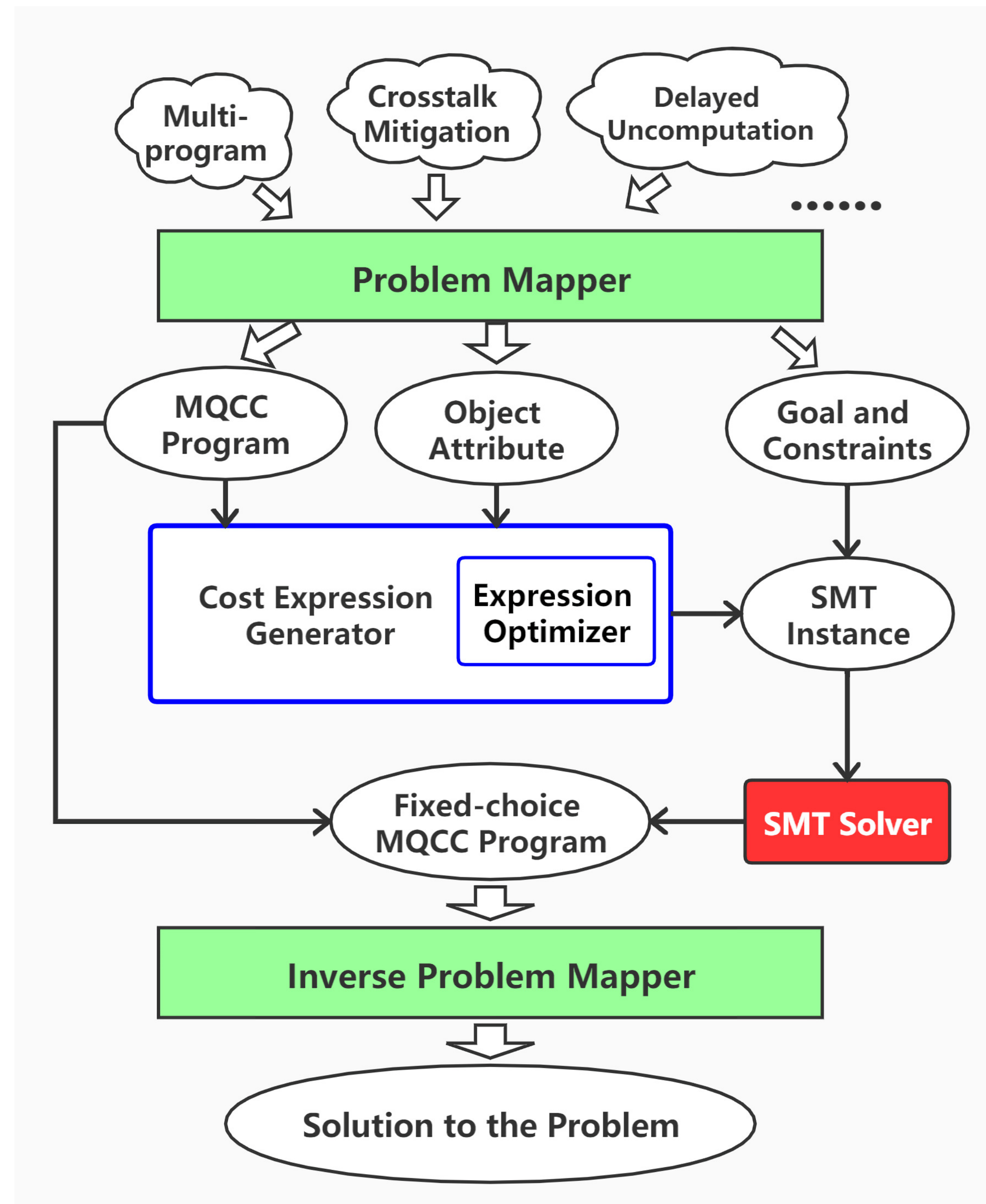
High Reusability & Productivity
of balancing different trade-offs
A Meta-Programming Framework



Meta Quantum Circuits with Constraints (MQCC)

```
\\Register and variable declarations
qreg q[10];
creg r[1];
fcho c1 = {0, 1};
fcho c2 = [0, 1];
\\lcho c = 1 - c1 * c2;

\\Module define
module Bell1(q1,q2){
    h(q1);
    cnot(q1, q2);
}

module Bell2(q1, q2){
    case (r[0]){
        1:  x(q1);
        0:  pass
    };
    h(q1);
    cnot(q1,q2);
}

\\Main part of the program
choice (c1){
    0:  Bell1(q[1], q[2]);
    1:  Bell1(q[7], q[8]);
};

h(q[0]);
measure(q[0],r[0]);
choice (c2){
    0:  Bell2(q[1], q[2]);
    default:  Bell2(q[7], q[8]);
};
```

A Sample Code of MQCC which shares many features with OpenQASM



Meta Quantum Circuits with Constraints (MQCC)

```
\\Register and variable declarations
qreg q[10];
creg r[1];
fcho c1 = {0, 1};
fcho c2 = [0, 1];
\\lcho c = 1 - c1 * c2;

\\Module define
module Bell1(q1,q2){
    h(q1);
    cnot(q1, q2);
}

module Bell2(q1, q2){
    case (r[0]){
        1:  x(q1);
        0:  pass
    };
    h(q1);
    cnot(q1,q2);
}

\\Main part of the program
choice (c1){
    0:  Bell1(q[1], q[2]);
    1:  Bell1(q[7], q[8]);
};

h(q[0]);
measure(q[0],r[0]);
choice (c2){
    0:  Bell2(q[1], q[2]);
    default:  Bell2(q[7], q[8]);
};
```

A Sample Code of MQCC which shares many features with OpenQASM

Define CHOICE variables

Free Choice (fcho) c1, c2 $\in \mathbb{Z}$, in certain ranges

Limited Choice (lcho) c=1-c1*c2 $\in \mathbb{Z}$

Stitch Many Programs w/ choice variables

choice (c.v) $\{i : P_i\}$

$$n \in \mathbb{N} \quad i \in \mathbb{Z} \quad r \in \mathbb{R} \quad var \in Vars$$

$$qreg \in Quantum\ reg. \quad creg \in Classical\ reg.$$

$$\begin{aligned}
reg &::= qreg \mid creg \\
P \in Program &::= \overrightarrow{D}\ S \\
D \in Declaration &::= RegDecl \mid VarDecl \\
RegDecl &::= \textbf{qreg}\ qreg; \mid \textbf{creg}\ creg; \\
VarDecl &::= Free \mid Limit \\
Free &::= \textbf{fcho}\ var = \{\overrightarrow{i}\}; \mid \textbf{fcho}\ var = [i_1, i_2]; \\
Limit &::= \textbf{lcho}\ var = E; \\
E \in VarExp &::= i \mid var \mid E + E \mid E - E \\
&\quad \mid E * E \mid E/E \mid (E) \\
S \in Stmt &::= \epsilon \mid O \mid case \mid choice \mid S; S \\
O \in Operation &::= x(\overrightarrow{r}, \overrightarrow{reg}) \\
case &::= \textbf{case}(creg)\{\overline{i:\ S_i}\} \\
choice &::= \textbf{choice}(var)\{\overline{i:\ S_i}\}
\end{aligned}$$

Meta Quantum Circuits with Constraints (MQCC)

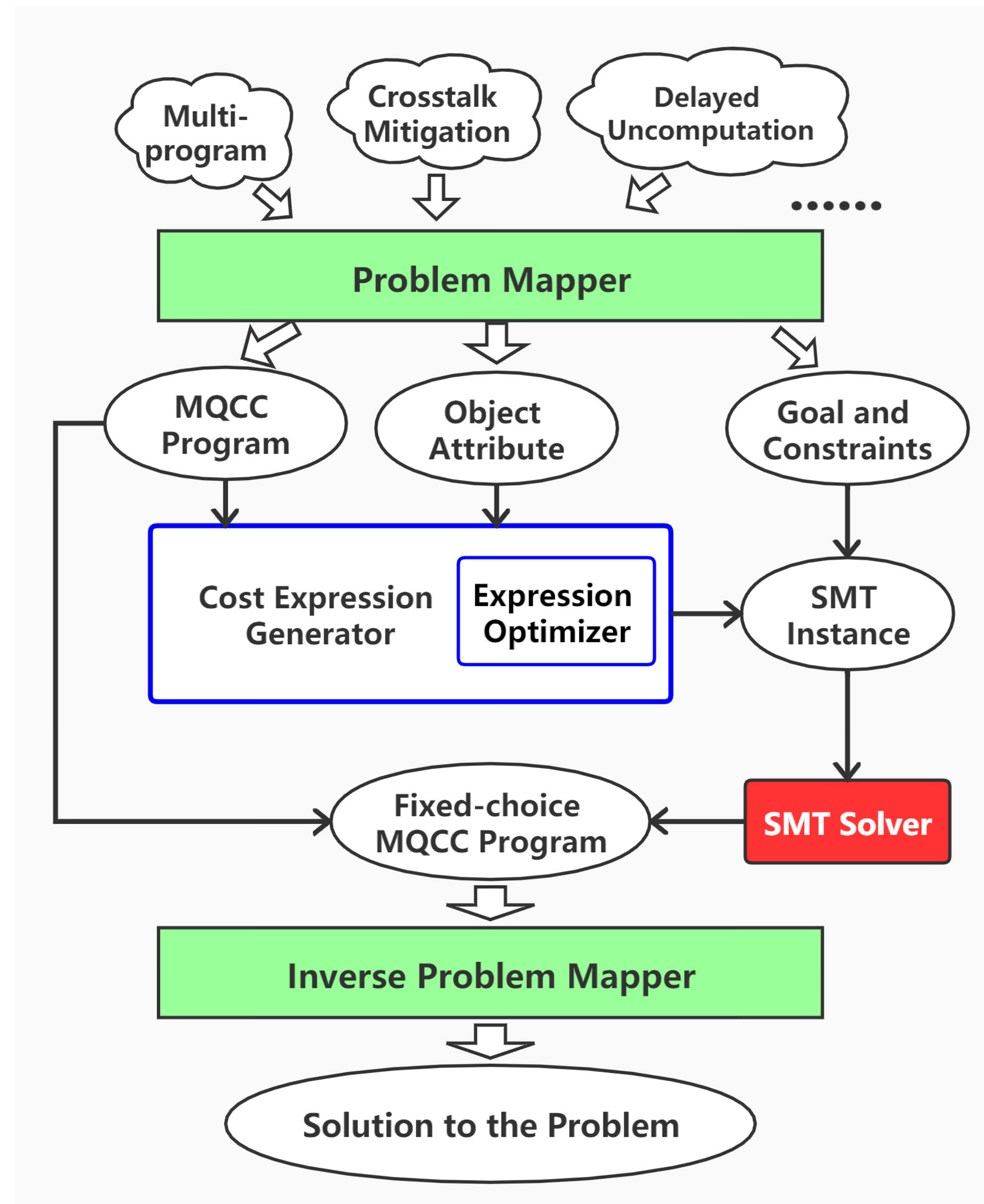
```
\\Register and variable declarations
qreg q[10];
creg r[1];
fcho c1 = {0, 1};
fcho c2 = [0, 1];
\\lcho c = 1 - c1 * c2;

\\Module define
module Bell1(q1,q2){
    h(q1);
    cnot(q1, q2);
}

module Bell2(q1, q2){
    case (r[0]){
        1:  x(q1);
        0:  pass
    };
    h(q1);
    cnot(q1,q2);
}

\\Main part of the program
choice (c1){
    0:  Bell1(q[1], q[2]);
    1:  Bell1(q[7], q[8]);
};

h(q[0]);
measure(q[0],r[0]);
choice (c2){
    0:  Bell2(q[1], q[2]);
    default:  Bell2(q[7], q[8]);
};
```

A Sample Code of MQCC which shares many features with OpenQASM

$$\textbf{Depth}: \quad 7\delta_{c_1}^0\delta_{c_2}^0 + 5\delta_{c_1}^0\delta_{c_2}^1 + 5\delta_{c_1}^1\delta_{c_2}^0 + 7\delta_{c_1}^1\delta_{c_2}^1$$

$$\textbf{Noise}: \quad 0.045\delta_{c_1}^0 + 0.066\delta_{c_1}^1 + 0.027\delta_{c_2}^0 + 0.043\delta_{c_2}^1$$

where $\delta_c^i = 1$ *iff* $c = i$; *otherwise 0*



Expressing the Constraints on Costs/Attributes

Express **desired goals** as objects called **Attributes**. Thus, any MQCC program is a **transformer** on attributes.

Precisely, any **attribute A** is defined by a tuple (*T, empty, op, case, value*) s.t.:

- T is a data type of the states. A state of type T consists of information needed in the computation of the cost.
- $\texttt{empty} : T$ is the initial state at the beginning of the program.
- $\texttt{op} : T \times \text{string} \times \overrightarrow{\mathbb{R}} \times \overrightarrow{reg} \rightarrow T$ receives a state, an operation's name and its arguments, and generates a new state that merges the old state and the information of the operation.
- $\texttt{case} : T \times \overrightarrow{T} \rightarrow T$ receives an old state, a list of states corresponding to each case branch which has merged the corresponding sub-programs' information on the old state, and generates a new state merging the old state and the sub-programs' states.
- $\texttt{value} : T \rightarrow \mathbb{R}$ computes the cost of this attribute from the information stored in a state.

choice vars

program S' attribute semantics $\left[\!\left[S\right]\!\right]$: $(Vars \rightarrow \mathbb{Z}) \times T \rightarrow T$

transformer on T

$$\frac{S = opID(exps,\ regs)}{[\![S]\!]\,(\sigma, s) = \texttt{op}(s, opID,\ exps,\ regs)}$$

$$\frac{}{[\![S_1; S_2]\!]\,(\sigma, s) = [\![S_2]\!]\,(\sigma, [\![S_1]\!]\,(\sigma, s))}$$

$$\frac{S = \textbf{case}(creg)\{\overline{i:\ S_i}\}}{[\![S]\!]\,(\sigma, s) = \texttt{case}(s, [[\![S_i]\!]\,(\sigma, s)]_i)}$$

$$\frac{S = \textbf{choice}(var)\{\overline{i:\ S_i}\} \qquad k = \sigma[var]}{[\![S]\!]\,(\sigma, s) = [\![S_k]\!]\,(\sigma, s)}$$

how transformers evolve over programs

Express the constraints on the final T as SMT instances
(some optimization applied but details omitted)

Expressing the Constraints on Costs/Attributes

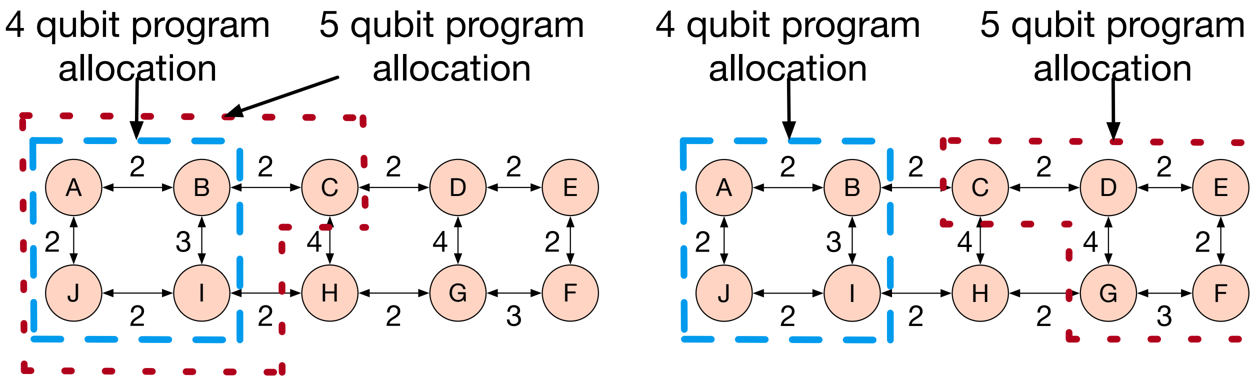
Simple Examples of Attributes

```
Attribute Noise:
  T:                   noise : ℝ
  empty():=            init s : T, s.noise = 0
                       return s
  value(s : T):=       return s.noise
  op (s : T, OpID : str, exps : →ℝ, regs : →Reg):=
                 s.noise += calNoise(OpId, exps, regs)
                       return s
  case (s : T, group : Vector of T):=
                       s.noise = max {n.noise| n ∈group}
                       return s

Attribute Depth:
  T:              dep : Map of Reg → ℕ
  empty():=       init s : T, s.dep = ∅
                  return s
  value(s : T):=  return (max s.dep.values)
  op (s : T, OpID : str, exps : →ℝ, regs : →Reg):=
                  share = s.dep.keys ∩ regs
                  next = max {s.dep[i]| i ∈share} + 1
                  for i ∈regs: s.dep.update(i, next)
                  return s
  case (s : T, group : Vector of T):=
                  all = ⋃_{n∈group} n.dep.keys
     s.dep = {(k, max {n.dep[k] | n∈group}) | k∈all}
                  return s
```

Case Study

All experiments performed on IBMQ machines

Multi-Programming (MICRO 2019) :



(a) (b)

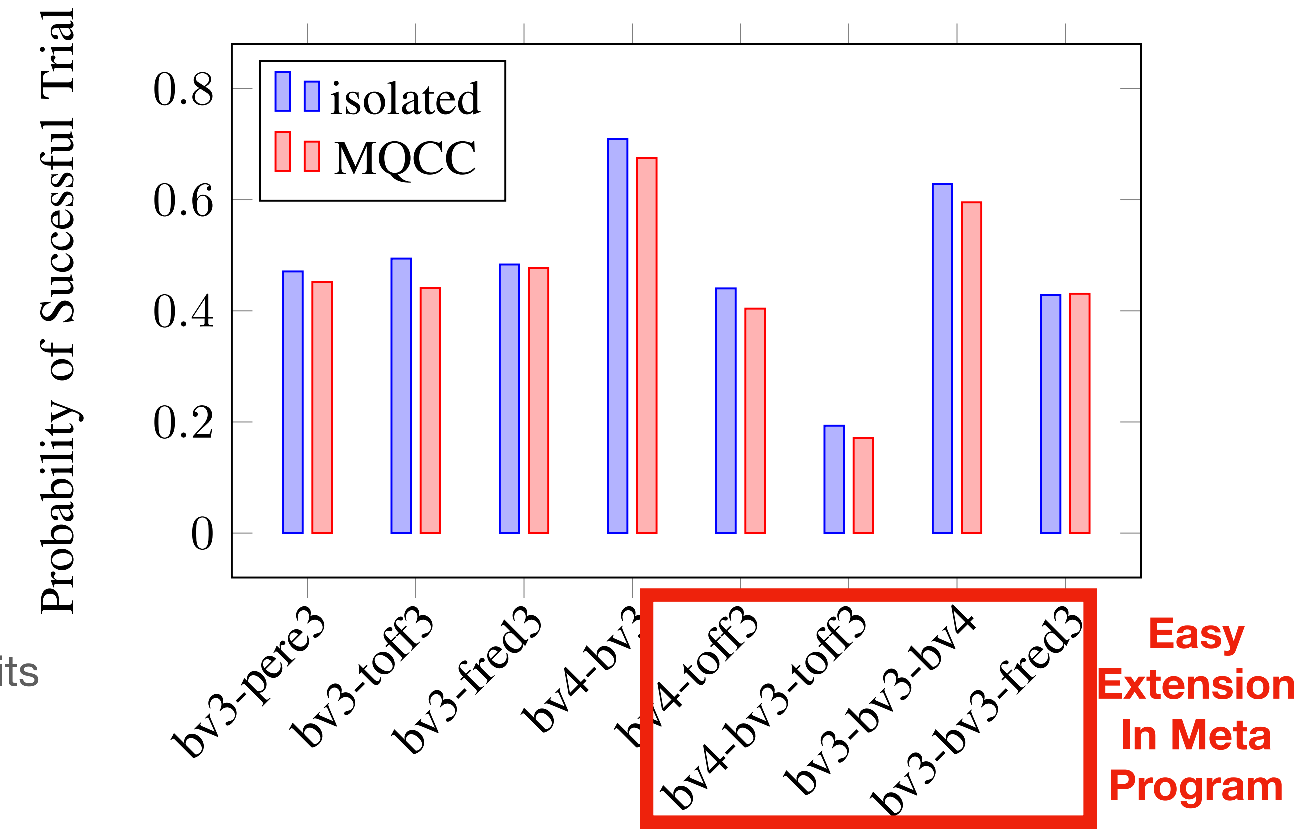
Competing Goals:
Depth vs High-quality Qubits

isolated

Sequential: always high-quality qubits

MQCC

Multi-Programming with MQCC



Multi-Tasks over Simple Quantum Algorithms

Recover some essential ideas of MICRO 2019 while ignoring others.

```
qreg q[10];
creg r[1];

module Bell1(q1,q2){
    h(q1);
    cnot(q1, q2);
}

module Bell2(q1, q2){
    case (r[0]){
            1:  barrier(q1);
                x(q1);
            0:  pass
        }
    h(q1);
    cnot(q1,q2);
}
```

```
fcho c1 = {0, 1};
fcho c2 = [0, 1];

choice (c1){
    0:  Bell1(q[1], q[2]);

    1:  Bell1(q[7], q[8]);
}

choice (c2){
    0:  Bell2(q[1], q[2]);

    1:  Bell2(q[7], q[8]);
}
```

Group A-B contains two applications A and B. Similarly A-B-C.

A success trial if all applications in the group are successful.

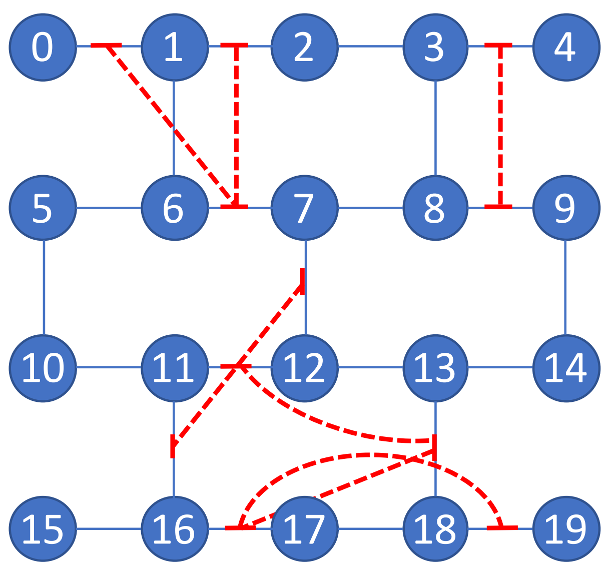
Execute 8192 trials on IBMQ Rochester for each group.

Comparable results to MICRO 2019 for this part.

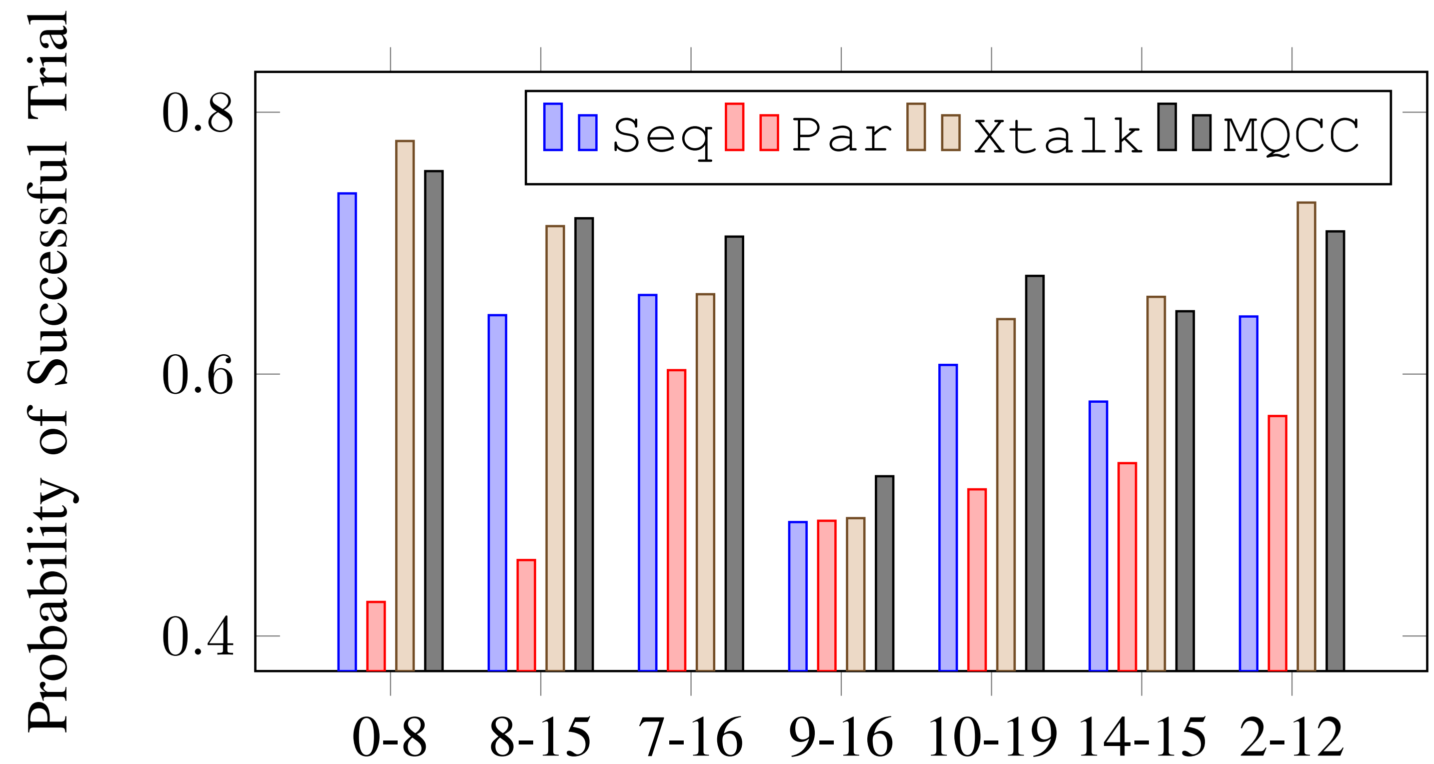
Application	Description	Qubits	# of gates	# of CNOTs
bv3	Bernstein-Vazirani [3]	3	8	2
bv4	Bernstein-Vazirani [3]	4	11	3
Toff3	Toffoli gate	3	15	6
Fred3	Fredkin gate	3	17	8
Pere3	Peres gate	3	16	7

Case Study

Cross-talk: (Xtalk - ASPLOS 2020)



Competing Goals:
Circuit Depth (decoherence) vs Cross-Talk



Benchmark over SWAP circuits connecting a-b on IBM Boeblingen

```
module cnotb(c,q1,q2){
    choice (c){
        0:  cnot(q1,q2);
        1:  barrier(q1,q2);
            cnot(q1,q2);
}   }
```

use "barrier" to control the order of the gates

Benchmark Test:

CNOT 15 8 = SWAP 15 16; SWAP 16 11; SWAP 8 7l SWAP 7 12; CNOT 11 12

Execute 8192 trials on IBMQ Boeblingen for SWAP circuits connecting a-b

Seq running all instructions serially

Par maximize the parallel execution, default in Qiskit

```
Attribute Crosstalk
  T:   dep : Map of Reg → ℕ
       rep : Map of ℕ → Set of (str × Reg⃗)
  empty ():=     init s:T, s.dep = ∅, s.rep = ∅
                 return s
  value (s : T):=   return calCross(rep)
  op (s : T, opID : str, exps : ℝ⃗, regs : Reg⃗):=
       if opId == "barrier" :
         cur = max s.dep.values
         for i∈regs: s.dep.update(i, cur)
       else:
         share = s.dep.keys ∩ regs
         next = max {s.dep[i] | i ∈ share} + 1
         s.rep[next].insert( (OpID, regs) )
         for i∈regs: s.dep.update(i, next)
       return s
  case (s : T, group : Vector of T):=
       all = ⋃ n.dep.keys, n∈group
       s.dep = {(k, max {n.dep[k]|n∈group})| k∈all}
  s.rep = {(k, ⋃_{n∈group} n.rep[k]) |∃u ∈ group, k∈ u}
       return s
```

Case Study: Multi-Programming + Cross-Talk

Optimizing Goal:
Noise + Decoherence + Crosstalk

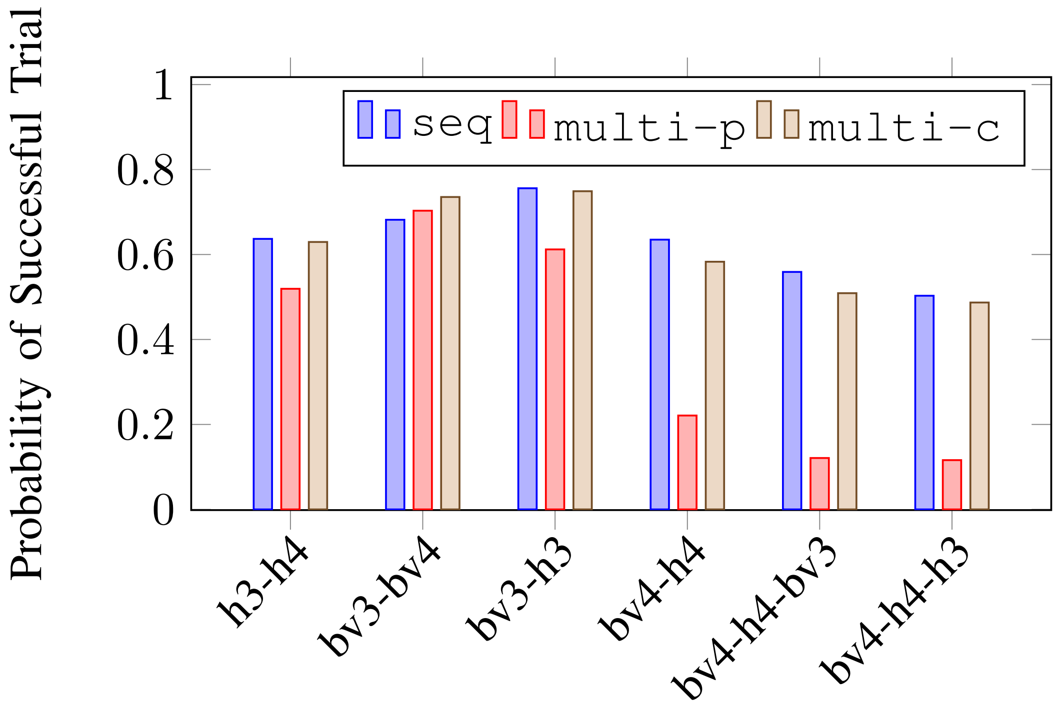
EASY implementation in MQCC

All experiments performed on IBMQ machines

`seq` Sequential: always high-quality qubits.
but larger depth (decoherence)

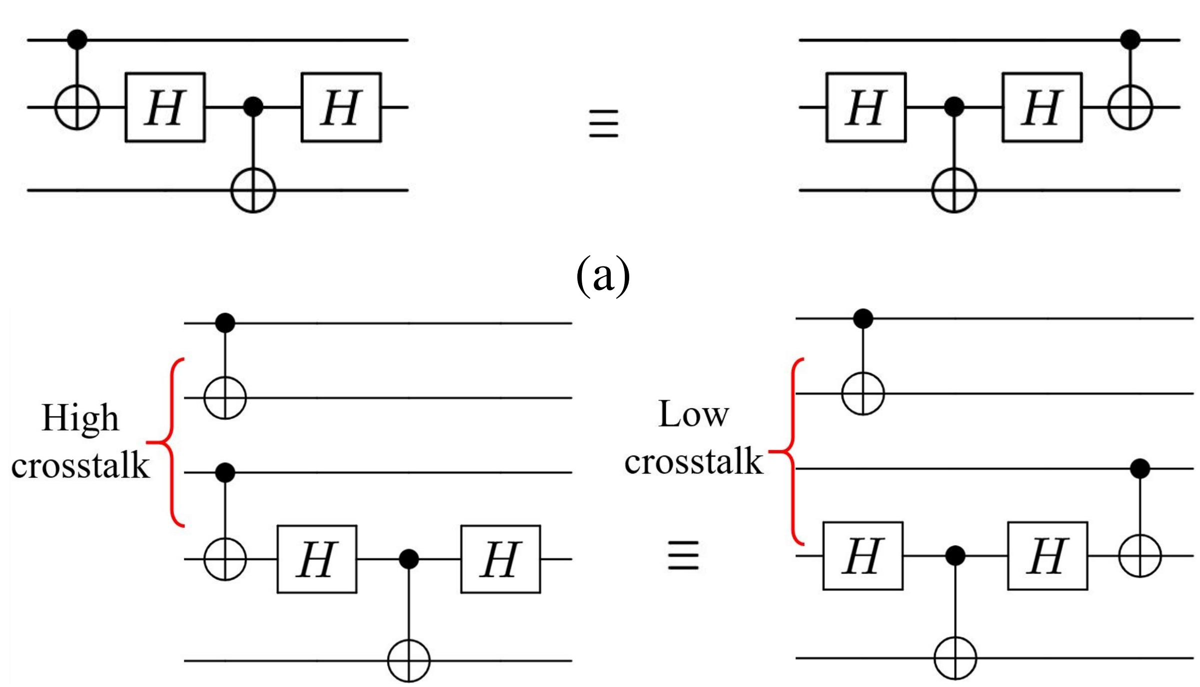
`multi-p` Multi-programs without considering crosstalk
short depth, but large crosstalk errors

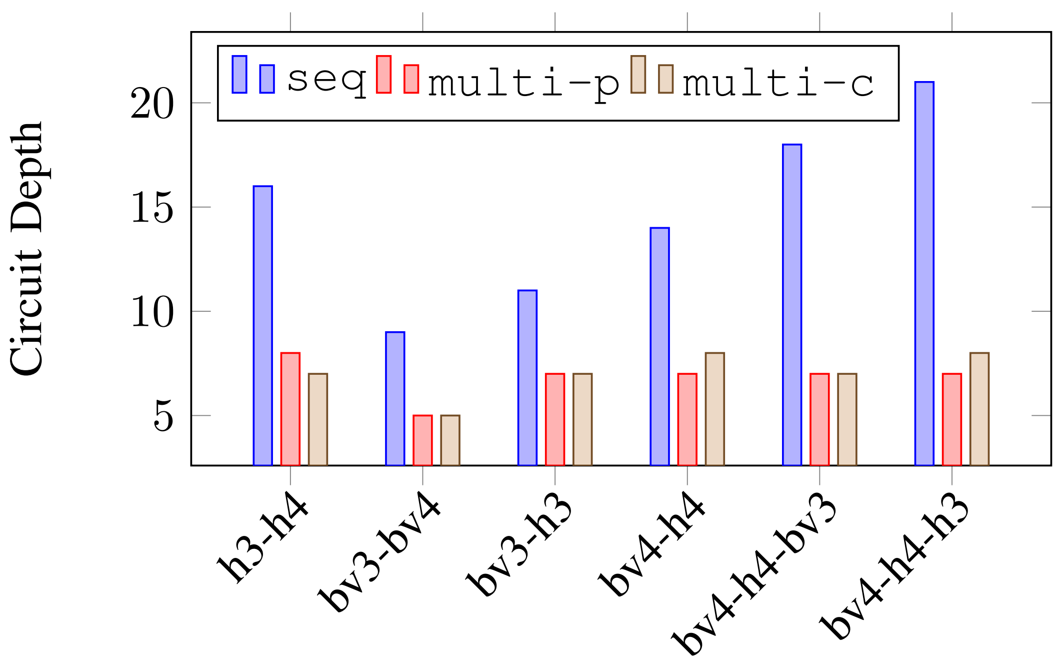
`multi-c` Multi-programs with crosstalk
short depth and large successful probability



(a) Probability of Successful Trial. Here higher PST is better.

More Optimization w/ MQCC





(b) Circuit Depth. Here lower circuit depth is better.

Case Study: Cost-Effective Uncomputation

Recovering one idea from SQUARE (ISCA 2020)

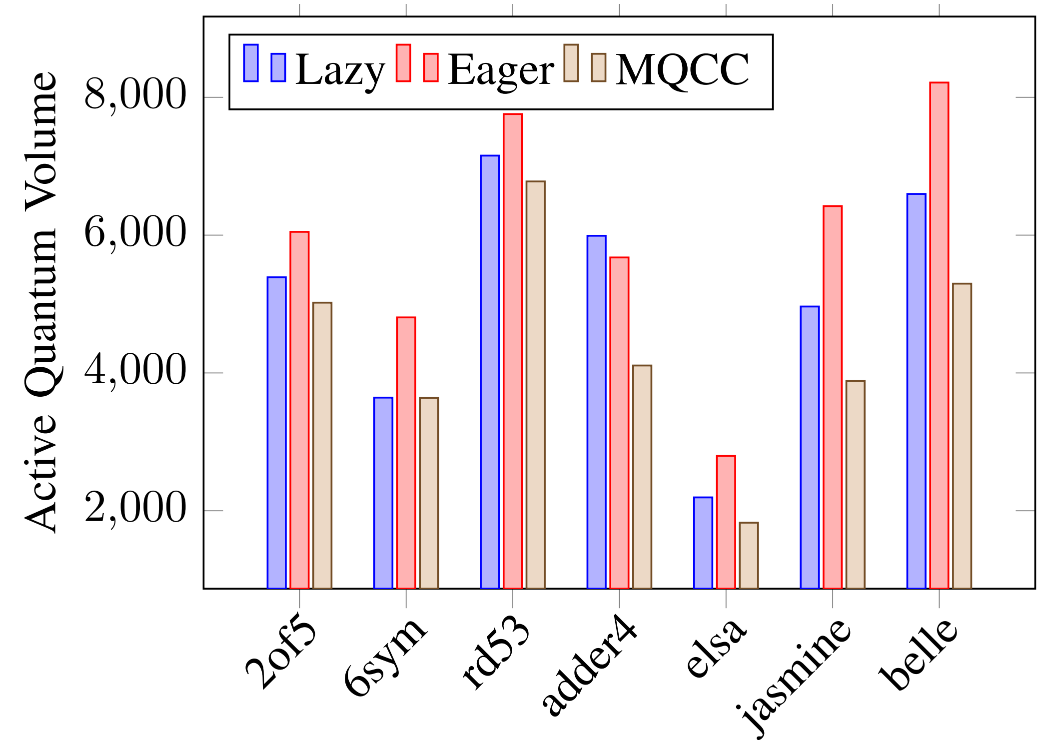
Strategic Quantum Ancilla Reuse for Modular Quantum Programs

Deciding the point to uncompute for ancilla reuse

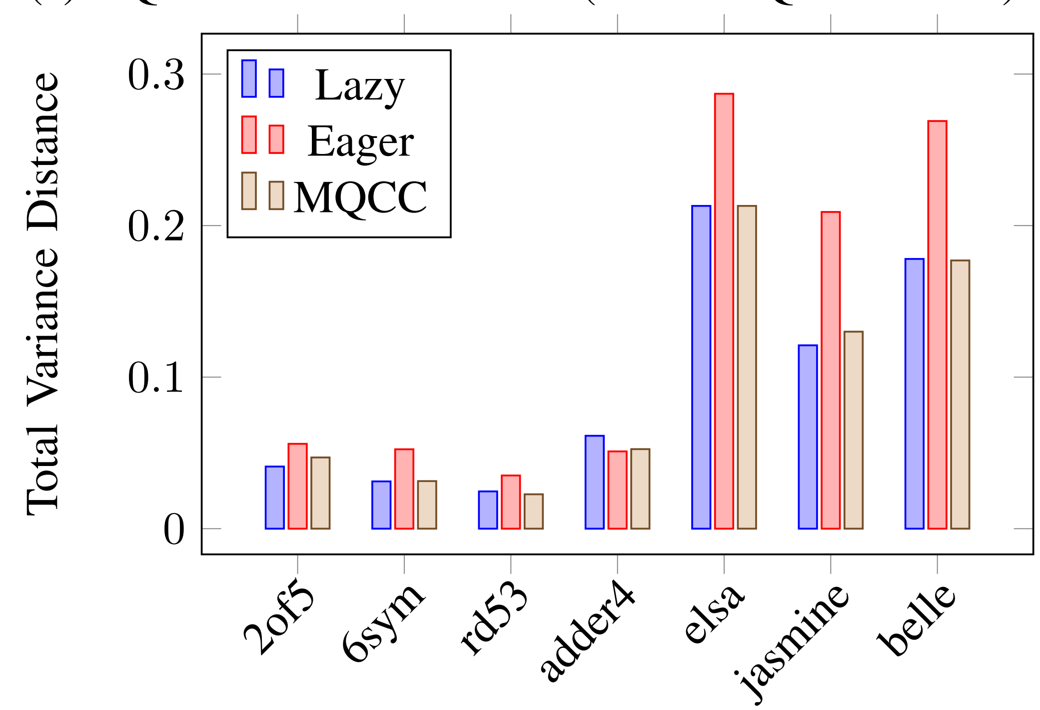
```
module foo(c,...){
    ...
    choice (c) {
        0:   pass \\No uncomputation
        1:   uncomputation code; \\Do uncomputation
             release(ancillas);
    }
}
```

```
fcho c1,c2 = {0,1};
lcho ct = 1 - c1*c2;

foo1(c1,...);
foo2(c2,...);
choice (ct) {
    0: pass \\No uncomputation
    1: uncompute code \\Do uncomputation
}
```

Name	Discription	Gate Number	Qubits
2of5	Output is 1 if number of 1s in its input equals two.	1528	8
6sym	Function with 6 inputs and 1 output.	1620	11
rd53	Input weight function with 5 inputs and 3 outputs.	1849	10
adder4	4-bit in-place controlled-addition.	1748	12
elsa	Heavy workload and shallowly nested synthetic function.	256	14
jasmine	Shallowly nested synthetic function.	604	11
belle	Light workload and deeply nested synthetic function.	768	9



(a) AQV of the benchmarks. (Lower AQV is better.)



(b) Realistic noise simulation using IBM Qiskit Aer simulator. (Lower total variation distance is better.)

Thank You!

MQCC:
- *github/sqrta/MQCC*