QCTSA

Introduction

The goal of quantum circuit transformation is to map a logical circuit to a physical device by inserting additional gates as few as possible in an acceptable amount of time. We present an effective approach called TSA to construct the mapping. It consists of two key steps: one makes use of a combined subgraph isomorphism and completion to initialize some candidate mappings, the other dynamically modifies the mappings by using tabu search-based adjustment. Our experiments show that, compared with state-of-the-art methods GA, SABRE and FiDLS proposed in the literature, TSA can generate mappings with a smaller number of additional gates and it has a better scalability for large-scale circuits.

All the experiments are conductedon a Ubuntu machine with 2.2GHz CPU and 64G memory, Python version 3.6.9. For the details, please refer to our paper"Quantum Circuit Transformation Based on Tabu Search".

If you have any further questions, please contact with us 52215902019@stu.ecnu.edu.cn.

Main Process

- processing.py: Batch preprocessing.
- complete.py: Batch generate initial mappings.
- run.py: Batch adjust the circuits.
- tsa.py: Handle a single circuit.
- main.py: Statistical data.

Test

Ready to work

1.**Compile the IMSM:**

Under the directory `./CISC/SubgraphMatching/`, execute the following commands to compile the source code.

```
mkdir build
cd build
cmake ..
make
```

2.**Install dependencies**

We provide python dependency packages in the package **site-packages.zip** and use the following command to install the dependencies.

```
mkdir -p /home/tacas22/.local/lib/python3.8/site-packages
unzip ./site-packages.zip -d /home/tacas22/.local/lib/python3.8/site-packages
```

If unsuccessful, we suggest to connect to the network and use the command `pip install pyquil` to install the dependency packages.
Note that a large number of generated circuit files will be generated during the search process. It is recommended to add a circuit file generated by a timed task.

```
vim /etc/crontab
eg:
*/10 * * * *    root    find /home/tacas22/Desktop/QCTSA/results/circuits/   -
name '*.*'  -mmin +60 -ls -exec rm -v {} \;
```

Execute a circuit

execute the following command to transform a quantum circuit.

```
python tsa.py [1 connect/degree] [2 num/depth/cca] [3 l_a] [4 delta] [5 input
file path] [6 the output file prefix]
eg:
python3 tsa.py  connect num 3 0.51 ./data/0example.qasm  qct
```

The circuit generated by QCTSA outputs in directory `result/circuits/paramter 6` and the the number of additional gates inserted are in directory `./result/paramter 6`.
Every two lines in the file represent the result of a circuit transformation. The first line is the circuit name,
and the next line, '(prefix, min_index, ini_gates, gates, depth, min_swaps, l_a, \delta, min_time)', prefix: the label of method. min_index: the initial mapping index corresponding to the fewest additional gate. ini_gates: the number of gates of the original circuit, gates: the number of gates of the circuit generated by QCTSA, depth: the depth of the circuit generated by QCTSA. min_swap: the fewest number of additional gates inserted. l_a: the number of look ahead layers. \delta: attenuation factor.   min_time: the runtim in seconds.

Batch process the circuits

First execute the following command in the Processing folder: batch preprocessing and
 start_index (resp. end_index) represents the start (resp. end) index of the directory `./data/`.

```
python3 processing.py  [start_index]  [end_index]
```

The result stored in folders `./processed_data/`.

Second execute the following command: connect or num to indicate the strategy for generating the initial mapping.

```
python3 complete.py  [connect/degree]
```

The results are stored in the folder `./connect_ini_mapping_q20/` or `./degree_ini_mapping_q20/`, which includes all initial mappings.

Final execute the following command to transform the quantum circuits in the directory `./processed_data/`. All of results are in the directory `./results/`

```
python run.py [[1 connect/degree] [2 small/medium/large/all] [3 num/depth/cca] [4
the output file prefix] [5 initial mapping path] [6 look-ahead start index] [7
look-ahead end index]  [8 attenuation factor start index] [9 attenuation factor
end index] [10 the attenuation factor gap]
eg:
python3 run.py connect all num qct connect_ini_mapping_q20 3 4 0.1 1.0 0.02
```

How to replicated the results?

We optimized the code, and the existing results are slightly better than the data of the submitted paper.The new version does not sort the candidate SWAP edges, so there will be randomness when selecting candidates with the same value. This randomness adds more possibilities to the circuit compilation, and it is possible to find a fewer solution for inserting the SWAP gate. The total number of inserted SWAP gates for 159 cases of submitted papers is 291161 in the file **./results/data/tsa/tsa**. The new results are in the directory **./results/new/**. The **tsa** results are consistent with the paper.

We compare with three state-of-the-art algorithms GA ,SABRE, FiDSL.

1. Process the data:

   ```
   python3 processing.py  0  158
   ```

2. Then, complete the initial mapping:

   ```
   python3 complete.py connect
   python3 complete.py degree
   ```

3. Calculate the optima look-ahead layers l_a and attenuation factor \delta.

   ```
   python3 run.py connect all num qct 1 10 0.01 1 0.025
   ```

4. Generate data for three evaluation functions.

   Use the number of additional gates in the generated circuit:

   ```
   python3 run.py connect all num qct 1 10 0.01  1 0.025
   ```

   Use the depth of the generated circuit:

   ```
   python3 run.py connect all depth qct 1 0 0.01  1  0.025
   ```

   Use CCA:

   ```
   python3 run.py connect all cca qct 1 10 0.01  1 0.025
   ```

5. Generate data for initial mapping algorithm comparison and adjustment algorithm comparison experiments.

When doing comparison experiments, it is necessary to combine the initial mapping and adjustment algorithms of **GA**, **SABRE**, **FiDSL**, and **TSA** in pairs. We provide a comparison program between TSA and other algorithms. Use the following methods respectively. We recommend running data separately for small, medium and large circuits, so that more data can be obtained faster.

1. We provide the initial mapping result of **GA (./optm_ini)**, **SABRE (./sabre_ini)**, **FiDSL (./fidsl_ini)**. When you enter the command, select the corresponding initial mapping path, and then you can switch between different initial mapping methods. The combination of TSA and other algorithms can be obtained by applying the initial mapping result in `connect_ini_mapping_q20` to the corresponding adjustment algorithm. In order to obtain the range of data, this process may take a long time. You can modify the gap precision of \delta to shorten the running time.

   The total circuits:

   ```
   TSA_num: python3 run.py connect all num qct 1 10 0.01 1 0.025
   TSA_cca: python3 run.py cca all num qct 1 10 0.01 1 0.025
   GA+TSA_num: python3 run.py ga all num qct 1 10 0.01 1 0.025
   SABRE+TSA_num: python3 run.py sabre all num qct 1 10 0.01 1 0.025
   FiDSL+TSA_num: python3 run.py fidsl all num qct 1 10 0.01 1 0.025
   ```

   The small circuits:

   ```
   TSA_num: python3 run.py connect small num qct 1 10 0.01 1 0.0025
   TSA_cca: python3 run.py cca small num qct 1 10 0.01 1 0.0025
   GA+TSA_num: python3 run.py ga small num qct 1 10 0.01 1 0.0025
   SABRE+TSA_num: python3 run.py sabre small num qct 1 10 0.01 1 0.0025
   FiDSL+TSA_num: python3 run.py fidsl small num qct 1 10 0.01 1 0.0025
   ```

   The medium circuits:

   ```
   TSA_num: python3 run.py connect medium num qct 1 10 0.01 1 0.0025
   TSA_cca: python3 run.py cca medium num qct 1 10 0.01 1 0.0025
   GA+TSA_num: python3 run.py ga medium num qct 1 10 0.01 1 0.0025
   SABRE+TSA_num: python3 run.py sabre medium num qct 1 10 0.01 1 0.0025
   FiDSL+TSA_num: python3 run.py fidsl medium num qct 1 10 0.01 1 0.0025
   ```

   The large circuits:

   ```
   TSA_num: python3 run.py connect large num qct 1 10 0.01 1 0.025
   TSA_cca: python3 run.py cca medium large qct 1 10 0.01 1 0.025
   GA+TSA_num: python3 run.py ga medium large qct 1 10 0.01 1 0.025
   SABRE+TSA_num: python3 run.py sabre large num qct 1 10 0.01 1 0.025
   FiDSL+TSA_num: python3 run.py fidsl large num qct 1 10 0.01 1 0.025
   ```

2. TSA is used as the initial mapping, and the adjustment results on **GA**, **SABRE**, and **FiDSL** need to be combined with their code to run, and the running time of each case is controlled within one hour. We provide the running results in the folders **results/data/optm**, **results/data/sabre**, **results/data/fidsl**, **results/data/tsa**, **results/data/cca**.

6. After getting the results of each operation, we need to process the statistics. The file **main.py** is process the old statistics in the directory **./results/data/**, and the file **main_new.py** process the new statistics in the directory **./results/new/**.

    1. By changing the parameters l_a and \delta, we can get the minimum configuration for inserting SWAP gates (corresponding to Fig 5 (a)).

    ```
    python3 main.py best [data folder]
    eg: python3 main.py best ./results/qct/
    ```

    2. Search each case to insert the least SWAP gate and output to the specified file. The data of commands 'pairwise' is from files `./results/data/fidsl/fidsl`, `./results/data/sabre/sabre`, `./results/data/tsa/tsa`, `./results/data/cca/cca`, and `./results/data/tsa/tsa_depth`, respectively. The output file does not contain the 0 6 7 columns of the original data, and it is output according to the original data on the command line.

    ```
    python3 main.py minigate [data folder] [output file path]
    eg: python3 main.py minigate ./results/qct/  ./results/data/tsa/tsa
        python3 main.py minigate ./results/qct/  ./results/data/cca/cca
        python3 main.py minigate ./results/qct/
    ./results/data/tsa/tsa_depth
        python3 main.py minigate ./results/qct/  ./results/data/cca/tsacca
    ```

3. Statistics of initial mapping results (corresponding to Table 2).

```
python3 main.py ini
```

4. Statistics of adjustment results (corresponding to Table 3). .

```
python3 main.py adj
```

5. Statistics of evaluation function results (corresponding to Fig 5 (b)).

```
python3 main.py [evalnum]  [TSA_cca_path]  [TSA_depth_path] [TSA_num_path]
eg:
python3 main.py evalnum ./results/data/cca/cca ./results/data/tsa/tsa_depth
./results/data/tsa/tsa
```

```
python3 main.py [evaldepth]  [TSA_cca_path]  [TSA_depth_path] [TSA_num_path]
eg:
python3 main.py evaldepth ./results/data/cca/cca
./results/data/tsa/tsa_depth ./results/data/tsa/tsa
```

6. Comparison of SABRE, FiDLS, TSA_num, TSA_cca (corresponding to Table 4).

```
python3 main.py [pairwise] [small/medium/large/all] [SABRE_path]
[FiDSL_path] [TSA_num_path] [TSA_cca_path]
eg:
python3 main.py pairwise all ./results/data/sabre/sabre
./results/data/fidsl/fidsl ./results/data/tsa/tsa ./results/data/cca/tsa_cca
python3 main.py pairwise small ./results/data/sabre/sabre
./results/data/fidsl/fidsl ./results/data/tsa/tsa ./results/data/cca/tsa_cca
python3 main.py pairwise medium ./results/data/sabre/sabre
./results/data/fidsl/fidsl ./results/data/tsa/tsa ./results/data/cca/tsa_cca
python3 main.py pairwise large ./results/data/sabre/sabre
./results/data/fidsl/fidsl ./results/data/tsa/tsa ./results/data/cca/tsa_cca
```