数理解析研究所講究録
第 556 巻 1985 年 93-103

Redundant Coding and Local Computability in Parallel Computation

京都大学工学部 安浦 寛人 (Hiroto Yasuura)

京都大学工学部 高木 直史 (Naofumi Takagi)

京都大学工学部 矢島 脩三 (Shuzo Yajima)

1. Introduction

In the theory of computational complexity, it is well known that coding schemes play important roles and a difference of coding scheme often causes a drastic discrepancy of time or space complexity. Many famous efficient algorithms for sequential computation owe to discovery of good coding schemes.

In parallel computation, it also seems that the discovery of good coding schemes is a key to developping efficient algorithms. Avizienis pointed out the advantages of redundant coding schemes in design of high-speed arithmetic circuits [AVIZ6109]. In recent years, we have been designed several VLSI oriented hardware algorithms for arithmetic operations using a redundant binary coding scheme [TAKAY8306] [TAKAY8404] [TAKAY8402] [TAKAA8501]. These results suggest that there are some possibilities for us to design very high-speed hardware algorithms in various areas of computer science using redundant coding techniques.

In the redundant binary coding, we can construct a carry-propagation-free adder. Namely, we can compute each digit of the sum from only each three digits of the addend and the augend in this addition rule. Thus addition of two numbers can be done by a constant depth circuit independent of the length of the operands. It is clearly impossible to construct such a fast addition algorithm when we use the ordinary binary representation. In this case, since the most significant digit of the sum depends on all digits of the addend and the augend, the depth of circuits should be at least $\Theta(\log n)$ on the assumption that fan-in of logic elements is restricted to a constant number. Moreover, Winograd showed that one can not construct a constant depth adder using any nonredundant coding scheme [WINO6504].

Our inevitable question, which is mainly discussed in this paper, is for what kinds of operations we can construct efficient parallel algorithms that are realized by constant depth circuits. Of cause, we allow the usage of redundant coding techniques.

In order to clarify the relation between coding schemes and the computational complexity on combinational logic circuits, we will introduce a new concept called '<u>local computability</u>'. The local computability is defined by the number of code digits of operands required to determine each digit of the result. Therefore our question will be reduced to what kinds of operations have coding schemes under which they are k-locally computable for some constant k independent of the size of the domain of operations.

The main result of this paper is that any operation of finite <u>Abelian group</u> is k-locally computable under certain redundant coding scheme where k depends only on the size of an alphabet of coding independent of the length of each code.

2. Local Computability

2.1 Computation Model

In this paper, we adopt combinational circuits constructed with fan-in restricted logic elements as a model of parallel computation. A combinational circuit is represented by a directed acyclic graph whose vertices and edges correspond to logic elements (or terminals) and connecting lines (wires) in the circuit, respectively. The complexity of a combinational circuit is measured by its size (the number of logic elements included in it) and depth (the length of the longest path in it).

2.2 Coding Scheme

Let S be a finite set and $\circ$ be a binary operation defined on S. Assume that S is closed under $\circ$. We denote the number of elements in S by $|S|$. Let an alphabet A be a finite set of symbols. A^n represents a set of strings on A with length n. We encode elements in S into strings on A of length n as follows. In this paper, we only consider with fixed-length codes.

[Definition 1] C is a coding scheme for $(S,\circ)$ on A if and only if the following two conditions are satisfied:

(1) $C: A^n \to S \cup \{\perp\}$, where $\perp$ is not in S.

(2) For any element s in S, there is at least one element x in A^n such that $C(x)=s$.

We can define a binary operation $*$ on A^n such that

$C(x*y)=C(x)\circ C(y)$ for any x and y such that both $C(x)$ and $C(y)$ are in S. Namely, S is homomorphic to $\{x|\ x\in A^n$ and $C(x)\in S\}$. In this paper, we assume that $|A|>1$.

Since a coding scheme is defined as a mapping from a code space A^n to the original set $S \cup \{\perp\}$, we can specify a redundant coding scheme. A coding scheme C is said to be <u>redundant</u> if there is an element in S which is an image of two or more elements in A^n.

2.3 Local Computability

Here we define local computability of operations.

[Definition 2] A binary operation $\circ$ on S is <u>k-locally computable</u> under a coding scheme $C : A^n \to S\cup\{\perp\}$ if and only if there exists an operation $*$ on A^n such that in the computation of $z=x*y$ each z_i is a function of at most k elements in $\{x_1, x_2, \ldots , x_n, y_1, y_2, \ldots , y_n\}$, where $x=x_1x_2\ldots x_n$, $y=y_1y_2\ldots y_n$ and $z=z_1z_2\ldots z_n$.

If an operation is k-locally computable under some coding scheme, one can construct a circuit computing the operation with depth $O(k)$. From the view point of high-speed computation, we are interested in the k-locally computable operation such that k is independent of the set size.

3. Residue Class Group Z_m and Redundant Binary Coding

Let Z_m be a residue class modulo m in the set of integers, where + is usual integer addition to modulus m. It is well known that $(Z_m, +)$ is a cyclic group. Namely, Z_m is a finite group and all elements in Z_m is generated from '1'.

Now we can show the following fact.

[Lemma 1] For any positive integer m, there is a coding scheme C for $(Z_m,+)$ on $\{0, 1, -1\}$ such that + is 16-locally computable under C.

(Proof) Let alphabet A be $\{0, 1, -1\}$ and r be $\lceil \log_2 m \rceil$. Then $2^{r-1} < m \leq 2^r$. Consider the following coding scheme.

$$C : A^r \rightarrow Z_m \cup \{\perp\}$$

$$C(x_1 x_2 \ldots x_r) = \begin{cases} s & \text{if } 0 \leq s < m, \\ m + s & \text{if } -m < s < 0, \\ \perp & \text{otherwise.} \end{cases}$$

where

$$s = \sum_{i=1}^{r} x_i 2^{r-i}$$

Next we define a binary operation * on the coding space A^r. We must define the operation * such that z=x*y iff C(z)=C(x)+C(y) for any elements x and y whose images are included in Z_m. Let x, y and z in A^r be denoted by $x_1x_2 \ldots x_r$, $y_1y_2 \ldots y_r$ and $z_1z_2 \ldots z_r$, respectively. We will give a computation rule of * such that each z_i depends on 16 digits in $x_1x_2 \ldots x_r$ and $y_1y_2 \ldots y_r$. Before discussing the operation *, we first introduce a carry-propagation-free addition on the redundant coding.

[Procedure ADD] (p is a positive integer)

Input $x_1x_2...x_p$ and $y_1y_2...y_p$.

Output $u_0u_1...u_p$, where $\sum_{i=0}^{p} u_i 2^{p-i} = \sum_{i=1}^{p} x_i 2^{p-i} + \sum_{i=1}^{p} y_i 2^{p-i}$.

(Here + is the ordinary arithmetic addition.)

Step 1. Generate the intermediate sum $s_1s_2...s_p$ and the intermediate carry $c_0c_1...c_{p-1}0$ from $x_1x_2...x_p$ and $y_1y_2...y_p$ according to the addition rule in Table 1.

Step 2. Add $s_1s_2...s_p$ and $c_0c_1...c_{p-1}0$. The sum is denoted by $u_0u_1...u_p$.

x_i	y_i	x_{i+1} y_{i+1}	c_{i-1}	s_i
1	1	- -	1	0
1 0	0 1	containing 1	1	-1
		not containing 1	0	1
0 1 -1	0 -1 1	- -	0	0
-1 0	0 -1	not containing -1	0	-1
		containing -1	-1	1
-1	-1	- -	-1	0

Table 1. Addition Rule

Since, no carry is generated in Step 2, each u_i is a function of only x_i, x_{i+1}, x_{i+2}, y_i, y_{i+1} and y_{i+2}. Using this addition procedure, we define * as follows:

[Procedure MODADD]

Input $x=x_1x_2...x_r$ and $y=y_1y_2...y_r$.

Output $z=z_1z_2...z_r$, where $\sum_{i=1}^{r} z_i 2^{r-i} = \sum_{i=1}^{r} x_i 2^{r-i} + \sum_{i=1}^{r} y_i 2^{r-i}$ mod m.

Step 1. Let $u = u_0u_1...u_r$ = ADD(x, y).

Step 2. If $m=2^r$ then $z=u_1u_2...u_r$ and stop, otherwise go to Step 3.

Step 3. According to the value of u_0 and u_1, select one of the following operations:

```
If $(u_0, u_1) = (0,0)$, $z=u_1u_2...u_r$.
If $(u_0, u_1) = (0,1)$ or $(1,-1)$,
  (1) If $(1,-1)$ then rewrite it into $(0,1)$.
  (2) $v_{-1}v_0v_1...v_r$ = ADD(u, -m).
  (3) $z=v_1v_2...v_r$.
If $(u_0, u_1) = (0,-1)$ or $(-1,1)$,
  (1) If $(-1,1)$ then rewrite it into $(0,-1)$.
  (2) $v_{-1}v_0v_1...v_r$ = ADD(u, m).
  (3) $z=v_1v_2...v_r$.
If $(u_0, u_1) = (1,0)$,
  (1) $v_{-1}v_0v_1...u_r$ = ADD'(u,-m).
  (2) $z=v_1v_2...v_r$.
If $(u_0, u_1) = (-1,0)$,
  (1) $v_{-1}v_0v_1...u_r$ = ADD''(u, m).
  (2) $z=v_1v_2...v_r$.
If $(u_0, u_1) = (1,1)$,
  (1) $v_{-1}v_0v_1...u_r$ = ADD'(u,-2m).
  (2) $z=v_1v_2...v_r$.
If $(u_0, u_1) = (-1,-1)$,
  (1) $v_{-1}v_0v_1...u_r$ = ADD''(u, 2m).
  (2) $z=v_1v_2...v_r$.
```

where

(a) $m = m_0m_1...m_r$ is represented by the ordinary binary representation and $-m = (-m_0)(-m_1)...(-m_r)$. Note that $m_0=0$. 2m and -2m are represented by $m_1m_2...m_r0$ and $(-m_1)(-m_2)...(-m_r)0$, respectively.

(b) ADD' and ADD'' are modification of ADD. In ADD'(ADD''), if

$(x_i, y_i) = (1,0)$ or $(0,1)$ $((-1,0)$ or $(0,-1))$ then $(c_{i-1}, s_i) = (0,1)$ $((0,-1))$. Moreover, (c_0, s_1) is uncoditionally $(-1, 1)$ $((1,-1))$.

It is easy to show that the operation $*$ defined by MODADD satisfies the condition of $C(x*y)=C(x)+C(y)$. Since in the computation of ADD, ADD' and ADD'' each sum digit depends on at most only 6 digits of input operands, z_i, the i-th digit of final result of MODADD, depends on x_1, x_2, x_3, y_1, y_2, y_3 (u_0 and u_1 are computed from them),and x_i, x_{i+1}, x_{i+2}, x_{i+3}, x_{i+4}, y_i, y_{i+1}, y_{i+2}, y_{i+3}, and y_{i+4}. Thus + is 16-locally computable under C.

Q.E.D.

4. Local Computability of Abelian Group Operations

From the result of the previous section, we can directly deduce the following Lemma.

[Lemma 2] For any cyclic group$(S,\circ)$ and any alphabet A, there is a coding scheme C on A such that $\circ$ is k-locally computable under C, where k is a constant depending only on A.

(Proof) Since any cyclic group $(S,\circ)$ is isomorhic to $(Z_m,+)$ where $m=|S|$, this lemma is directly deduced from Lemma 1 in section 3.

Q.E.D.

It is known in algebra that any finite Abelian group (i.e. finite commutative group) $(S,\circ)$ is a direct product of finite cyclic groups. So we can directly derived the following theorem.

[Theorem] For any finite Abelian group $(S, \circ)$, there is a coding scheme C on any alphabet A such that $\circ$ is k-locally computable under C, where k is a constant depending only on A.

[Corollary] For any positive integer p, there is a coding scheme C for $(Z_p - \{0\}, \times)$ on any alphabet A such that $\times$ is k-locally computable under C, where k is a constant independent of p and $\times$ is the normal integer multiplication to modulus p.

(Proof) It is known in the group theory that $(Z_p - \{0\}, \times)$ is a finite Abelian group. So using logarithmic notation and redundant coding technique, we can easily construct a k-locally computable coding scheme for any p and A under which $\times$ is k-locally computable where k is independent of p. Q.E.D.

5. Discussions

For the definition of local computability, we can introduce more conditions from the view point of practical algorithm design.

(1) Efficiency

In general, a redundant code is longer than a nonredundant code. It is desired that the length of code is as short as possible for reducing the hardware resources. When efficiency of coding scheme $C : A^n \to S$ is defined by $n / \lceil \log_a |S| \rceil$ where $a = |A|$, the efficiency of practical coding should be bounded by a small constant.

(2) Homogeneity

From the standpoint of hardware algorithm design, it is

.02

desired that, in the computation of $z=x*y$, each function f_i computing each digit z_i is homogeneous.

(3) Consecutiveness

In the design of VLSI oriented algorithms, it is important that operands of each f_i are consecutive in the strings of $x_1x_2...x_n$ and $y_1y_2...y_n$, since the area for wiring may be small for such coding schemes.

(4) Universality

The concept of local computability should be extended for a set S and a set of operations on S. Namely, we want to have a coding scheme in which each operation is realized by local computation. For example, local computability of operations in rings or finite fields should be investigated.

(5) Code Conversion

In the practical use, the computational complexity of code conversion is a very important factor of total efficiency. Especially, conversion between a redundant coding scheme and a usual coding such as the binary representation should be considered carefully in the design of the redundant coding scheme for high-speed computation.

Acknowledgment

The authors would like to express their appreciation to Dr.Hiraishi and members of Yajima's Lab. of Kyoto University for their helpful discussions. This work is partly supported by the science foundation grant of the Ministry of Education of Japan.

References

[AVIZ6109] A.Avizienis: 'Signed-Digit Number Representations for Fast Parallel Arithmetic,' IRE Trans. Elec. Comp., EC-10, 3, pp.389-400, Sept. 1961.

[TAKAA8501] N.Takagi, T.Asada and S.Yajima: 'A Hardware Algorithm for Trigonometric Functions Using Redundant Binary Representation,' IECEJ, Tech. Report AL84-59, Jan. 1985 (in Japanese).

[TAKAY8306] N.Takagi, H.Yasuura and S.Yajima: 'A VLSI-Oriented High-Speed Multiplier Using a Redundant Binary Addition Tree,' Trans. IECEJ J66-D, 6, pp.683-690, June 1983 (in Japanese).

[TAKAY8402] N.Takagi and S.Yajima: 'Hardware Algorithms for Logarithmic and Exponential Functions Using Redundant Binary Representation,' IECEJ, Tech. Report AL83-70, Feb. 1984 (in Japanese).

[TAKAY8404] N.Takagi, H.Yasuura and S.Yajima: 'A VLSI-Oriented High-Speed Divider Using Redundant Binary Representation,' Trans. IECEJ J67-D, 4, pp.450-457, April 1984 (in Japanese).

[WINO6504] S.Winograd: 'On the Time Required to Perform Addition,' JACM 12, 2, pp.277-285, April 1965.