On-Chip High-Speed Solver of Inverse Problems Based on Quantum-Computing Principle

Minoru Fujishima
School of Frontier Sciences
The University of Tokyo
Kashiwa, Japan
fuj@mail.u-tokyo.ac.jp

Masahiro Shimura
School of Engineering
The University of Tokyo
Kashiwa, Japan
masahiro3.shimura@toshiba.co.jp

Abstract—A high-speed solver of an inverse problem to a one-way function becomes important although the problems become complicated according to the change in social situation. However, a chip multiprocessor dedicated to the inverse problem to the one-way function is nonexistent although a single-chip processor consuming limited power is useful in realizing a high-speed solver. In this paper, we propose a new chip multiprocessor operating with a procedure similar to that of quantum computing. The chip multiprocessor is implemented on a field programmable gate array (FPGA), and the factorization of a 64-bit integer is demonstrated. As a result, the proposed processor reduces the calculation time by 35% compared with a general-purpose processor with 3.4GHz clock frequency. As a result, it is shown that the proposed processor solves inverse problems such as factorization and discrete logarithm problems at higher speed than a general-purpose processor consuming limited power.

I. INTRODUCTION

There are many inverse problems to a one-way function in the world, and their high-speed solution is important. For instance, database retrieval is a problem of discovering the requested information from a huge amount of information. A route search problem, such as the traveling salesman problem, is applied to circuit design and other applications. A constraint satisfaction problem, such as the knapsack problem, is applied to obtain the maximum value under a certain restriction. These problems become complicated with changing social situation. Therefore, the significance of a high-speed single-chip processor to solve the inverse problem to the one-way function is increasing. However, it is difficult to improve the performance merely by increasing the operation frequency because of an increasing power consumption that may surpass the power limit determined by its package restriction [1]. To avoid this issue, a chip multiprocessor (CMP) [2,3] dedicated to each application is effective for improving the processing speed with limited power consumption. For instance, the MPEG-4 CODEC [4] is a CMP with the architecture to compress and expand MPEG-4 at high speed. Network processors [5] process packets in the internet and intranet at high speed. Vector processors [6] execute the vector operation for high-speed science and technology calculations. However, a CMP dedicated to the inverse problem to the one-way function is nonexistent. It is noted that the inverse problem is solved at high speed when the forward problem is calculated and the solution suitable for the request is retrieved in parallel using, for example, clustering and cell computing. However, these processors are unsuitable for integrating many processors under limited power consumption since the processor used for these computers contains a redundant circuit for the inverse problem.

In this paper, we focus on the nature of quantum computing to solve the forward problem and extract the answer in parallel by quantum superposition [7]. The condition of the chip multiprocessor necessary for emulating a procedure similar to that of quantum computing is explained to solve the inverse problem to the one-way function at high speed with limited power consumption. After an architecture based on dual-instruction multiple-data (DIMD) that satisfies the condition is proposed, the implementation of this processor on a field programmable gate array (FPGA) and the result of the factorization are described.

![Fig. 1. Operation steps for an inverse problem in quantum computing.](image)

(a) The state of all quantum bits is initialized to $|0\rangle$. (b) The probability is deconcentrated equally to all quantum-processing elements (QPEs). (c) Unitary operations are executed in parallel. (d) The probability is concentrated and the solution is retrieved. Figures and variables in each box are probability amplitudes.

---

1 Current affiliation is Broadband System LSI Development Department I, Broadband System LSI Development Center, Broadband System LSI Division, Toshiba Corporation Semiconductor Company.
II. PARALLEL COMPUTING THEORY IN QUANTUM COMPUTING

In quantum computing for solving the inverse problem, the procedures are categorized into four steps: (i) initialization, (ii) probability deconcentration, (iii) parallel unitary transformation, and (iv) probability concentration, as shown in Fig. 1. In (ii) and (iii), the quantum bit (qubit) is classified into two groups according to their roles: index qubit (iqubit) and data qubit (dqubit). Here, it is assumed that \( n_i \) iqubits are iqubits and \( n_d \) iqubits are dqubits in an \( n \)-qubit quantum computer. In this case, the entire quantum state \(|\phi\rangle\) is described as

\[
|\phi\rangle = \sum_{i=0}^{2^n-1} c_i |i\rangle = \sum_{j=0}^{2^n-1} c_j |j\rangle
\]

where \(|i\rangle\) and \(|j\rangle\) are the bases for the \( i \)-th iqubit and the \( j \)-th dqubit, respectively, and \( c_i \) and \( c_j \) are the probability amplitudes of \(|i\rangle\) and \(|j\rangle\). As described in (1), \(|\phi\rangle\) is considered to be the superposition of the probability amplitude vector \(\sum_{j=0}^{2^n-1} c_j |j\rangle\). Namely, each module of the index and its probability-amplitude vector is considered to comprise a quantum-processing element (QPE). The transition of the probability amplitude from (i) to (iv) is shown in Figs. 1 (a) to (d).

In step (i), \(|\phi\rangle\) is initialized to \(|0\rangle\). Namely, the probability amplitude for \(c_{00} = 1\) and the others are 0.

In step (ii), the Walsh-Hadamard transformation \([7] U_{WH}\) is applied to all the iqubits Thus, the probability amplitudes are distributed equally to \(2^n\) QPEs to establish an initial condition of parallel calculation in (iii).

In step (iii), the unitary transformation is applied to dqubits, and the forward problem is solved in parallel. Conditional operations are executed with controlled quantum gates[7]. Here, the probability amplitude of each QPE always satisfies the normalizing condition

\[
\sum_{j=0}^{2^n-1} |c_j|^2 = \frac{1}{2^n}
\]

In step (iv), the probability amplitude of a target base is increased according to the condition, and the solution of the inverse problem is extracted. Although Grover’s algorithm [8] is commonly used in quantum computing, quantum Fourier transformation (QFT) is used in Shor’s algorithm [9] to derive the answer of factorization.

III. SINGLE-INSTRUCTION MULTIPLE-DATA PROCESSOR WITH CONDITIONAL OPERATION

For an inverse problem, the calculation procedures for multiprocessor \( M_S \) are categorized into three steps: [i] initialization, [ii] parallel operation, and [iii] search, as shown in Fig. 2. \( M_S \) has \( n_i \) processing elements (PEs) which have \( n_d \) memories. Consequently, \( M_S \) has \( n_i n_d \) data in total. Here, the index \( i \) of PEs and the address \( j \) of memories are regarded as base vectors \(|i\rangle\) and \(|j\rangle\) to correlate with quantum computing. When all the data is converted to vectors, the entire state \(|\Psi_s\rangle\) of \( M_S \) is described as

\[
|\Psi_s\rangle = \sum_{i=0}^{n_i-1} \sum_{j=0}^{n_d-1} d_{ij} |i\rangle |j\rangle = \sum_{i=0}^{n_i-1} (|i\rangle \otimes \sum_{j=0}^{n_d-1} d_{ij} |j\rangle)
\]

where \( d_{ij} \) is the datum in the \( j \)-th address.

In the single-instruction multiple-data (SIMD) processor with conditional operation, the processing speed of the conditional branch required in solving the inverse problem is improved when the PE not satisfying the conditions can execute operations other than identity transformation. Two cases must be considered; one is that the conditions are based on the data vector and the other is that they are based on the PE index. In the former case,

\[
f_o = \text{cond} \cdot f_{o1} + (1 - \text{cond}) \cdot f_{o2},
\]

where \( f_{o1} \), \( f_{o2} \), and \( f_{o2} \) are the functions whose arguments are \( n_d \)-dimension data vectors, and \( \text{cond} \) is a logical value of 1 when the data vector satisfies a given condition and 0 when they do not. The PE satisfying the conditions applies the function \( f_{o1} \), while one not satisfying the conditions applies the function \( f_{o2} \), where \( f_{o1} \) and \( f_{o2} \) have the \( n_d \)-dimension data vectors as their arguments.

\[
f_s = ((E_S - C_s) \otimes f_{o2} + C_s \otimes f_{o1}),
\]

where \( C_s \) is an \( n \)-dimension diagonal matrix whose diagonal sections corresponding to the base satisfying the condition are 1 and the others are 0, and \( E_S \) is an \( n \)-dimension unit matrix.

In this case, the transition of \(|\Psi_s\rangle\) is described as

\[
|\Psi_s\rangle \rightarrow f_s(|\Psi_s\rangle)
\]

\[
= \sum_{i=0}^{n_i-1} \left( (E_i - C_s) \otimes f_{o2} \left( \sum_{j=0}^{n_d-1} d_{ij} |j\rangle \right) \right) + \sum_{i=0}^{n_i-1} C_s |i\rangle \otimes f_{o1} \left( \sum_{j=0}^{n_d-1} d_{ij} |j\rangle \right)
\]

The data vectors are mapped with different functions simultaneously. The processor executing these operations is the dual-instruction multiple-data (DIMGD) processor. The PEs are connected so that the binary search may be applied to retrieve the answer at high speed after the forward problems are calculated in parallel. The DIMGD processor has a controller as well as parallel PEs to control the operation procedure. Details of the DIMGD processor architecture are described in the following sections.

[i] initialization  [ii] parallel operation  [iii] search

![Fig. 2. The operation step in which the SIMD processor \( M_S \) with conditional operation is used. [i] All data is initialized to 0. [ii] The data in each PE is calculated with the same function in parallel. [iii] The answer is retrieved by a classical technique such as binary retrieval.](image-url)
A. Processing Element

The architecture of the PE is shown in Fig. 3. Each PE has an individual index, a 16-bit arithmetic logical unit (ALU), a 2,408-bit local memory with 16-bit bandwidth, two registers reg1 and reg2, and six flags Z, C, S, V, T, and E which are zero, carry, sign, overflow, true, and enable flags, respectively. The PE has an indirect addressing mode in order to read data from different addresses. To execute one of two operations described in (4) and (6), the instruction code for the PE has two operation codes (opcode1 and opcode2), one condition code, and three operands comprising one destination operand and two source operands. The operation to be executed in each operation step is verified by comparing the condition code and the flags. The PE executes the conditional operation with the same operands as shown in Fig. 4. The PE executes opcode1 when the condition determined by the flags is satisfied, and executes opcode2 when the condition is not satisfied.

The flag “T” is 1 when the PE executed opcode1 in the previous operation, and is 0 when it executed opcode2. The T flag is useful when multiple operations are executed while maintaining the flag conditions. Let us take the non-restoring division algorithm using a divisor with more than 16 bits for example. Each PE adds the lower 16-bit divisor to the lower 16 bits of the intermediate surplus or subtracts the divisor from the intermediate surplus. Then, the same operation applies to the higher 16 bits with the same condition. In this case, the T flag is used to preserve the previous condition although the values of the other five flags change at each operation. The E flag switches the modes of “enable” and “standby” according to the PE instruction. Complex conditional branches for more than two directions are realized by changing the E flag in each PE. One of two registers in the PE is a normal register, and the other is a shared-register with another PE. The shared register can be used for indirect addressing.

Finally, the retrieval result is extracted in the PE with the index of 0. The normal register in the PE with the index 0 is shared with the controller, which verifies the correctness of the extracted answer. Details are described in the following section.

B. Controller

Fig. 6. Architecture of the controller to manage instruction flows. The controller can read verification data from PE000. One of three registers sends data to all PEs.
The controller manages the calculation procedure of the PEs using a program counter and stack pointers. In the calculation of the inverse problem, since each PE only has to change its data vector using a common function, the calculation procedure need not be controlled one by one. All the PEs execute the same branch if necessary. Since the PEs and the controller operate different instructions simultaneously, the controller operates in the background of the PE calculations. The controller has a 16-bit ALU, three registers, five flags (Z,C,S,V,T), and a 4,096-bit local memory with 16-bit bandwidth, as shown in Fig. 6. Since the controller never sleeps from the beginning to the end of the calculation, there is no E flag. As described in section III.A, the controller reads data from the shared register of the PE with the index of 0. Although the calculation is finished if the target answer is found, it is repeated if no answer is found.

C. Implementation on FPGA

The DIMD processor described above is implemented in a FPGA with 1.5 million gates. The performance is summarized in Table I and the entire system is shown in Fig. 7. The DIMD processor is implemented in one of the two FPGAs, and the other FPGA is used as an instruction memory and memory controller. The communication between the memory and the processor is synchronized, and instructions for the PEs and the controller are issued simultaneously in one clock. There are 192 PEs implemented in the FPGA. In order to execute basic arithmetic logical operations, the instruction set shown in Table II is prepared.

Using the DIMD processor, a 64-bit integer is factorized for the example of the inverse problem to the one-way function. For comparison, the same integer is factorized using a general-purpose processor with a 3.4 GHz clock frequency. As a result, the DIMD processor finishes the factorization in 34 seconds, while the general purpose processor requires 52 seconds. Namely, the DIMD processor reduces the operation time by 35%. In the discrete logarithm problem of finding r in \( g^r \mod p = x \) when g and x are given as being smaller than p, r can be obtained at high speed if \( g^r \mod p \) is calculated using the PEs, and the index of the PE equal to x is retrieved.

### Table I. Specifications of the DIMD Processor

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>APEX 20K1500E (1.5M gates)</td>
</tr>
<tr>
<td>Gate usage in processor</td>
<td>1.2M gates</td>
</tr>
<tr>
<td>Number of PEs</td>
<td>192</td>
</tr>
<tr>
<td>Clock frequency</td>
<td>40 MHz</td>
</tr>
<tr>
<td>Data bit width</td>
<td>16 bits</td>
</tr>
<tr>
<td>Instruction bit width</td>
<td>56 bits</td>
</tr>
</tbody>
</table>

Remark: SD represents a destination operand, and $S1$ and $S2$ represent source operands respectively. “C” represents a carry flag. (M) represents a data from data memory shown in Fig. 7.

### Table II. Instruction Set for DIMD Processor

<table>
<thead>
<tr>
<th>Instruction for PEs</th>
<th>Instruction for a controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD SD, $S1$, $S2$</td>
<td>JMP Jump to absolute address</td>
</tr>
<tr>
<td>ADC SD, $S1$, $S2$</td>
<td>BR Jump to relative address</td>
</tr>
<tr>
<td>SUB SD, $S1$, $S2$</td>
<td>PUSH $S1$ Push $S1$ to stack</td>
</tr>
<tr>
<td>SBB SD, $S1$, $S2$</td>
<td>POP SD Pop stack to $SD$</td>
</tr>
<tr>
<td>SBI SD, $S1$, $S2$</td>
<td>CALL Call subroutine</td>
</tr>
<tr>
<td>SBIB SD, $S1$, $S2$</td>
<td>RET Return from subroutine</td>
</tr>
<tr>
<td>AND SD, $S1$, $S2$</td>
<td>LD SD, (M) SD = (M)</td>
</tr>
<tr>
<td>OR SD, $S1$, $S2$</td>
<td>ST (M), $S1$, (M) = $S1$</td>
</tr>
<tr>
<td>MOV SD, $S1$</td>
<td>PE sleep</td>
</tr>
<tr>
<td>SRL SD, $S1$</td>
<td>WUP PE wake up</td>
</tr>
</tbody>
</table>

### REFERENCES