Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 20, Number 3 (2024), pp. 415-422

http://www.ripublication.com

Quantum Algorithm for Minimum Multiprocessor Scheduling 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 minimum multiprocessor scheduling problem by the Shor's Fourier transform with the RAM on the QCEngine, and its example are reported. When n tasks are parted by m processors, and a sum of length of each task in the v-th processor [$1 \leq v \leq m$. v is an integer.] is t_v, it is decided whether t_v is a finish time D or less or not. In the quantum algorithm by the Shor's Fourier transform with the RAM, its search is done by [(number of processers)-1]×(several times).

Keywords: Quantum algorithm, minimum multiprocessor scheduling 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 minimum multiprocessor scheduling 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 minimum multiprocessor scheduling problem is regarded as a special pattern of the knapsack problem, the complexity of

the minimum multiprocessor scheduling problem is able to be [(number of processers)-1]×(several times).

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

2. Minimum Multiprocessor Scheduling Problem

When n tasks are parted by m processors, and a sum of length of each task in the v-th processor [$1 \leq v \leq m$. v is an integer.] is t_v, it is decided whether t_v is a finish time D or less or not. [1]

3. Quantum Algorithm

It is assumed that n tasks are parted by m processors, a finish time is D, and j is number of work qubits that included the sum of length of each task.

First of all, query quantum registers $|x_i\rangle$ [$1 \leq i \leq n$. i and n are integers.], work1 quantum registers $|w_{1,j}\rangle$ [$1 \leq j \leq u$. j and u are integers. u is a necessary number for the sum of length of each task.], work2 quantum registers $|w_{2,p}\rangle$ [$1 \leq p \leq u$ +1. p and u are integers. u is a necessary number for the sum of length of each task. +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 lengths of tasks' 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 lengths of tasks' data of $0 \rightarrow (n - 1)$.] is incremented in $|w_{2,p}\rangle$. In a function, $F = \Sigma_{i=1 \rightarrow n} L_i\, x_i$ is computed, where L_i is length of i-th task. This operation makes entangled data base.

Step 5: For $|w_{2,p}\rangle$, mod(k) [k is $(1/m)\Sigma_{i=1 \rightarrow n} L_i \leq$ D.] 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 $k = L_1x_1 + L_2x_2 + \ldots + L_nx_n$, it is one of answers [number of combination of necessary sum].

Step 12: Next, a value of k is changed, and then step 1 to step11 are repeated to obtain answers for m processors.

4. Example of Numerical Computation

It is assumed that lengths of 6 (= n) tasks are $L_1 = 5$, $L_2 = 3$, $L_3 = 8$, $L_4 = 7$, $L_5 = 6$, $L_6 = 1$, respectively, the upper bound of the sum of length of each task is 30, m is 3, and u is 5. Furthermore, it is assumed that the mod(k) = mod(10) [$k = (1/m)\Sigma_{i=1\rightarrow n}L_i = (1/3)30 = 10 \leq D = 11$.], and query quantum register qubits $i = n = 6$. In this example, when mod(10) is 0, P is 0, 10, 13, 50, 53, and 63. Still more, when mod(11) is 0, P is 0, 6, 17, 23, 42, 59, and 60.

An example of program on the QCEngine is the following.

```
10 var a = [5,3,8,7,6,1]; // RAM_a, lengths of tasks' data.
20 var query_qubits = 6;
30 var work1_qubits = 5;
40 var work2_qubits = 6;
50 var ancilla_qubits = 3; // subtractions of 3 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 query10 = 10; // one of query
230 var k = 10; // one of sum of length of each task
240 var work1_0 = 0; // one of work1. one of mod(k). k = 10.
250 qc.label('increment');
260 work2.add(a[0],query.bits(0x1));
270 work2.add(a[1],query.bits(0x2));
280 work2.add(a[2],query.bits(0x4));
290 work2.add(a[3],query.bits(0x8));
300 work2.add(a[4],query.bits(0x10));
310 work2.add(a[5],query.bits(0x20));
320 qc.label('mod(' + k + ')');
330 work2.subtract(k);
340 qc.cnot(ancilla.bits(0x1),work2.bits(0x20));
350 work2.add(k, ancilla.bits(0x1));
360 work2.subtract(k);
370 qc.cnot(ancilla.bits(0x2),work2.bits(0x20));
380 work2.add(k, ancilla.bits(0x2));
390 work2.subtract(k);
400 qc.cnot(ancilla.bits(0x4),work2.bits(0x20));
410 work2.add(k, ancilla.bits(0x4));
420 work1.add(work2);
430 qc.label('uncompute');
440 work2.subtract(k,ancilla.bits(0x4));
450 qc.cnot(ancilla.bits(0x4),work2.bits(0x20));
460 work2.add(k);
470 work2.subtract(k,ancilla.bits(0x2));
480 qc.cnot(ancilla.bits(0x2),work2.bits(0x20));
```

```
490 work2.add(k);
500 work2.subtract(k,ancilla.bits(0x1));
510 qc.cnot(ancilla.bits(0x1),work2.bits(0x20));
520 work2.add(k);
530 work2.subtract(a[5],query.bits(0x20));
540 work2.subtract(a[4],query.bits(0x10));
550 work2.subtract(a[3],query.bits(0x8));
560 work2.subtract(a[2],query.bits(0x4));
570 work2.subtract(a[1],query.bits(0x2));
580 work2.subtract(a[0],query.bits(0x1));
590 qc.label('QFT');
600 query.QFT();
610 var prob10 = 0;
620 prob10 += query.peekProbability(query10);
630 // Print output query-Prob
640 qc.print(' Prob_query10: ' + prob10);
650 var prob0 = 0;
660 prob0 += work1.peekProbability(work1_0); // work1_0 = 0
670 // Print output work1-Prob
680 qc.print(' Prob_work1_0: ' + prob0);
690 // read
700 qc.label('Rq');
710 var b2 = query.read();
720 // Print output result
730 qc.print(' Read query = ' + b2 + '.');
740 // 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 $|w_{1,j}\rangle = 0 : \approx 0.09375$.

The probe value of $|x_i\rangle = 10 : \approx 0.008139$.

The example of 10 times test: The read value of $|x_i\rangle$ = 39, 5, 57, 43, 12, 23, 19, 11, 23, 58. (= spike)

The candidates of number of spikes are estimated by the function [the function estimate_num_spikes (spike, range) [spike: read value, range: $2^n = 2^6 = 64$]:

39 → 3, 5, 10, 13, 18, 23 ; 5 → 13, 26, 38, 51 ; 57 → 9, 18, 27, 37, 46, 55 ; 43 → 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 34, 37, 40 ; 12 → 5, 11, 16, 32, 48 ; 23 → 3, 6, 8, 11, 14, 25, 39 ; 19 → 3, 7, 10, 17, 27, 37 ; 11 → 6, 12, 17, 23, 29, 35 ; 23 → 3, 6, 8, 11, 14, 25, 39 ; 58 → 11, 21, 32.

When P is 10, $2^0x_1 + 2^1x_2 + 2^2x_3 + 2^3x_4 + 2^4x_5 + 2^5x_6$:

$2^0\times0 + 2^1\times1 + 2^2\times0 + 2^3\times1 + 2^4\times0 + 2^5\times0 = 10$.

There is $L_1x_1 + L_2x_2 + L_3x_3 + L_4x_4 + L_5x_5 + L_6x_6$:

$5\times0 + 3\times1 + 8\times0 + 7\times1 + 6\times0 + 1\times0 = 10$ ($= k \leq D = 11$).

Therefore, tasks of one processor are 3 and 7.

Next, when mod(k) = mod(11) is 0, P is 0, 6, 17, 23, 42, 59 , and 60. And then, step1 to step11 are repeated to obtain answers for 2 processors.

A result of this program is the following.

The probe value of $|w_{1,j}\rangle = 0 : \approx 0.1094$.

The probe value of $|x_i\rangle = 17 : \approx 0.002946$.

The example of 10 times test: The read value of $|x_i\rangle$ = 54, 52, 60, 22, 0, 57, 61, 44, 14, 38. (= spike)

The candidates of number of spikes are estimated by the function [the function estimate_num_spikes (spike, range) [spike: read value, range: $2^n = 2^6 = 64$]:

54 → 6, 13, 19, 32 ; 52 → 5, 11, 16, 32, 48 ; 60 → 16, 32, 48 ; 22 → 3, 6, 9, 12, 15, 17, 20, 23, 26, 29, 32 ; 0 → nothingness ; 57 → 9, 18, 27, 37, 46, 55 ; 61 → 21, 43 ; 44 → 3, 6, 10, 13, 16, 32 ; 14 → 5, 9, 18, 23, 32 ; 38 → 3, 5, 10, 15, 17, 22, 27, 32.

When P is 17, $2^0x_1 + 2^1x_2 + 2^2x_3 + 2^3x_4 + 2^4x_5 + 2^5x_6$:

$2^0\times1 + 2^1\times0 + 2^2\times0 + 2^3\times0 + 2^4\times1 + 2^5\times0 = 17$.

There is $L_1x_1 + L_2x_2 + L_3x_3 + L_4x_4 + L_5x_5 + L_6x_6$:

$5\times1 + 3\times0 + 8\times0 + 7\times0 + 6\times1 + 1\times0 = 11$ ($= k \leq D = 11$).

Therefore, tasks of 2nd processor are 5 and 6. Still more, tasks of 3rd processer are 8 and 1 ($8 + 1 = 9 \leq D = 11$).

After all, answers are (3, 7), (5, 6), and (8, 1).

5. Discussion

In the knapsack problem, there are many combinations of luggage to obtain a value. In the minimum multiprocessor scheduling problem, there are several combinations of lengths of tasks to obtain a value.

Therefore, the search is difficult.

In section 4, there are n lengths of tasks. When N is 2^n, in the Grover's method, the complexity is $2\times N^{1/2} = 2\times 2^{n/2}$, in the Shor's Fourier transform, it is 2×(several times).

In $n = 6$, $2\times N^{1/2} = 2\times 2^{n/2} = 2\times 2^{6/2} = 2\times 8 = 16$, and 2×(several times) = 2×(10/2) = 2×5 = 10.

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 minimum multiprocessor scheduling problem by the Shor's Fourier transform with the RAM on the QCEngine, and its example are reported.

The complexity of this method is [(number of processors)-1]×(several times).

I will apply this method for other problems.

References

[1] Fujimura, T., 2012, "Quantum algorithm for minimum multiprocessor scheduling problem by central limit theorem," Glob. J. Pure Appl. Math., **8**, 183-189.

[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.