改良型矩陣乘法器之設計

Design of an Improved Matrix Multiplier

鄭夢涵
國立暨南國際大學資工系
s3321525@ncnu.edu.tw

杜迪榕
國立暨南國際大學資工系
drduh@ncnu.edu.tw

楊蘭超
國立暨南國際大學資工系
s96321528@ncnu.edu.tw

摘要

矩陣乘法是科學與工程計算中常見的運算之一，許多人皆為了能增進其計算效率而努力。近幾十年以來，為了加速這類需要龐大計算量的運算，平行處理不外乎為最佳的選擇。隨著硬體製造技術的進步，選擇高速的處理器或是採用多個處理器來執行這類型的運算也非常普遍。在此篇論文中，合併運算將被包含在平行架構中進行。合併運算打破個別的乘法器與加法器的界線，而將乘法與加法視為一體同時執行。然而，在做個別乘積項的加法時，並沒有任一個方法總是最好的。因此，我們提出一個包含之前的方法和新的混合方式來尋找最有效率的一種。有鑒於使用者對系統的考量並不唯一，我們的模擬程式將輸出三種量測標準供使用者選擇，分別是時間，成本和時間乘以成本。除此之外，大致的硬體連接方式也被呈現於結果中，協助之後的實作設計。對於追求高效能以及低成本的系統設計中，此研究的成果應能提供莫大的幫助。

關鍵詞：矩陣乘法，合併運算，定點運算，縮小部分乘積矩陣

Abstract

Since matrix multiplication is one of the most used operations in science and engineering, a lot of efforts for improving its efficiency have been made greatly. To accelerate such enormous computing, parallel processing architectures are mostly considered by decades. For the advance of manufacturing technology, high clock rate processors or multiple processors are also used to speed up the computation. In this work, another approach called merged arithmetic is included into our parallel architecture. It dissolves the boundary between the individual multipliers and adders to perform multiple multiply and addition in parallel. However, none of the methods, which were presented previous for reducing partial product matrix, is absolutely better than others. This study proposes a combined method to find out the most efficient reduction. Respecting the user’s demand is not the same all the time; our simulation results include three metrics, delay, cost, and delay × cost. Moreover, the hardware interconnection for further implementation is also offered. It is very helpful for the design of such systems because a high performance throughput and low cost system are both what we concern.

Keywords：Matrix multiplication, merged arithmetic, fixed-point arithmetic, partial product matrix reduction

1 Introduction

Matrix multiplication is widely used for solving numerical problems in many areas, such as signal processing, image processing, robotics, and computer graphics, etc [16]. It requires enormous computing power and long execution time during computation so that many researchers have tried to improve the performance of matrix multiplication [2]. It is very inefficient if the computation is done by software on the core central processing unit and the hardware implementation of such a processor is desirable [15]. For the advance of manufacturing technology, high clock rate processors or multiple processors are also used to speed up the computation. Hence, the hardware implementation of the matrix multiplier is an important issue on parallel processing.

It is known that the inner product is the basis of the matrix multiplication. Swartzlander introduced another concept called merged arithmetic for inner product function [9], [20]. It is introduced to reduce the implementation cost and improve the processing speed. In [17], when very high performance is needed, it is sometimes desirable, or even necessary, to build hardware structures to compute the function of interest directly without breaking it down into conventional operations. In [4], the complexity of merged two's complement multiplier-adders is analyzed. It also reveals that merged arithmetic is suitable for portable and low-power designs such as wireless communications.

If the matrix multiplication is performed in parallel, the communication network often becomes a bottleneck. Many high performance algorithms and architectures have been proposed to accelerate matrix multiplication [11], [12], [15], [18], [23]. Recently, field-programmable gate arrays (FPGAs) also become an attractive option for matrix multiplication [2], [14]. Matrix multiplication can be done most efficiently by an FPGA which is shown in [19]. Besides, the bit-level matrix multiplier has been proposed by Grover et al. [11]. In the bit-level design, the bits of a word do not have to be processed as a unit and the bits of individual weight columns can be computed simutaneously. Thus, it is faster than the word-level design.

The merged arithmetic dissolves the boundaries between discrete arithmetic elements and treats them as a whole computing block. The scalar (inner) product macrocell which exploits merged arithmetic has been proposed with substantial improvement in the deep submicron area [10]. The multiply accumulate units (MAC units) utilizing merging technique to enhance tree architecture for further speed improvement has been proposed by Fayed et al. [6]. A variation of merged arithmetic is applied to the implementation of the wavelet transform [3]. The use of merged arithmetic in the FIR filters and IIR lattice filters are investigated in [7], [8], [22]. In [22], a hybrid CSA tree is proposed to shorten the width of Carry Propagation Adder (CPA) to reduce the CPA time. In high performance design, however, the Carry Lookahead Adder (CLA) is usually used to achieve high speed instead of CPA. The hybrid CSA tree following by CLA is discussed and compared to other approaches later. The new combined approaches which are used to reduce the partial product matrix are also presented in this study.

The rest of this paper is organized as follows. The previous results are discussed in Section 2. Section 3 gives the result of our simulation. Finally, the conclusions and future works are addressed in Section 4.

2 Former Results

This section first considers the question of how the multiplication would be fast computed, because the multiplication dominates the whole computation time. Second, the concept of the merged arithmetic is introduced carefully.

2.1 Fast Multiplier

The multiplication of two numbers may be divided into three major steps and the detailed accounts of each step are explained in the following.

1. The partial products generation.
2. To sum up all partial products until only two vectors remain which is often called the reduction of partial product matrix.
3. The addition of the remaining two vectors.

Fig. 1(a) shows multiplying two unsigned 4-bit binary numbers. In the first step, a 2-input AND gate is used to generate one partial product bit. One input is derived from one bit of the multiplicand, and another input is derived from one bit of the multiplier. Sixteen partial product bits in this example are generated in the corresponding weighted columns and form four partial products. After generating the partial product matrix, the partial products are continually summed up by any summation approaches. The height of the partial product matrix is decreased until two vectors left in the course of summation. Then, the final result is computed from adding these two vectors. For simple representation, the multiplication of two unsigned 4-bit binary numbers is also depicted by the dot notation, where one dot represents one bit as shown in Fig. 1(b). The dot in the partial product matrix indicates the output of the 2-input AND gates in partial product generation. In the rest of this paper, the dot notation is used frequently.

To think over the three steps of multiplication, the implementation of the first step is the simplest one and is hard to make a change for acceleration. In order to get a high speed multiplier, the second step and the final step are both the major courses what we stress on in the subsequent discussion.

For speeding up the second step of multiplication, the Wallace tree or the Dadda tree is usually choused to reduce partial products to two vectors in a fast manner [5], [21]. The idea of both is that the carry propagation delay is eliminated during partial product reduction by strategically using carry-save adders. A carry-save adder (CSA) does not propagate the carry-out to a higher adder. It receives three multi-bit inputs and outputs two multi-bit outputs without waiting the carry-out from a lower adder. The Wallace tree and Dadda tree are the tree-structured multipliers which have the better performance than other structured-multipliers. The speed of the multipliers such as array-structured multipliers and iterative-structured multipliers are both mightily influenced by the length of the input numbers. In short, tree-structured multipliers are the emphases of this study and they provide the basic mechanism for merged arithmetic. Please see [17] for details of other multipliers.

2.1.1 Wallace Tree

By using the Wallace tree for fast reduction, there are three reduction steps in each reduction stage that are listed below. The reduction is finished while two vectors are left in the partial product matrix.

1. At each weighted column, the maximum full-adders (FAs) will be used to reduce three bits to two output bits which are the sum bit in the same column and the carry-bit to the next most significant column.

2. If there are two bits left after step 1, the half-adder (HA) will be used to reduce two bits to one sum bit and one carry-out bit similarly.

3. If there is single bit left after step 1, the single bit will be used in the next stage.

The main idea of the Wallace tree is to reduce partial product as soon as possible by using a large of FAs or HAs. Taking a close look to Fig. 2(a), the FAs and HAs are used as possible as they are in every weighted column. It reveals that the Wallace tree minimizes the stage of reduction but maximizes the cost used in reduction process.

2.1.2 Dadda Tree

Unlike the Wallace tree, however, the Dadda tree does as few reductions as possible (see Fig. 2(b)). The reduction rule of Dadda tree is determined by Table 1 which is constructed by a recurrence formula given below. In Table 1, the maximum number of operands means that the number of dots in the highest weighted column and one stage is equal to a FA delay or a HA delay. The number of operands $n(h)$ can be defined as

$$n(h) = \lfloor 3n(h-1)/2 \rfloor, \text{ where } h \in N \text{ and } n(0)=2 \quad (1)$$

By using the Dadda tree for fast reduction, there are three steps in each reduction stage that are listed below. The reduction is finished while two vectors are left in the partial product matrix.

1. To find the maximum operands $n_{\max}$ among all weighted columns.
2. To get the largest $n(h)$ from Table 1 which is less than the $n_{\max}$.
3. For each weighted column, to reduce its operands to $n(h)$ by using the smallest FAs and perhaps one HA is needed to accomplish it.

In each column, the number of stages needed is given by Table 1. Taking Fig. 2(b) as an example, the maximum number of operands in the highest column (middle column) has four operands, so the first stage is to reduce the number of operands to the next lower $n(h)$ value (i.e., 3). In the following stages, the two rows of dots are obtained by applying the same approach.

According to Table 1, reducing 10 operands and 13 operands has the same reduction stages (five stages). The number of FAs and HAs in every stage depend on the recurrence formula severely. From this point of view, the Dadda tree would save the unnecessary cost better than the Wallace tree. Nonetheless, the reduction stages of the Dadda tree may not be equal to the ones of the Wallace tree all the time and may have one extra stage in some cases.

In general, the Wallace tree optimizes speed, whereas the Dadda tree gives less area. As shown in Fig. 2, the total stages required for Wallace tree and Dadda tree are the same, but the cost of Wallace tree is greater than Dadda tree. Four FAs and six HAs are used in Fig. 2(a), and three FAs and three HAs are used in the Fig. 2(b).

2.1.3 Fast Adder

Finally, carry-look-ahead adder (CLA) will be used to sum up the two vectors after above reduction. The CLA will predict the former carry out in advance by the computation result from input. This is why the CLA outperforms than CPA in the high performance design. Please see [17] for detailed introduction.

2.2 2’s-complement Multiplication

So far, we have seen how the unsigned multiplication can be computed rapidly. This section discusses the 2’s complement multiplication. In [17], when one is multiplying 2’s-complement numbers directly, each of the partial products to be added is a signed number. Thus, for the CSA tree to yield the correct sum of its inputs, each partial product must be sign-extended to the width of the final product. It reveals that sign-extend 2’s complement multiplication will lead redundant cost in the hardware.

Baugh and Wooley have proposed a more efficient approach, called modified Baugh-Wooley method, for 2’s complement multiplication [1]. In order to understand this approach, Fig. 3 illustrates this method. Fig. 3(a) shows the multiplication of two

2’s complement numbers. Because of the negative weight of the sign bit in a 2’s complement number, some entries are depicted with ‘–’ signs. To avoid summing up these negative weight bits, a transfer formula is given as $-z = -(1-\overline{z}) = \overline{z} - 1, z \in \{0, 1\}$.

After applying this formula to the negative weight bits in Fig. 3(a), each negative weight bit will be replaced with a 1’s complement positive bit and negative one. All of the negative ones will be simplified to a coefficient which is depicted at the bottom of Fig. 3(b). It is clear that the modified Baugh-Wooley method never increase the column height and the time required in reduction process isn’t increased at all. On the whole, this approach is more efficient and less cost than sign-extended 2’s complement multiplication.

2.3 Merged Arithmetic

In [9] and [20], merged arithmetic has been proposed to do fast computation in inner product function. Merged arithmetic is faster than conventional design because it dissolves the boundaries between the multiplication and addition by computing these two functions at once. The Fig. 4(a) shows the conventional design of a two-term 4-bit inner product. Two multipliers and one adder are needed in this conventional structure.

Recall that the second step of the multiplication mentioned in Subection 2.1 is to sum up all partial products until two vectors left. For considering the products of multipliers are added eventually, the addition of these two products can be computed earlier (in the second step of the multiplication). Fig. 4(b) shows the merged arithmetic of a two-term 4-bit inner product. All partial product matrices from each product is summed up altogether by using fast tree reductions, where the fast compression of Dadda’s method is proposed by [9] due to its optimal circuit. The fast compression is illustrated in Fig. 5. In this example, 8 bits can be reduced into 4 bits (instead of 6) since there is no carry-in from the least significant column of bits. The inner product result is the same obtained by Fig. 4(a) and Fig. 4(b), but the latter leads one carry-propagate saving which is at the bottom of the reduction stages of the conventional multiplier and also minimizes the number of gate counts.

In [9], an alternative to performing above fully merged arithmetic is to keep the individual multipliers separate by performing the column reduction only within the individual multipliers until the partial product matrix is reduced to two equivalent rows. Then, a multi-input adder is used for summing up the two row matrix from each multiplier altogether instead of a conventional two-input adder as shown in

Fig. 6. This alternative approach is called partially merged arithmetic and its advantage is in much more regular structure. Nevertheless, the hardware reduction and speed gains are less than fully merged arithmetic. The result in [9] shows us that for all vector sizes less than 16, and word lengths between 2 and 32 bits, at most two additional full adder delays result from not using the fully merged arithmetic.

In above paragraphs, unsigned multiplications are discussed. In [4], the complexity of merged two’s complement multiplier-adder has been proposed. The efficient modified Baugh-Wooley method is used in the composite bit product matrix. Fig. 7 illustrates an example of a 2’s complement two-term inner product. The black dots are identical to those in unsigned multiplications, and the white dot indicates the output of a two-input NAND gate in the partial product generation. The 1’s indicate the logical value ONE’s which is the result of combining two coefficients at the bottom of each 2' complement multiplier (see Fig. 3(b)).

Choe and Swartzlander figured out that merged arithmetic reduces hardware complexity proportional to the number of inner-product terms [4]. Moreover, it improves slightly more for smaller word sizes. In particular, it is suitable for low-power designs such as wireless communication and digital-camera applications.

A hybrid CSA tree for merged arithmetic architecture of FIR Filter has been proposed in [22]. The Dadda strategy uses less hardware cost but may result in a longer CPA and more reduction stages than Wallace. On the other hand, the Wallace strategy uses more hardware cost and shorter CPA. The hybrid tree reduction scheme is a combination of the Wallace strategy and the Dadda strategy. The merged operation block in FIR filters is portioned into two sub-blocks. The Wallace and Dadda strategy is applied on each sub-block independently. As shown in Fig. 8(a), the Wallace strategy is applied to the right sub-block of the arrow, the Dadda strategy is applied to the left sub-block of the arrow. For the right sub-block, the purpose is to reduce each column to a single-bit output. This leads a shorter CPA which equals the result if only the Wallace strategy is applied (see Fig. 8(b)) and with a small amount of hardware in excess to that required by the Dadda tree structure. Thus, the partition line, i.e. the boundary the arrow points, must be known before the computation. The partition line is found by the output of the Wallace tree for the given block in advance. Taking the Fig. 8(b) for explanation, the output of the Wallace tree has four consecutive single-bit columns at the least significant columns so that the partition line is decided at the left of these columns.

However, in some cases, the Dadda structure yields one extra-stage than the Wallace structure. The combination of both structures of the left sub-block is preferred in order to maintain the same latency as the Wallace structure.

2.4 Fixed-Point Number System

This work focuses on the fixed-point number system. In [13], fixed-point arithmetic is usually used when hardware cost and speed is limited. Specialized DSP systems typically use fixed-point number representation for lower cost and greater speed. For basic signal processing computations such as digital filters and FFTs can be implemented in fixed-point representation with good performance. However, finite-precision quantization issue must be noticed carefully because fixed-point system offers the limited range and/or precision representation.

An n-bit fixed-point number k can be partitioned into three parts, one sign bit, p-bit integer part and $(n–p–1)$-bit fractional part. The sign bit is set to zero when k is positive, and is set to one when k is negative. The value for $k = \{k_{n-1}k_{n-2}\ldots k_0\}$ is expressed as

$$-2^p k_{n-1} + \sum_{i=0}^{n-2} 2^{i-n+p+1}. \qquad (2)$$

When multiplying two fixed-point n-bit numbers, the computational result would be $2n$-bit width which should be still n-bit number in some fixed-point system actuality. The least significant $n–p–1$ bits and the most significant $p+1$ bits should be cancelled out because the $2n$-bit result is out of the range that can be properly represented in the given data size. Thus, the overflow error and the truncation error will be inevitable. The overflow error must be carefully handled since it will make the result incorrect. The suitable truncation approach must be made up for the errors through truncating.

3 Our Result

In [9] and [20], merged arithmetic has been proposed to speed up the inner product with lower gate counts and reduction stages. In [22], a hybrid CSA tree has been proposed to shorten the width of the final CPA. Notably, their results are only compared to conventional designs. It is helpless for users who want to design such systems with different considerations. This work tries to find the most suitable merging approach according to the user's desired demand. Moreover, the hardware interconnection is also outputted for further implementation. Finally, this work also shows how to construct the improved matrix multiplier by utilizing such merged inner product.

In this section, the word length is denoted by *N* and the vector size is denoted by *M*. Our simulation inputs take *N* from 2 to 64 and *M* from 1 to 16. The outputs are the results of *N*-bit *M*-term merged inner product. The results of the simulation include the delay, cost, delay × cost and the hardware interconnection.

3.1 The Estimation Method

In this work, three metrics are used for our simulation result. The time delay, gate count cost and delay × cost are adopted in general. To estimate these three metrics, a widely accepted approach is described in the following. This method takes any monotonic gate (e.g. AND, NOR, etc.) has one gate delay and cost excluding the XOR gate which has two gate delays and costs. Any multi-input gate is transformed to a series of multiple 2-input monotonic gates for evaluation. For example, a four-input OR is transformed into using three 2-input OR gates. The delays and costs of some arithmetic blocks are given in Table 2 which will be used latter. In our estimation method, CLA is constructed with 2-bit lookahead blocks where *n* denotes the input data width of CLA and the estimation is penalizing by using two input gates.

3.2 Simulation Results of the Previous Approaches

In [9], the Dadda tree with fast reduction is applied for the fully merged arithmetic. According to our simulation results, the values of the Dadda tree with fast reduction are chiefly the same as those of the Dadda tree without fast reduction excepting some cases. Take a more look on these different cases, the delay and cost of the Dadda tree with fast reduction has saved at most one FA delay and 12 costs than without fast reduction respectively. In brief, the Dadda tree with fast reduction isn't worst than without fast reduction for the delay and cost metrics.

This work also compares the delay and cost between the Wallace tree and the Dadda tree with fast reduction in our simulation. The result shows that the delay of the Wallace tree is smaller than or equal to the Dadda tree with fast reduction in most cases and the former saves one FA delay or one HA delay than the latter. There are still few cases (i.e. 11 cases) where the delay of the Wallace tree is greater than the Dadda tree with fast reduction and the former spends one HA delay than the latter. However, the cost of the Dadda tree with fast reduction is smaller than or equal to the Wallace tree in most cases and the difference between them is increasing as the *N* or *M* increases approximately. For example, the difference in cost is 168 when $N = 16$ and $M = 2$ and is 472 when $N = 32$ and $M = 2$. There are few cases where the cost of the Dadda tree with fast reduction is greater than the Wallace tree, but the difference is small (i.e. the largest difference is 19). On the whole, the simulation results reveal the similar situation which has been discussed in Subsection 2.1.

In [22], the hybrid CSA strategy is applied for reduction process followed by a CPA. In high performance design, however, the Carry Lookahead Adder (CLA) is usually used to achieve high speed instead of CPA. It has not shown us that if a CLA is applied for final addition after reducing the partial product matrix to two vectors instead of a single-bit output at the least significant columns. This inspires us to do more simulation for comparing. The combination of the Wallace reduction stages and the Dadda reduction stages is also presented to make the total reduction stages as the Wallace-structured in [22]. Therefore, more explicit combinations are simulated to realize its affect. For the user's demands are not the same all the time, applying fixed strategy in all designs is inadvisable. Hence, more work must be taken subsequently.

3.3 The Reduction Methods on Demand

As mentioned in Section 2, the former reduction approaches include the Dadda strategy with and without fast reduction, the Wallace strategy, and the hybrid CSA tree. Based on the strategies mentioned above, we attempt to do some simulation which has not been done yet. The stages of the Wallace tree and the Dadda tree with fast reduction are combined with all possible combinations. This combined strategy is to apply the Wallace strategy first and the Dadda strategy with fast reduction later. One of the extreme cases is the pure Wallace tree and the other is the pure Dadda tree with fast reduction. The combined strategy is also applied to understand the affect of the hybrid CSA tree. This will be explained carefully later. After reducing partial product matrix to two vectors, a CLA with 2-bit lookahead generator is used for the final addition.

One of the reduction methods is denoted by $D(\omega$, F.D.), where ω is the number of reduction stages by using the Wallace tree only and F.D. means the Dadda tree with fast reduction. In the reduction with $D(\omega$, F.D.), first apply the Wallace strategy for ω stages and then apply the Dadda strategy with fast reduction until two vectors left. When ω is zero, the combined method is a pure Dadda tree with fast reduction. On the other hand, the combined method is a pure Wallace tree if ω equals the number of stages needed in the Wallace-structured reduction. The number of stages needed in the Wallace-structured reduction is denoted by $\omega_{\max}$ in this paper.

Another combined method is denoted by $H(\omega$, F.D.), where ω and F.D. are the same meaning with $D(\omega$, F.D.) but are only applied on the left part of the arrow (see Subsection 2.3). In the reduction with $H(\omega$, F.D.), first apply the Wallace strategy for ω stages and apply the Dadda strategy with fast reduction on the left part of the arrow. When ω is zero, the right part of the arrow is a pure Wallace tree and the left part of the arrow is a pure Dadda tree with fast reduction. The reduction tree is a traditional Wallace tree with single-bit output at the least significant columns if ω equals to $\omega_{\max}$.

All reduction approaches for the merged arithmetic architecture are listed in the following.

1. Applying the Wallace strategy to reduce partial product matrix to two vectors left. ($D(\omega_{\max,}$ F.D.))
2. Applying the Dadda strategy with fast reduction to reduce partial product matrix to two vectors left. (D(0, F.D.))
3. Applying the combined method $D(\omega$, F.D.) to reduce partial product matrix to two vectors left.
4. Applying the Wallace strategy to reduce partial product matrix to single-bit output at least significant columns. ($H(\omega_{\max}$, F.D.))
5. Applying the Wallace strategy to reduce partial product matrix in the right part of the arrow (in [22]) and applying the Dadda strategy with fast reduction in the left part. (H(0, F.D.))

6. Applying the Wallace strategy to the right part of the arrow (in [22]) and applying the third approach to the left part of the arrow. ($H(\omega$, F.D.))

After applying any of the above methods, the CLA with 2-bit lookahead generator is used for final two vectors addition. The items listed above reveal that these two reduction methods including the former approaches which are discussed in Section 2 and the new combined approaches which have not been tried yet. The affect of the hybrid CSA tree can be understood by comparing $D(\omega$, F.D.) and $H(\omega$, F.D.).

Fig. 9 shows the flow of our simulation program. First, N, M and D (user's demand) are assigned. Recall that the $\omega_{\max}$ must be obtained before performing the functions F1 and F2. Two $\omega_{\max}$s outputted from the functions Wallace1 and Wallace2 have the unequal meaning. The function Wallace1 results two vectors (carry and sum vectors) but the function Wallace2 results single vector at the least significant bits instead of two vectors. By the way, the combined methods can be well computed for a reasonable ω. Next, we can perform the combined methods and save the results. Finally, all saved results from function F1 and F2 are compared.

For the user's demand is changeable, we offer the best result depending on the user's demand. When the N and M are given, we will output the best approach according to the designate demand. It's helpful for the users who want to design such systems because the simulation result obtained in Subsection 3.2 shows that none of them is always better than others and some of them are not done. Our results are shown in the next section.

3.4 Our Simulation

About three metrics in our simulation, the delay demand is denoted by 1, the cost demand is denoted by 2 and the delay × cost demand is denoted by 3. For N=8, M=4, the H(1, F.D.) is the best approach on delay × cost demand as illustrated in Fig. 10. If the delay demand is selected, the H(1, F.D.) will be outputted at the bottom line. The H(0, F.D.) will not be outputted because its cost is not the lowest among all $H(\omega$, F.D.) and $D(\omega$, F.D.) with the same delay. Identically, the H(1, F.D.) will be outputted at bottom line if the cost demand is selected. The delay values in Fig. 10 exclude the delay of generating partial product matrix and so do the cost values. Those values can be ignored because both of them are fixed values as N and M unchangeable and are necessarily included in any combined approach.

The hardware interconnection of H(1, F.D.) is also shown in Fig. 11. Each line behalves a reduction stage of the H(1, F.D.), the order of each row is the stage order in reduction process. The figure fingers out that it requires eight stages to finish the reduction. The delay of each stage is a FA delay or a HA delay relying on the usage. The right columns of each row are the least significant columns. In each pair parentheses (x, y) at i-th row, x means the number of FAs which are used in stage i and y means the number of HAs which are used in stage i. The (0, 0) indicates that neither FAs nor HAs are used for the reduction. The hardware interconnections of CLAs are not shown in Fig. 11.

This work also makes efforts to realize the optimization of the proposed combined methods. Furthermore, we want to know that the best method for delay demand is equivalent to the best method for cost demand or not. As the demand is delay, the outputted delay and cost are denoted by MIN_DELAY and COST respectively. As the demand is cost, the outputted delay and cost are denoted by DELAY and MIN_COST respectively. One of the simulation results is on the viewpoint of

DELAY minus MIN_DELAY and another is on COST minus MIN_COST. In most cases of both viewpoints, the difference value is zero. It means that the combined method outputted by our program results in the most optimal method. Nevertheless, there are few cases that the difference value is non-zero. It is impossible to get the most optimal method because the delay demand and cost demand are the trade-off problem in such cases. Minimizing one of them will maximize the other. Besides, the non-zero values are in the range of the Wallace tree and the Dadda tree with fast reduction.

3.5 Improved Matrix Multiplier

The proposed matrix multiplier is based on the merged arithmetic approach mentioned in Subsection 3.3. The traditional multipliers and accumulators are dissolved and merged into a block to perform inner-product operation. Because the matrix multiplication has a large of inner-product operations indeed, the matrix multiplier can be improved by utilizing suitable merging approach. In brief, the improved matrix multiplier bases on the acceleration of the merged inner product function. One example of designing such kind of the matrix multiplier is described below.

The architecture of the improved matrix multiplier (as shown in Fig. 12) is similar to the architecture proposed by Huang and Duh [12]. The structure of an array processor includes a control unit (CU), a control unit memory (CUM), processor elements (PEs), processor element memories (PEMs), and the communication network. CU is dedicated to process program. It fetches instructions from CUM and passes data to PEMs for PEs. The data transfer between PEMs is through the communication network. However, each PE in our design has merged M multiplications and M–1 additions into one merged computing block and has PEMs for storing operands and results. In addition, the width of each bus is M×N bits. The circuit of the computing block is got from the simulation result of Subsection 3.3 according to the given demand. Data on each X bus or Y bus can be simultaneously shard by M processor elements; and every processor among these M processors can be outgoing data. The data on the X_i bus, a_{i0}, a_{i1}, a_{i2}, a_{i3},…, $a_{i(M-1)}$ can be simultaneously used by processors $PE_{i,0}$,…, $PE_{i,M-1}$; and the data on the Y_j bus, b_{0j}, b_{1j}, b_{2j}, b_{3j},…, $b_{(M-1)j}$, can be simultaneously used by processors $PE_{0,j}$,…, $PE_{M-1,j}$. Therefore, multiplying two M×M matrices exists no time delay for data transfer.

4 Concluding Remarks

The matrix multiplier is widely used in many scientific computations. Due to the enormous power and time spend in matrix multiplication, many efforts are done to achieve high performance. In general, a parallel processing architecture is often adopted for speeding up. Besides the parallel computing on matrix multiplier, merged arithmetic has been proposed to speed up inner product with lower cost and delay. In the bit-level design, individual bits of a word can be parallel processed. Thus, it is desirable when very high performance is needed.

This work concentrates on the inner product operations because the basic operations of matrix multiplication comprise a large of inner products. The partial product reduction of the merged inner product is performed by using fast tree strategies such as the Wallace tree, the Dadda tree with and without fast reduction and the CSA hybrid tree. In this work, a combined strategy is proposed to find the best reduction approach according to user’s demand. This combined method includes the former methods and some combined methods which have not been done before. Moreover, when *N* and *M* are given (the word length is denoted by *N* and the vector size is denoted by *M*), the time delay, hardware cost and multiplication of both are given in the simulation result. Besides three metrics are outputted, the corresponding hardware interconnection is also outputted for further implementation.

In this study, fixed-point matrix multiplier is discussed. For considering the high precision and large range of data in some applications, floating-point representation is more suitable than fixed-point representation. But the hardware resources and the execution time used to perform floating-point operations are greater than fixed-point operations very much. The design of efficient floating-point hardware is a difficulty challenge. This leads the floating-point hardware in an expensive choice. How to accelerate floating-point matrix multiplier by applying merged arithmetic is what we concern in the future.

Some rough ideas are presented in the following. For accelerating the floating-point multiplication, the merged arithmetic can be applied to the multiplication of two significands. For addition of multiple floating-point numbers, all of the numbers can be merged to one time addition. For dealing with the floating-point inner product, take two-term inner product as an example. The exponential computation includes two additions and one subtraction that can be merged to perform at once. The individual product of each term may not be available in fact and the CSA tree can be applied to the partial product matrix to reduce it to two rows left. This leads one time carry-propagate saving. Finally, sum up these four rows altogether under the correct shift of each exponent.

References

[1] C. R. Baugh, and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Transaction on Computers*, vol. C-22, pp. 1045-1047, 1973.

[2] F. Bensaali, A. Amira, and A. Bouridane, "Accelerating matrix product on reconfigurable hardware for image processing applications," *IEE Proceedings of Circuits, Devices and Systems*, vol. 152, no. 3, pp. 236-246, 2005.

[3] G. Choe and E. E. Swartzlander, Jr., "Merged Arithmetic for computing wavelet transforms", in *Proceedings of the 8th Great Lakes Symposium on VLSI*, 1998, pp. 196-201.

[4] G. Choe and E. E. Swartzlander, Jr., "Complexity of merged two's complement multiplier-adders," in *Proceedings of the 35th IEEE Midwest Symposium on Circuits and Systems*, 1999, vol. 1, pp. 384-387.

[5] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, pp. 349-356, 1965.

[6] A. Fayed, W. Elgharbawy and M. Bayoumi, "A Data Merging Technique High-Speed Low-Power Multiply Accumulate Units, " in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 2004, pp. V- 145-8.

[7] K. A. Feiste and E. E. Swartzlander, Jr., "High-speed VLSI implementation of FIR lattice filters," in *Proceedings of the 29th Asilomar Conference on Signals, Systems and Computers*, 1995, pp. 127-131.

[8] K. A. Feiste and E. E. Swartzlander, Jr., "High-speed VLSI implementation of IIR lattice filters," in *Proceedings of the 30th Asilomar Conference on Signals, Systems and Computers*, 1996, pp. 1057-1062.

[9] K. A. Feiste and E. E. Swartzlander, Jr., "Merged arithmetic revisited," in *Proceedings of the IEEE Workshop on Signal Processing Systems*, 1997, pp. 212-221.

[10] J. Gu, C.-H. Chang and K.-S. Yeo, "Algorithm and Architecture for a High Density, Low Power Scalar Product Macrocell," *IEE Proceedings on Computer Digital Technology*, vol. 151, no. 2, pp. 161-172, 2004.

[11] R. S. Grover, W. Shang, and Q. Li, "Bit-level two's complement matrix multiplication," *Integration, the VLSI Journal*, vol. 33, no. 1, pp. 3-21, 2002.

[12] H.-P. Huang and D.-R. Duh, "Fast computation algorithm for robot dynamics and its implementation," in *Proceedings of the IEEE International Symposium on Industrial Electronics*, 1992, pp. 352-356.

[13] D. L. Jones, *Fixed-Point Number Representation*, Connexions Web site. http://cnx.org/content/

m11930/1.2/, Dec 28, 2004.

[14] J.-W. Jang, S. B. Choi, and V. K. Prasanna, "Energy- and time-efficient matrix multiplication on FPGAs," *IEEE Transaction on Very Large Scale Integration Systems*, vol. 13, no. 11, pp. 1305-1319, November 2005.

[15] R. Lin, "A reconfigurable low-power high-performance matrix multiplier design," in *Proceedings of the IEEE First International Symposium on Quality Electronic Design*, 2000, pp. 321-328.

[16] E. L. Leiss, *Parallel and Vector Computing*, McGraw-Hill, New York, 1995.

[17] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford Univ. Press, New York, 2000.

[18] V. Y. Pan, "How can we speed-up matrix multiplication?" *SIAM Review*, vol. 26, no. 3, pp.393-415, 1984.

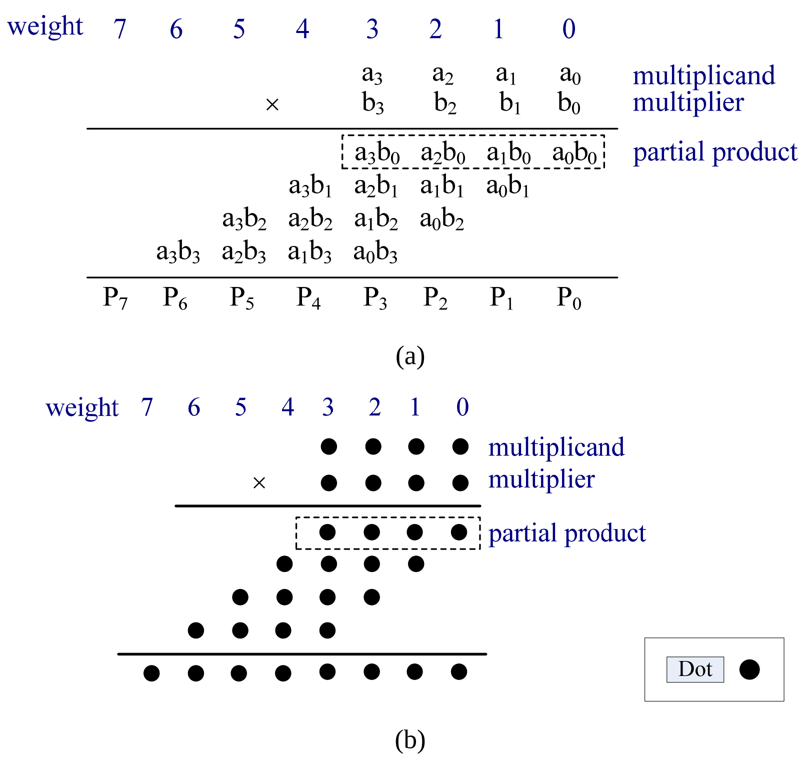
[19] R. Scrofano, S. Choi and V. K. Prasanna, "Energy Efficiency of FPGAs and Programmable Processors for Matrix Multiplication", in *Proceedings of the IEEE International Conference on Field-Programmable Technology*, 2002, pp. 422-425.

[20] E. E. Swartzlander, Jr., "Merged arithmetic," *IEEE Transaction on Computers*, vol. C-29, no. 10, pp. 946-950, October 1980.

[21] C. S. Wallace, "A suggestion for a fast multiplier", *IEEE Transaction on Electronic Computing*, vol. EC-13, pp. 14-17, 1964.

[22] Z. Ye and C.-H. Chang, "A hybrid CSA tree for merged arithmetic architecture of FIR filter," in *Proceedings of the 3rd International Symposium on Image and Signal Processing and Analysis*, 2003, pp. 449-453.

[23] L. Zhuo and V. K. Prasanna, "High performance linear algebra operations on reconfigurable systems," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, 2005.



weight 7 6 5 4 3 2 1 0
a_3 a_2 a_1 a_0
× b_3 b_2 b_1 b_0
multiplicand
multiplier
a_3b_0 a_2b_0 a_1b_0 a_0b_0
partial product
a_3b_1 a_2b_1 a_1b_1 a_0b_1
a_3b_2 a_2b_2 a_1b_2 a_0b_2
a_3b_3 a_2b_3 a_1b_3 a_0b_3
P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0
(a)
weight 7 6 5 4 3 2 1 0
multiplicand
multiplier
×
partial product
Dot
(b)

Fig. 1. Unsigned 4×4 multiplication (a) in binary number and (b) in dot notation.

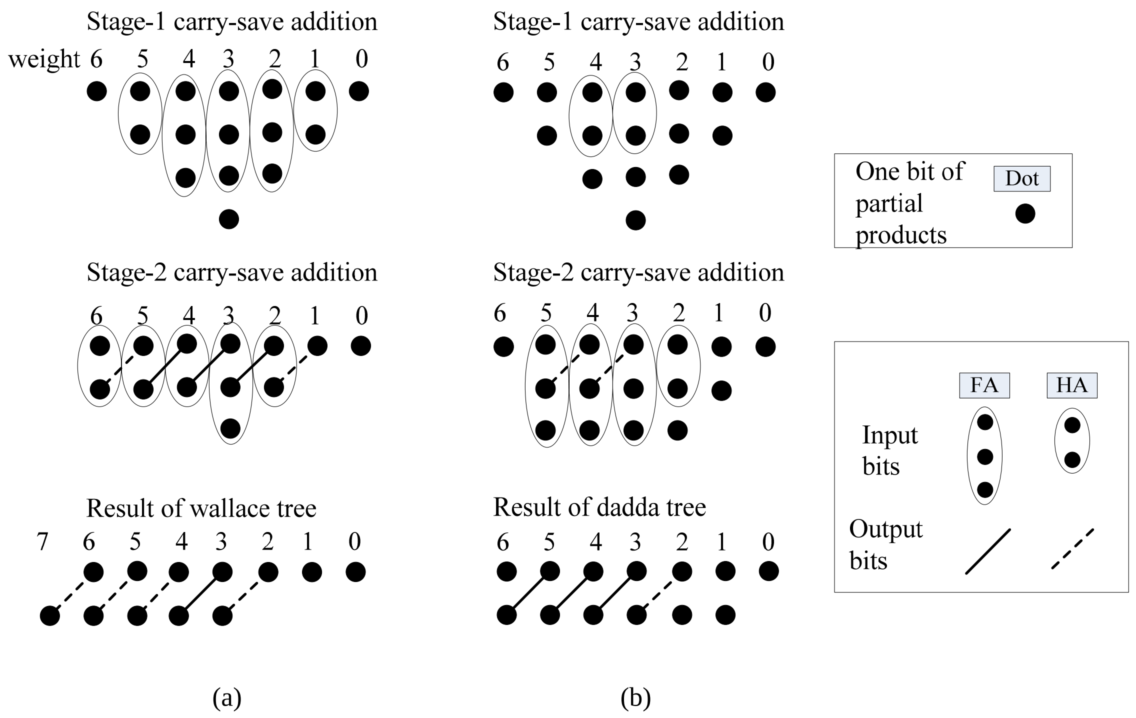


Fig. 2. The reduction trees. (a) The Wallace tree. (b) The Dadda tree.

Table 1 The Maximum Number of Operands $n(h)$ for an h-Level Carry-Save-Adder Tree

Number of operands ($n(h)$)	Number of stages (h)
2	0
3	1
4	2
6	3
9	4
13	5
19	6
28	7
42	8
63	9
94	10

weight	7	6	5	4	3	2	1	0
					a_3	a_2	a_1	a_0
				$\times$	b_3	b_2	b_1	b_0
					$-a_3b_0$	a_2b_0	a_1b_0	a_0b_0
				$-a_3b_1$	a_2b_1	a_1b_1	a_0b_1	
			$-a_3b_2$	a_2b_2	a_1b_2	a_0b_2		
		a_3b_3	$-a_2b_3$	$-a_1b_3$	$-a_0b_3$			
	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

(a)

weight	7	6	5	4	3	2	1	0
					a_3	a_2	a_1	a_0
				$\times$	b_3	b_2	b_1	b_0
					$\overline{a_3b_0}$	a_2b_0	a_1b_0	a_0b_0
				$\overline{a_3b_1}$	a_2b_1	a_1b_1	a_0b_1	
			$\overline{a_3b_2}$	a_2b_2	a_1b_2	a_0b_2		
		a_3b_3	$\overline{a_2b_3}$	$\overline{a_1b_3}$	$\overline{a_0b_3}$			
	1			1				
	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

(b)

Fig. 3. 2's complement multiplier. (a) 2's Complement Multiplier. (b) Modified Baugh-Wooley Method.

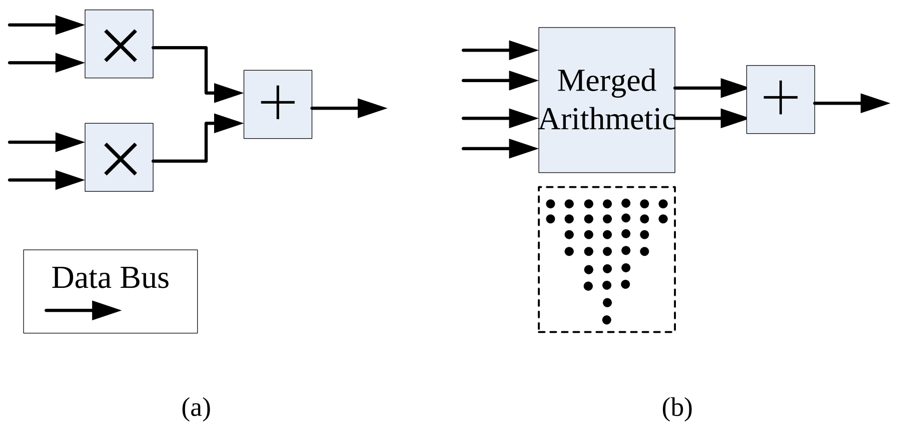


Fig. 4. Merged arithmetic. (a) Traditional inner product. (b) Merged inner product.

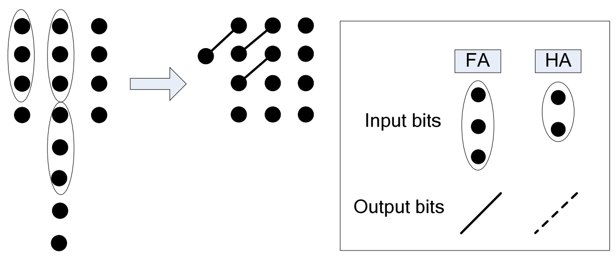


Fig. 5. The Dadda tree with fast reduction.

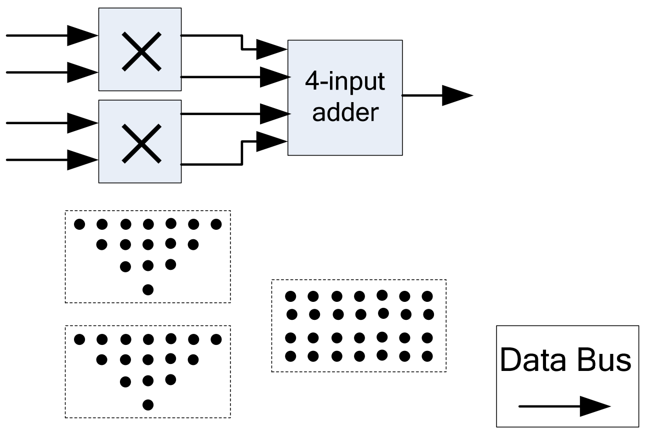


Fig. 6. Partially merged arithmetic.

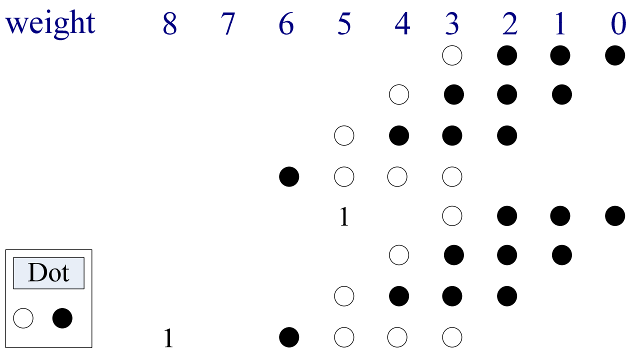


Fig. 7. The composite bit product matrix for merged 2's complement two-term inner product.

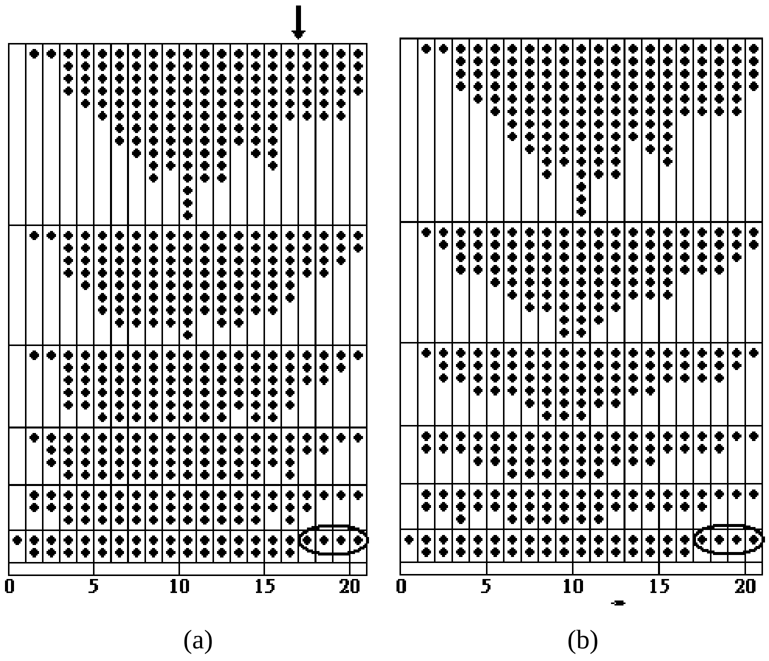
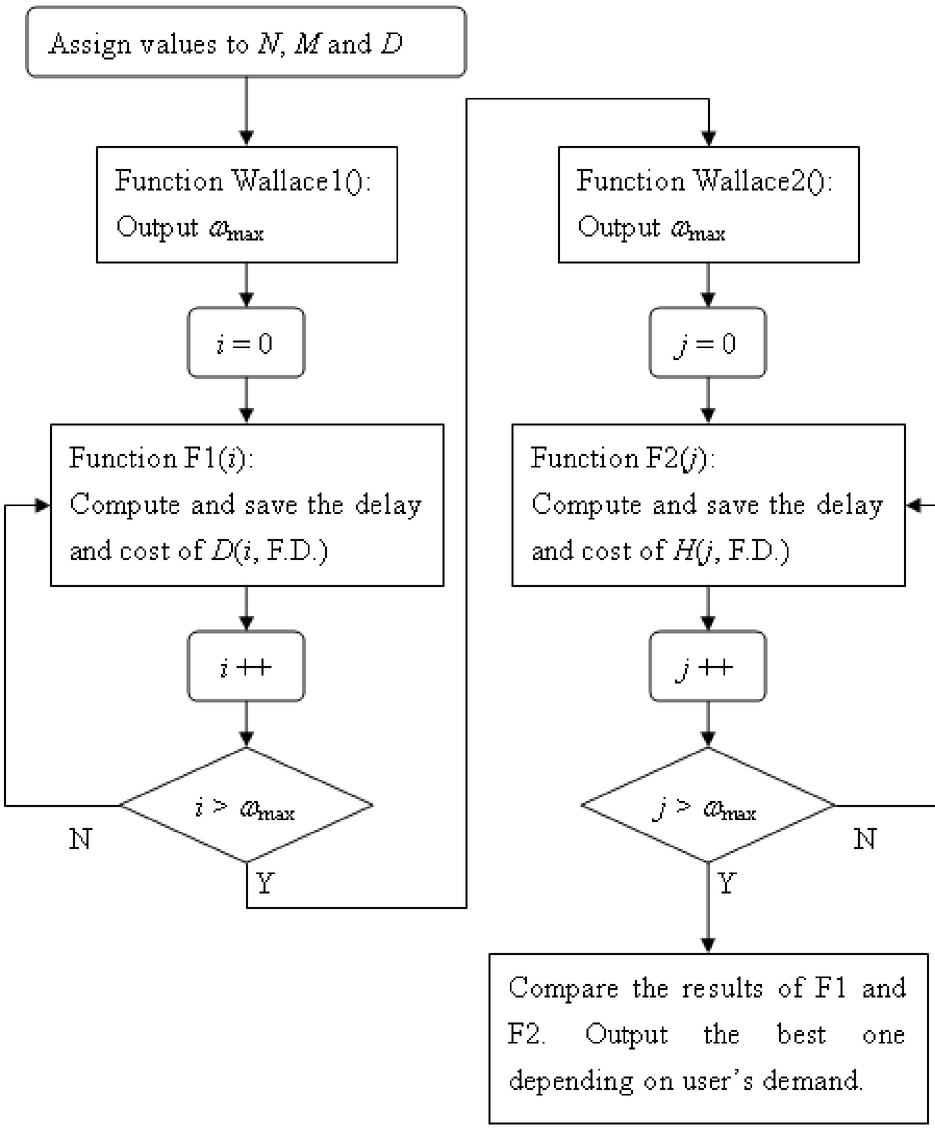


Fig. 8. CSA tree reductions in [22]. (a) CSA tree reduction with hybrid structure. (b) CSA tree reduction applying Wallace's strategy.

Table 2 Delay and Cost of Some Arithmetic Blocks

	Delay	Cost
Half-adder (HA)	2	3
Full-adder (FA)	4	7
CLA with 2-bit lookahead carry generator	$1+2\times(\lceil\log_2 n\rceil-1)+2$ $+2\times(\lceil\log_2 n\rceil-1)+4$	$2\times 2^{\lceil\log_2 n\rceil}+3\times(\lceil\log_2 n\rceil-1)+6$ $+2\times(\lceil\log_2 n\rceil-1)+4\times 2^{\lceil\log_2 n\rceil}$



Assign values to N, M and D
Function Wallace1(): Output a_{max}
$i = 0$
Function F1(i): Compute and save the delay and cost of D(i, F.D.)
i++
$i > a_{max}$
N
Y
Function Wallace2(): Output a_{max}
$j = 0$
Function F2(j): Compute and save the delay and cost of H(j, F.D.)
j++
$j > a_{max}$
N
Y
Compare the results of F1 and F2. Output the best one depending on user's demand.

Fig. 9. The flow of our simulation program.

```
D(0, F.D.):delay=55,cost=1815          H(0, F.D.):delay=51,cost=1801
D(1, F.D.):delay=51,cost=1823          H(1, F.D.):delay=51,cost=1798
D(2, F.D.):delay=51,cost=1820          H(2, F.D.):delay=51,cost=1798
D(3, F.D.):delay=53,cost=1835          H(3, F.D.):delay=51,cost=1798
D(4, F.D.):delay=53,cost=1844          H(4, F.D.):delay=51,cost=1804
D(5, F.D.):delay=53,cost=1861          H(5, F.D.):delay=51,cost=1818
D(6, F.D.):delay=53,cost=1866          H(6, F.D.):delay=51,cost=1826
D(7, F.D.):delay=53,cost=1874          H(7, F.D.):delay=51,cost=1834
D(8, F.D.):delay=53,cost=1874          H(8, F.D.):delay=51,cost=1834
=>D(2, F.D.): 92820 on demand 3        =>H(1, F.D.): 91698 on demand 3
```

Fig. 10. Simulation result for *N*=8 and *M*=4.

Weight	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Stage1	(0,0)	(0,0)	(0,0)	(1,0)	(2,1)	(4,0)	(5,0)	(7,0)	(8,0)	(9,0)	(10,1)	(9,0)	(8,0)	(6,1)	(5,0)	(4,0)	(2,1)	(1,0)
Stage2	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,1)	(3,0)	(3,1)	(6,0)	(5,0)	(4,0)	(3,0)	(2,0)	(1,0)	(0,1)
Stage3	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(2,1)	(4,1)	(6,0)	(5,1)	(5,0)	(3,1)	(3,0)	(2,0)	(2,0)	(1,0)	(1,0)	(0,0)
Stage4	(0,0)	(0,0)	(0,0)	(0,0)	(1,0)	(3,1)	(4,0)	(4,0)	(4,0)	(3,1)	(3,0)	(2,0)	(1,1)	(1,1)	(1,0)	(1,0)	(0,0)	(0,0)
Stage5	(0,0)	(0,0)	(0,0)	(1,1)	(3,0)	(3,0)	(3,0)	(3,0)	(3,0)	(3,0)	(2,1)	(1,1)	(1,0)	(1,0)	(0,1)	(0,0)	(0,0)	(0,0)
Stage6	(0,0)	(0,0)	(0,1)	(2,0)	(2,0)	(2,0)	(2,0)	(2,0)	(2,0)	(2,0)	(1,1)	(1,0)	(1,0)	(0,1)	(0,0)	(0,0)	(0,0)	(0,0)
Stage7	(0,0)	(0,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(0,1)	(0,1)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
Stage8	(0,0)	(0,1)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(1,0)	(0,1)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)

Fig. 11. The hardware interconnection of *H*(1, F.D.) for *N*=8 and *M*=4.

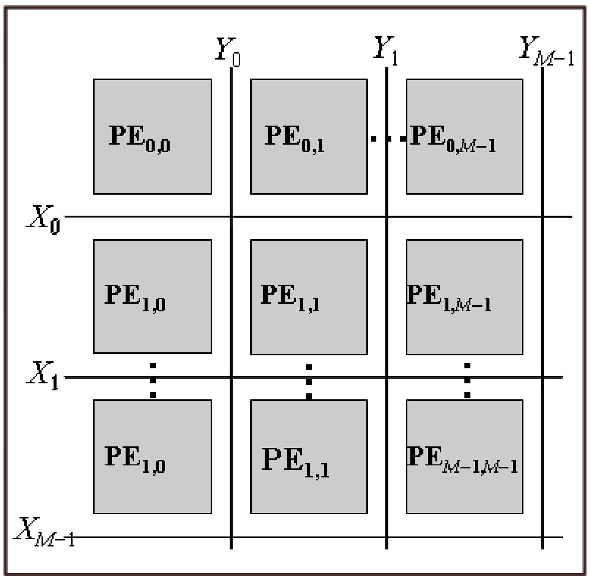


Fig. 12. The architecture of the proposed matrix multiplier.