Parallel Addition for Double Base Number System

Wutthipat chalermchatwichien[1], and Athasit Surarerks[2]
ELITE (Engineering Laboratory in Theoretical Enumerable System),
Department of Computer Engineering, Faculty of Engineering,
Chulalongkorn University, Pathumwan, Bangkok 10330, Thailand
[1]Wutthipat.C@student.chula.ac.th
[2]athasit.s@chula.ac.th

***Abstract*— Double base number system (DBNS) is an alternative number system besides the binary system. Its representation is similar to the radix number system together with two bases, usually be two and three. DBNS preserves the two important properties: redundancy and sparseness. The redundancy is the property accommodating with the parallelism. In this research, we are interested in parallel addition algorithm on DBNS. Our theoretical result shows that parallel addition in DBNS can be performed. An addition algorithm together with the proof of correctness is described in this paper. In generally, we study the generalization form of DBNS addition algorithms in any sizes.**

Keywords—Double Base Number System, Parallel Addition, Redundancy Property

I. INTRODUCTION

Arithmetic computation plays an important role in arithmetic and logic unit (ALU) part of central processing unit (CPU). Binary system is classical used for number representations in ALU in order to do arithmetic operations. This system is simple but hard to perform any parallel arithmetic operations. However, there are many proposed number systems which can speed up the computational time. The most efficiency technique is to apply a signed-digit concept to the representation. The system had first proposed by Avizienis in [1]. The concept of signed-digit number representation system is to limit the carry propagation chain in fundamental arithmetic operations (i.e., addition, subtraction, multiplication, and division), see detail in [2]. The signed-digit number system is demonstrated to be able to process some arithmetic operations in parallel manner. On the other side, some researches studied on how to pipeline all arithmetic operations in this system. It is proved to be realized if all operations perform in an on-line manner, all operation serially performs from the most significant digit to the least significant digit, digit-by-digit, see [3].

Classical technique used for fast computing (especially for parallel computing) is a redundancy property. Redundancy means that a number can have more than one representation in the system. This technique is proved to be able to reduce the carry propagation chain, which can speed up the time complexity of the operations.

Using the concept of redundancy, double-base number representation system (DBNS) designed by V.S. Dimitrov, G.A. Jullien and W.C. Miller in 1997 in [4] has been proposed in order to reduce the number of 1's digits in the representation. Any number in double-base representation is illustrated by two bases: two and three. Moreover, a number can also have more than one representation. Fundamental arithmetic operations such as addition, subtraction, multiplication, and division have been proposed in [4].

In this research we introduce a parallel algorithm for DBNS addition operation. We consider the representation of number as a two-dimensional table with respect to the bases two and three. Each table (representation) can be considered as the combination of several square sub-tables. Our technique is to propose an algorithm for performing addition of all sub-tables in the same time. Theoretical result shows that addition can be solved in the constant time complexity.

This paper is organized as follows: Section II gives some background knowledge on double base number representation with notations and definitions used in this paper. We proposed an algorithm for parallel addition in DBNS in Section III. Section IV gives an example for parallel addition. Finally, the conclusion is found in the last section.

II. DOUBLE BASE NUMBER SYSTEM

The double base number system is a redundant system that has been first proposed by V.S. Dimitrov, G.A. Jullien and W.C. Miller [4]. The following definition describes the DBNS representation system.

Definition 1 Double base number representation system (DBNS) *is a number representation system using the exponential form of bases 2 and 3 with digits in D = {0, 1}. For an arbitrary integer m, it can be represented in this system by* $m = \Sigma\, d_{i,j}2^i3^j$ *such that* $d_{i,j} \in \{0, 1\}$ *with integers i and j.*

It is obviously that any integer can have a representation in this system; any binary representation is also of the form of DBNS by setting *j* to zero. Since a representation contains two bases, the number is usually illustrated by a table as shown in Example 1. An algorithm for generating a representation of numbers in this system is proposed in [5-6] using the greedy technique, named *multipath greedy* algorithm.

978-1-4799-0806-6/13/$31.00 ©2013 IEEE

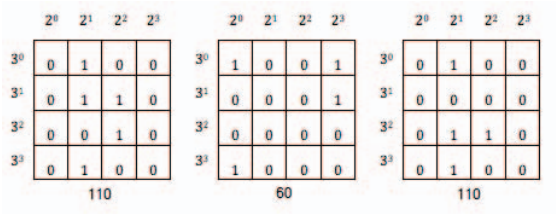

Fig. 1 The examples of DBNS representation.

Example 1 *Integers 60 and 110 can be represented in double base number representation system.*

Consider

$$110 = 2^1 3^0 + 2^1 3^1 + 2^2 3^1 + 2^2 3^2 + 2^1 3^3$$

and

$$60 = 2^0 3^0 + 2^3 3^0 + 2^3 3^1 + 2^0 3^3.$$

To simplify the representation of a number, we usually express it as an array $n \times n$ as shown in Figure 1. □

Since DBNS is a redundancy system (*i.e.*, at least one number in the DBNS has more than one representation,) as illustrated by Figure 1. For instance, 110 can have at least two representations. These are

$$110 = 2^1 3^0 + 2^1 3^1 + 2^2 3^1 + 2^2 3^2 + 2^1 3^3,$$
$$110 = 2^1 3^0 + 2^1 3^2 + 2^2 3^2 + 2^1 3^3.$$

Addition of two numbers in DBNS can be realized by performing addition of all corresponding (same degree of the both bases) cells simultaneously.

There are three important reduction rules to simplify the table representation. First, the two consecutive one's in a row can be transformed to the one's at the below row and the left one's column. Second, the two consecutive in a column can be transformed to the one's at the two next column and the above one's row. Last, the overlapped one's (can be replaced with two's) can be transformed to the one's at the right next cell.

III. Parallel Addition on DBNS in General Form

In this section, we consider a technique for performing addition in parallel manner. Since DBNS satisfies the redundancy property which causes the proposed parallel algorithm. Our technique is to group a square number of cells together and evaluate the numerical value of each square. Then, calculate the carry-out from the proposed equations and bring the carry-out of the above square to be the carry-in for every *n*-square simultaneously. Consequently, we get the full *Result* table representation by combining every *n*-square together. We show in this section how parallel addition can be done. The following theorem demonstrates that addition can be operated in constant time.

Theorem 1 *Addition of two numbers in DBNS can be performed by a constant time algorithm.*

Proof In order to prove the theorem, we introduce an algorithm for addition together with its proof of correctness. The concept of the algorithm is to divide the representation into several groups of n×n-cells as shown in the Figure 2. Addition of each corresponding group must be done simultaneously. The Figure 2 also shows the flow of carry propagation and the algorithm is shown as follows:

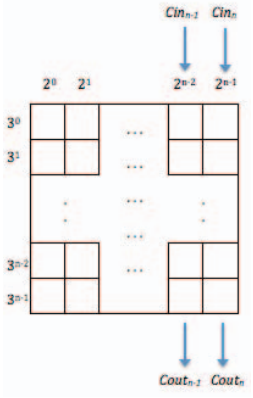

Fig. 2 The *n*-square and its carry propagation flow.

Algorithm: Parallel Addition

Input: $A=\sum_{i=0,j=0}^{cn-1} a_{i,j} 2^i 3^j$ and

$B=\sum_{i=0,j=0}^{cn-1} b_{i,j} 2^i 3^j$ and $a_{i,j}, b_{i,j} \in \{0, 1\}$

Output: $Z=\sum_{i=0,j=0}^{cn-1} z_{i,j} 2^i 3^j$ and $z_{i,j} \in \{0, 1\}$

Note

The inputs and the output are in table representation form. Let *i*, *j* be the coordinate of each cell in the original tables and *k*, *l* is the coordinate of each *n*-square related with the full table.

Begin

Step1 Perform addition of A and B, cell-by-cell, in parallel.

$$x_{i,j} = a_{i,j} + b_{i,j}$$

Step2 Evaluate the carry-out numerical value of each *n*-square simultaneously.

$$Sum_{k,l} = \sum_{i=0,j=0}^{n-1} x_{i,j} 2^i 3^j$$

$$Cout_{n_{k,l}} = \left\lfloor \frac{Sum_{k,l}}{2^{n-1} 3^n} \right\rfloor$$

$$Cout_{n-1_{k,l}} = \left\lfloor \frac{Sum_{k,l}}{2^{n-2} 3^n} \right\rfloor - 2Cout_{n_{k,l}}$$

Step3 Take all of *Couts* become *Cins* of the below *n*-square simultaneously related with their own *k.l* coordinate. Then, map the numerical value of each $Result_{k,l}$ to be in table representation form(*n*-square). At last, we get the last *Result* table from combining every *n*-square.

$$Result_{k,l} = Sum_{k,l} + Cin_{k,l} - Cout_{k,l}$$

where $Cout_{k,l} = 2^{n-2} 3^n Cout_{n-1_{k,l}} + 2^{n-1} 3^n Cout_{n_{k,l}}$

End

Proof of correctness: We show that the output is equal to the sum of two input operands as shown in the Figure 3 below.

$$\sum_{i=0,j=0}^{cn-1} x_{i,j}\, 2^i 3^j$$

$$= Sum_{0,0} + 2^0 3^n Sum_{1,0} + 2^0 3^{2n} Sum_{2,0} + \ldots + 2^0 3^{n(c-1)} Sum_{c-1,0} + 2^n 3^0 Sum_{0,1} + 2^n 3^n Sum_{1,1} + 2^n 3^{2n} Sum_{2,1} + \ldots + 2^n 3^{n(c-1)} Sum_{c-1,1} + \ldots + 2^{n(c-1)} 3^{n(c-1)} Sum_{c-1,c-1}$$
$$= (Sum_{0,0} + Cin_{0,0} - Cout_{0,0}) + 2^0 3^n (Sum_{1,0} + Cin_{1,0} - Cout_{1,0}) + \ldots + 2^0 3^{cn} Cin_{c,0} + \ldots + 2^{n(c-1)} 3^{n(c-1)} (Sum_{c-1,c-1} + Cin_{c-1,c-1} - Cout_{c-1,c-1}) + 2^{n(c-1)} 3^{cn} Cin_{c,c-1}$$
$$= (Sum_{0,0} + 0 - Cout_{0,0}) + 2^0 3^n (Sum_{1,0} + Cin_{1,0} - Cout_{1,0}) + \ldots + 2^0 3^{cn} Cin_{c,0} + \ldots + 2^{n(c-1)} 3^{n(c-1)} (Sum_{c-1,c-1} + Cin_{c-1,c-1} - Cout_{c-1,c-1}) + 2^{n(c-1)} 3^{cn} Cin_{c,c-1}$$
$$= Result_{0,0} + 2^0 3^n Result_{1,0} + \ldots + 2^{n(c-1)} 3^{n(c-1)} Result_{c-1,c-1} + \sum_{l=0}^{c-1} 2^{nl} 3^{cn} Cin_{c,l}$$
$$= Result + \sum_{l=0}^{c-1} 2^{nl} 3^{cn} Cin_{c,l}$$

Fig. 3 The proof of correctness equations, when $\sum_{i=0,j=0}^{cn-1} x_{i,j} 2^i 3^j$ is the *Sum* table archived by overlapping two operands together.

The result is the $cn \times cn$-size table and is in the DBNS representation form.

Proof of validation: We have to demonstrate that the result must be preserved the double base number representation. To simplify the proof, we will define some notations as follows:

Notation 1 *The numerical value of DBNS table representation which every cells contain bit one's is substituted by* $[1]_n$ *and the value of DBNS table representation which every cells contain bit two's is substituted by* $[2]_n$. *When n is the dimension of the* $n \times n$ *table representation.*

For each *n*-square, the numerical values of every variable are as follows:

$$\begin{aligned}
[1]_n &= (2^n - 1)(3^n - 1)/2 \\
[2]_n &= (2^n - 1)(3^n - 1) \\
Cout_n &= 2^{n-1}3^n \\
Cout_{n-1} &= 2^{n-2}3^n \\
Cout_{n-2} &= 2^{n-3}3^n \\
Cin_n &= 2^{n-1} \\
Cin_{n-1} &= 2^{n-2}
\end{aligned}$$

Note that each variable is related to each *n*-square and used as the variable of *k*, *l* for *n*-square (as in Figure 2).

Equation 1 $Result_{k,l} = Sum_{k,l} + Cin_{k,l} - Cout_{k,l}$

Function 1 $Cout_{n_{k,l}} = \left\lfloor \frac{Sum_{k,l}}{2^{n-1}3^n} \right\rfloor$

Function 2 $Cout_{n-1_{k,l}} = \left\lfloor \frac{Sum_{k,l}}{2^{n-2}3^n} \right\rfloor - 2Cout_{n_{k,l}}$

In order to demonstrate that the equations in general form above are correct and independent on the size $n \times n$ when the $Sum_{k,l}$ table is $n \times n$, the carry-in are Cin_n , Cin_{n-1} or both and the carry-out are $Cout_n$, $Cout_{n-1}$ or both, we have to show that

The $Cout_{n-1}$ and $Cout_n$ are 0 when $Sum_{k,l}$ is 0 to $2^{n-2}3^n - 1$. To show that $Result_{k,l}$ is still in DBNS representation form although there are both Cin_n and Cin_{n-1}, we have to prove that

$$\begin{aligned}
&Cin_n + Cin_{n-1} + 2^{n-2}3^n - 1 \le [1]_n; \\
&2^{n-1} + 2^{n-2} + 2^{n-2}3^n - 1 \le (2^n - 1)(3^n - 1)/2 \\
&2^{n-2}(3 + 3^n) - 1 \le (2^{n-1} - 0.5)\,)(3^n - 1)
\end{aligned}$$

It is obvious that this inequality is true. When n is an arbitrary integer that $n \ge 2$.

The $Cout_{n-1}$ is 1 and the $Cout_n$ is 0 when the $Sum_{k,l}$ is more than or equal $2^{n-2}3^n$ but not reach $2^{n-1}3^n$. Consequently, we have to prove the two inequalities below.

We have to prove that $Cout_{n-1}$ is enough to make the representation been in DBNS representation form.

$$\begin{aligned}
&(Cout_n - 1) - Cout_{n-1} + Cin_n + Cin_{n-1} \le [1]_n ; \\
&2^{n-1}3^n - 1 - 2^{n-2}3^n + 2^{n-1} + 2^{n-2} \le (2^n - 1)(3^n - 1)/2 \\
&2^n 3^n - 2 - 2^{n-1}3^n + 2^n + 2^{n-1} \le 2^n 3^n - 2^n - 3^n + 1 \\
&2^{n-1}3^n - 3^n - 2^{n+1} - 2^{n-1} + 3 \ge 0
\end{aligned}$$

It is obvious that this inequality is true. When n is an arbitrary integer that $n \ge 2$.

The $Cout_{n-1}$ must be less than or equal to $[1]_n$ in case of making sure that when $Sum_{k,l}$ is greater than $[1]_n$ and not reach $2^{n-1}3^n$, there is at least the carry-out $Cout_{n-1}$ to make the $Result_{k,l}$ been in DBNS representation form. Consequently, we have to prove that

$$\begin{aligned}
&Cout_{n-1} \le [1]_n;\ 2^{n-2}3^n \le (2^n - 1)(3^n - 1)/2 \\
&2^{n-1}3^n \le 2^n 3^n - 2^n - 3^n + 1 \\
&2^{n-1}3^n - 2^n - 3^n + 1 \ge 0
\end{aligned}$$

It is obvious that this inequality is true. When n is an arbitrary integer that $n \ge 2$.

The $Cout_n$ is 1 when $Sum_{k,l}$ is greater than or equal to $2^{n-1}3^n$. In case of making sure that the $Result_{k,l}$ is in DBNS representation form, we have to prove the two inequalities below.

If the carry-out is only $Cout_n$ and $Sum_{k,l}$ is maximum, it is enough to make the $Result_{k,l}$ been in DBNS representation form.

$[2]_n - Cout_n + Cin_n + Cin_{n-1} \le [1]_n$;
$(2^n - 1)(3^n - 1) - 2^{n-1}3^n + 2^{n-1} + 2^{n-2} \le (2^n - 1)(3^n - 1)/2$
$2^{n-1}3^n - 2^{n-1} - 2^{n-2} \ge 2^{n-1}3^n - 2^{n-1} - 0.5(3^n) + 0.5$
$2^{n-1} - 3^n + 1 \le 0$

It is obvious that this inequality is true. When n is an arbitrary integer that $n \ge 2$.

Then, the $Cout_n$ must be less than or equal $[2]_n$ in order to make the $Result_{k,l}$ not to be less than 0.

$Cout_n \le [2]_n$; $2^{n-1}3^n \le (2^n - 1)(3^n - 1)$.

It is obvious that this inequality is true. When n is an arbitrary integer that $n \ge 2$.

From Function 1, the $Cout_{n_{k,l}}$ must be only either 0 or 1.

$[2]_n / Cout_n < 2$; $(2^n - 1)(3^n - 1) / (2^{n-1}3^n) < 2$
$(2^n3^n - 2^n - 3^n + 1) / (2^{n-1}3^n) < 2$
$2 - 2 / 3^n - 1 / 2^{n-1} + 1 / (2^{n-1}3^n) < 2$

It is obvious that this inequality is true. When n is an arbitrary integer that $n \ge 2$.

From Function 2, when $Sum_{k,l}$ is maximum, the value of $\left\lfloor \frac{Sum_{k,l}}{2^{n-2}3^n} \right\rfloor$ must not be greater than or equal to 4 to fix the value of $Cout_{n-1_{k,l}}$ to be only either 0 or 1.

$[2]_n / Cout_{n-1} < 4$; $(2^n - 1)(3^n - 1) / (2^{n-1}3^n) < 4$
$(2^n3^n - 2^n - 3^n + 1) / (2^{n-2}3^n) < 4$
$4 - 4 / 3^n - 1 / 2^{n-2} + 1 / (2^{n-2}3^n) < 4$

It is obvious that this inequality is true. When n is an arbitrary integer that $n \ge 2$.

If we consider Function 1 and 2, we will find that if $Sum_{k,l} \ge 3\ Cout_{n-1}$, the $Cout_{n_{k,l}}$ will be 1 and the $\left\lfloor \frac{Sum_{k,l}}{2^{n-2}3^n} \right\rfloor$ will be 3 that causes the carry- out to be both $Cout_n$ and $Cout_{n-1}$. Consequently, we have to prove that the sum value of $Cout_n$ and $Cout_{n-1}$ must be less than or equal $3Cout_{n-1}$ in order to make the $Result_{k,l}$ not been less than 0.

$Cout_n + Cout_{n-1} \le 3Cout_{n-1}$
$2^{n-1}3^n + 2^{n-2}3^n \le 3(2^{n-2}3^n)$
$2^{n-2}3^{n+1} \le 2^{n-2}3^{n+1}$

The left hand side of inequality equals the right one. It realizes that the inequality is true. Consequently, we can get the DBNS table representation form of the full *Result* table from the additive operation with any arbitrary integer n where $n \ge 2$. It consumes $\Theta(1)$ time complexity.

IV. Example for 2-Square

The addition of 110 and 43 can be shown as the operation below.

First, we have to overlap two table representations of operands together and we get the table representation of 153 as shown in the Figure 4 below.

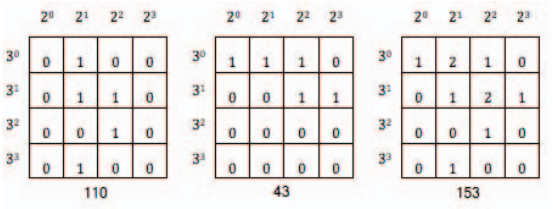

Fig.4 The two operands and their *Sum* table.

Next, we have to divide the *Sum* table to four 2×2 squares as shown in the Figure 5 below.

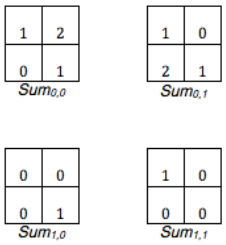

Fig.5 The divided 2×2 squares and their numerical values.

Then, calculate the *Couts* of each 2×2 squares and we get

$Sum_{0,0} = 11$, $Cout_{2_{0,0}} = 0$, $Cout_{1_{0,0}} = 1$
$Sum_{0,1} = 13$, $Cout_{2_{0,1}} = 1$, $Cout_{1_{0,1}} = 0$
$Sum_{1,0} = 6$, $Cout_{2_{1,0}} = 0$, $Cout_{1_{1,0}} = 0$
$Sum_{1,1} = 1$, $Cout_{2_{1,1}} = 0$, $Cout_{1_{1,1}} = 0$.

Take *Couts* of the above 2×2 squares to be the *Cins* of the squares below.

We get the $Result_{k,l}$ from the equation

$$Result_{k,l} = Sum_{k,l} + Cin_{k,l} - Cout_{k,l}$$

$Result_{0,0} = 2$,
$Result_{0,1} = 4$,
$Result_{0,0} = 7$,
$Result_{0,0} = 2$.

At last, we map each value of $Result_{k,l}$ to the 2×2 cell. Then, combine all of 2×2 cell, we will get the full table as the last *Result* as shown in the Figure 6 below.

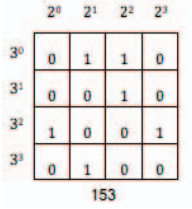

Fig.6 The *Result* table representation of 110 + 43 = 153

The *Result* representation is in form of $153 = 2^13^0 + 2^23^0 + 2^23^1 + 2^03^2 + 2^33^2 + 2^13^3$ which equals to 110 + 43.

V. CONCLUSIONS

In this paper, we propose an addition algorithm for double base number system in parallel manner. Additions of several smaller *n*-square representations are able to be performed at the same time. We demonstrated that addition in DBNS can be realized with a constant-time complexity algorithm. Our approach is also suitable for implementing an ALU. We found that this concept can be used for developing other fundamental arithmetic operations in DBNS in the future.

REFERENCES

[1] A. Avizienis, Signed-digit number representation for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, Volume 10, 1961, pp. 369-400

[2] Christiane Frougny, EditaPelantova, Milena Svobodova, Parallel addition in non-standard numeration systems, Theoretical Computer Science 412 (2011) 5714-5727

[3] Surarerks A., Digit Set Conversion by On-line Finite Automata, *Bulletin of The Belgian Mathematical Society Simon Stevin*, Volume 8, 2001, pp. 337-358

[4] V.S. Dimitrov, G.A. Jullien, W.C. Miller, Theory and Applications for Double-Base number System, IEEE Transactions on Computers, Volume 46, pp. 44-51.

[5] Guillaume Gilbert and J. M. Pierre Langlois, Multipath Greedy Algorithm for Canonical Representation of Numbers in the Double Base Number System, IEEE Transactions on Computers, Volume 49, 2005.

[6] Ming Li Kunpeng Wang, New Greedy Algorithm for Double-Base Chains, 2009 Fifth International Conference on Information Assurance and Security, 2009, pp. 445-450.