Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 20, Number 3 (2024), pp. 393-400

http://www.ripublication.com

Quantum Algorithm for Partition Problem by Shor's Fourier Transform with RAM on QCEngine

Toru Fujimura

Art and Physical Education area security office, University of Tsukuba, Ibaraki-branch, Rising Sun Security Service Co., Ltd., 1-1-1, Tennodai, Tsukuba, Ibaraki 305-8577, Japan

Abstract

A quantum algorithm for the partition problem by the Shor's Fourier transform with the RAM on the QCEngine, and its example are reported. In the partition problem, when *n* natural numbers are parted by two groups, it is decided whether a sum of numbers of one group is equal to it of another group or not. In the quantum algorithm by the Shor's Fourier transform with the RAM, its search is done by several times.

Keywords: Quantum algorithm, partition problem, Shor's Fourier transform, RAM, QCEngine.

AMS subject classification: Primary 81-08; Secondary 68R05, 68W40.

1. Introduction

Fujimura discussed a quantum algorithm for the partition problem by the central limit theorem. [1] Still more, Fujimura discussed a quantum algorithm for the knapsack problem by the Shor's Fourier transform with the RAM on the QCEngine. [2]

According to my advanced study, when the partition problem is regarded as a special pattern of the knapsack problem, the complexity of the partition problem is able to be several times.

Therefore, the quantum algorithm for the partition problem is examined by the Shor's Fourier transform with the RAM on the QCEngine, its result is reported.

2. Partition Problem

When n natural numbers are parted by two groups, it is decided whether a sum of numbers of one group is equal to it of another group or not. [1]

3. Quantum Algorithm

It is assumed that n natural numbers are m_i [$1 \leq i \leq n$. i is an integer.], and j is number of work qubits that included the sum of numbers.

First of all, query quantum registers $|x_i\rangle$ [$1 \leq i \leq n$. i and n are integers.], work1 quantum registers $|w_{1,j}\rangle$ [$1 \leq j \leq t$. j and t are integers. t is a necessary number for the sum of numbers.], work2 quantum registers $|w_{2,p}\rangle$ [$1 \leq p \leq t+1$. p and t are integers. t is a necessary number for the sum of numbers. +1 is a qubit for the negative integer. [3]], and ancilla quantum qubits $|a_q\rangle$ [q is a necessary number for decrement with modulus.] are prepared.

Step 1: The natural numbers' data are introduced to the RAM [3].

Step 2: Each qubit of $|x_i\rangle$, $|w_{1,j}\rangle$, $|w_{2,p}\rangle$, and $|a_q\rangle$ is set $|0\rangle$.

Step 3: The Hadamard gate H [1-8] acts on each qubit of $|x_i\rangle$. It changes them for entangled states.

Step 4: For $|x_i\rangle$, RAM [i - 1] [RAM has natural numbers' data of $0 \rightarrow (n - 1)$.] is incremented in $|w_{2,p}\rangle$. In a function, $F = \Sigma_{i=1 \rightarrow n} m_i\, x_i$ is computed, where m_i is i-th natural numbers. This operation makes entangled data base.

Step 5: For $|w_{2,p}\rangle$, mod(k) [k is $(1/2)\Sigma_{i=1 \rightarrow n} m_i$.] is done, where mod($k$) is made by subtraction and addition in this program. [3] Therefore, the subtraction and the addition are done by necessary times, where work1 quantum registers are added work2 quantum registers, and the uncompute is done.

Step 6: For $|x_i\rangle$, the quantum Fourier transform (= QFT) [2-6] is done.

Step 7: For $|x_i\rangle$ and $|w_{1,j}\rangle$, the proves are done.

Step 8: For $|x_i\rangle$, the read is done.

Step 9: A number of spikes is estimated by the function (https: //oreilly-qc. github. io? p = 12-4 [3]), where the function estimate_num_spikes (spike, range) [spike: read value, range: 2^n] is used.

Step 10: From candidates of the number of spikes, the repeat period P is obtained.

Step 11: From $P = 2^0x_1 + 2^1x_2 + 2^2x_3 + 2^3x_4 + \ldots + 2^{n-1}x_n$, when there is $k = m_1x_1 + m_2x_2 + \ldots + m_nx_n$, it is answer [number of combination of necessary sum].

4. Example of Numerical Computation

It is assumed that 7 (= n) natural numbers are $m_1 = 13$, $m_2 = 2$, $m_3 = 1$, $m_4 = 6$, $m_5 = 7$, $m_6 = 12$, $m_7 = 15$, and the upper bound of the natural number is 56. Furthermore, it is assumed that the mod(k) = mod(28), and query quantum register qubits $i = n = 7$. In this example, when mod(28) is 0, P is 0, 27, 39, 62, 65, 88, 100, and 127.

An example of program on the QCEngine is the following.

```
10 var a = [13, 2, 1, 6, 7, 12, 15]; // RAM_a, natural numbers' data.
20 var query_qubits = 7;
30 var work1_qubits = 6;
40 var work2_qubits = 7;
50 var ancilla_qubits = 2; // subtractions of 2 times are able.
60 qc.reset(query_qubits + work1_qubits + work2_qubits + ancilla_qubits);
70 var query = qint.new(query_qubits, 'query');
80 var work1 = qint.new(work1_qubits, 'work1');
90 var work2 = qint.new(work2_qubits, 'work2');
100 var ancilla = qint.new(ancilla_qubits, 'ancilla');
110 qc.label('q'); // set query
120 query.write(0);
130 query.hadamard();
140 qc.label(' ');
150 qc.label('w1'); // set work1
160 work1.write(0);
170 qc.label('w2'); // set work2
180 work2.write(0);
190 qc.label('a'); // set ancilla
200 ancilla.write(0);
210 qc.print('RAM before increment: '+a+'¥n');
220 var query27 = 27; // one of query
230 var query39 = 39; // one of query
240 var query62 = 62; // one of query
250 var query65 = 65; // one of query
260 var query88 = 88; // one of query
```

```
270 var query100 = 100; // one of query
280 var k = 28; // one of natural numbers' sum
290 var work1_0 = 0; // one of work1. one of mod(k). k = 28.
300 qc.label('increment');
310 work2.add(a[0],query.bits(0x1));
320 work2.add(a[1],query.bits(0x2));
330 work2.add(a[2],query.bits(0x4));
340 work2.add(a[3],query.bits(0x8));
350 work2.add(a[4],query.bits(0x10));
360 work2.add(a[5],query.bits(0x20));
370 work2.add(a[6],query.bits(0x40));
380 qc.label('mod(' + k + ')');
390 work2.subtract(k);
400 qc.cnot(ancilla.bits(0x1),work2.bits(0x40));
410 work2.add(k, ancilla.bits(0x1));
420 work2.subtract(k);
430 qc.cnot(ancilla.bits(0x2),work2.bits(0x40));
440 work2.add(k, ancilla.bits(0x2));
450 work1.add(work2);
460 qc.label('uncompute');
470 work2.subtract(k,ancilla.bits(0x2));
480 qc.cnot(ancilla.bits(0x2),work2.bits(0x40));
490 work2.add(k);
500 work2.subtract(k,ancilla.bits(0x1));
510 qc.cnot(ancilla.bits(0x1),work2.bits(0x40));
520 work2.add(k);
530 work2.subtract(a[6],query.bits(0x40));
540 work2.subtract(a[5],query.bits(0x20));
550 work2.subtract(a[4],query.bits(0x10));
560 work2.subtract(a[3],query.bits(0x8));
570 work2.subtract(a[2],query.bits(0x4));
```

```
580 work2.subtract(a[1],query.bits(0x2));
590 work2.subtract(a[0],query.bits(0x1));
600 qc.label('QFT');
610 query.QFT();
620 var prob27 = 0;
630 var prob39 = 0;
640 var prob62 = 0;
650 var prob65 = 0;
660 var prob88 = 0;
670 var prob100 = 0;
680 prob27 += query.peekProbability(query27);
690 prob39 += query.peekProbability(query39);
700 prob62 += query.peekProbability(query62);
710 prob65 += query.peekProbability(query65);
720 prob88 += query.peekProbability(query88);
730 prob100 += query.peekProbability(query100);
740 // Print output query-Prob
750 qc.print(' Prob_query27: ' + prob27);
760 qc.print(' Prob_query39: ' + prob39);
770 qc.print(' Prob_query62: ' + prob62);
780 qc.print(' Prob_query65: ' + prob65);
790 qc.print(' Prob_query88: ' + prob88);
800 qc.print(' Prob_query100: ' + prob100);
810 var prob0 = 0;
820 prob0 += work1.peekProbability(work1_0); // work1_0 = 0
830 // Print output work1-Prob
840 qc.print(' Prob_work1_0: ' + prob0);
850 // read
860 qc.label('Rq');
870 var b2 = query.read();
880 // Print output result
```

890 qc.print(' Read query = ' + b2 + '.');

900 // end

When this program is copied on Programming Quantum Computers https: //orelly-qc. github. io/# [free on-line quantum computation simulator QCEngine] [3], you can run it. [Caution!: Please delate the line numbers.]

A result of this program is the following.

The probe value of $|w_{1,j}\rangle = 0 : \approx 0.06250$.

The probe value of $|x_i\rangle = 27 : \approx 0.01131$.

The probe value of $|x_i\rangle = 39 : \approx 0.008644$.

The probe value of $|x_i\rangle = 62 : \approx 0.002458$.

The probe value of $|x_i\rangle = 65 : \approx 0.02011$.

The probe value of $|x_i\rangle = 88 : \approx 0.01039$.

The probe value of $|x_i\rangle = 100 : \approx 0.005204$.

The example of 10 times test: The read value of $|x_i\rangle$ = 46, 110, 0, 85, 71, 49, 24, 27, 34, 51. (= spike)

The candidates of number of spikes are estimated by the function [the function estimate_num_spikes (spike, range) [spike: read value, range: $2^n = 2^7 = 128$]:

46 → 3, 6, 8, 11, 14, 25, 39, 64 ; 110 → 7, 14, 21, 28, 36, 43, 50, 57, 64 ; 0 → nothingness ; 85 → 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 65, 68, 71, 74, 77, 80, 83 ; 71 → 2, 5, 7, 9, 18, 27, 36, 45, 54, 63, 65 ; 49 → 3, 5, 8, 13, 26, 34, 47 ; 24 → 5, 11, 16, 32, 48, 64, 80, 96 ; 27 → 5, 10, 14, 19, 38, 57, 71, 90 ; 34 → 4, 8, 11, 15, 30, 34, 49, 64 ; 51 → 3, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 68, 73.

When P is 27, 39, and 65 [100 is another combination for 27, 88 is another combination for 39, and 62 is another combination for 65.], $2^0x_1 + 2^1x_2 + 2^2x_3 + 2^3x_4 + 2^4x_5 + 2^5x_6 + 2^6x_7$:

$2^0\times1 + 2^1\times1 + 2^3\times1 + 2^4\times1 = 27$, $2^0\times1 + 2^1\times1 + 2^2\times1 + 2^5\times1 = 39$, $2^0\times1 + 2^6\times1 = 65$,

$2^2\times1 + 2^5\times1 + 2^6\times1 = 100$, $2^3\times1 + 2^4\times1 + 2^6\times1 = 88$, $2^1\times1 + 2^2\times1 + 2^3\times1 + 2^4\times1 + 2^5\times1 = 62$.

There is $m_1x_1 + m_2x_2 + m_3x_3 + m_4x_4 + m_5x_5 + m_6x_6 + m_7x_7$:

$13\times1 + 2\times1 + 6\times1 + 7\times1 = 13\times1 + 2\times1 + 1\times1 + 12\times1 = 13\times1 + 15\times1 =$

$1\times1 + 12\times1 + 15\times1 = 6\times1 + 7\times1 + 15\times1 = 2\times1 + 1\times1 + 6\times1 + 7\times1 + 12\times1 = 28$ (= k).

5. Discussion

In the knapsack problem, there are many combinations of luggage to obtain a value. In the partition problem, there are several combinations of natural numbers to obtain a value.

Therefore, the search is difficult.

In section 4, there are n natural numbers. When N is 2^n, in the Grover's method, the complexity is $N^{1/2} = 2^{n/2}$, in the Shor's Fourier transform, it is several times.

In $n = 7$, $N^{1/2} = 2^{n/2} = 2^{7/2} \approx 11$, and (several times) $\approx 10/4 \approx 3$.

In this range, the Shor's Fourier transform is less than the complexity of the Grover's method.

6. Summary

The quantum algorithm for the partition problem by the Shor's Fourier transform with the RAM on the QCEngine, and its example are reported.

The complexity of this method is several times.

I will apply this method for other problems.

References

[1] Fujimura, T., 2011, "Quantum algorithm for partition problem by central limit theorem," Glob. J. Pure Appl. Math., **7**, 415-419.

[2] Fujimura, T., 2023, "Quantum algorithm for knapsack problem by Shor's Fourier transform with RAM on QCEngine," Glob. J. Pure Appl. Math., **19**, 547-554.

[3] Johnston, E. R., Harrigan, N., and Gimeno-Segovia, M., 2019, Programming Quantum Computers, O'Reilly, ISBN 978-1-492-03968-6.

[4] Takeuchi, S., 2005, Ryoshi Konpyuta (Quantum Computer), Kodansha, Tokyo, Japan [in Japanese].

[5] Shor, P. W., 1994, "Algorithm for quantum computation: discrete logarithms and factoring," Proc. 35th Annu. Symp. Foundations of Computer Science, IEEE, pp.124-134.

[6] Miyano, K., and Furusawa, A., 2008, Ryoshi Konpyuta Nyumon (An Introduction to Quantum Computation), Nipponhyoronsha, Tokyo, Japan [in Japanese].

[7] Grover, L. K., 1996, "A fast quantum mechanical algorithm for database search," Proc. 28th Annu. ACM Symp. Theory of Computing, pp.212-219.

[8] Grover, L. K., 1998, “A framework for fast quantum mechanical algorithms,” Proc. 30th Annu. ACM Symp. Theory of Computing, pp.53-62.