Systolic Polynomial Evaluation and Matrix Multiplication with Multiple Precision

Jonathan Schaeffer
Darrell Makarenko

VLSI Research Group,
Department of Computing Science,
University of Alberta,
Edmonton, Alberta,
Canada T6G 2H1

ABSTRACT

The design and implementation of a systolic VLSI multi-precision polynomial evaluator and matrix multiplier is described. The use of bit-serial arithmetic allows for a very simple cell design (two registers and an accumulator) enabling a substantial number of cells to be placed on a chip. A configuration of N^2 cells can evaluate N polynomials of N coefficients at N points and perform N-width band matrix multiplication and $N \times N$ full matrix multiplication, each in linear time. Using current technology, 100 polynomials of 100 coefficients can be evaluated at 100 data points with 32 bit precision in an estimated one millisecond.

1. Introduction

Systolic algorithms for VLSI abound in the literature. Most are presented from a theoretical point of view rather than with view towards an actual implementation. Systolic matrix multiplication algorithms, such as Kung's [1] and Weiser and Davis' [2], are regular and elegant but do not address the practical considerations necessary for a VLSI implementation. These considerations include cell complexity, chip area, and pin-out. In a VLSI systolic design, maximum benefits can be achieved only if one can place a substantial number of cells on a chip.

This paper presents the design and implementation of a systolic VLSI polynomial evaluator and matrix multiplier. A configuration of N^2 cells can evaluate N polynomials of N coefficients at N data points and perform N-width band matrix multiplication and $N \times N$ full matrix multiplication, each in linear time. Previous work has addressed the band matrix multiplication problem [1,2] and the related inner-product problem [3] without generalizing the solution to include polynomial evaluation.

The cell used is simple in design and easy to implement (essentially two registers and an accumulator) allowing many cells to be placed on a single chip. The simplicity of the design is a result of the bit-serial arithmetic and transmission of data. For a computation such as $a = b \times c$, the use of bit-serial arithmetic allows arbitrary precision at no cost along the a and c data paths. Arbitrary precision can be achieved along the b data path either linearly in time or space.

There are 2 contributions made by this work. Firstly, a systolic design for polynomial evaluation is presented. The

design uses a simple, bit-serial cell that is easily implemented. Secondly, the same cell can be used to support arbitrary precision matrix multiplication.

A prototype chip has been successfully fabricated. At an estimated clock speed of 200 nanoseconds, 100 polynomials of 100 coefficients at 100 data points can be evaluated to a precision of 32 bits in 1.28 milliseconds.

2. Bit-serial Computations

Polynomial evaluation and matrix multiplication can be viewed as a series of multiplications and additions. Having a full multiplier and accumulator per cell is expensive in chip area and would constrain the calculations to a fixed precision. Thus an alternate means for performing the multiply and accumulate operations was used, bit-serial arithmetic. Bit-serial multipliers are not new, having been studied extensively in the literature [4,5]. Using a bit-serial approach, a simple multiply and accumulate cell can be designed.

A multiplier that computes the product $s = a \times b$, where a, b, and s are n bits in length, is straight-forward to construct. If b is latched in an n-bit register, then a can be streamed bit-serially through the cell, each time adding the value of that bit times b to the accumulator. If the bits of a are entered in the order least to most significant (lsb to msb), then the accumulator need only hold n bits. Since an accumulation allows the lsb in the accumulator to take on its final value, the accumulator can be shifted right each time allowing the answer to stream bit-serially out of the accumulator.

To allow successive values of a and s to be fed through the cell, we require that they be the same length, that being the maximum possible size that a, b, or s can take. After the last bit of s for one computation shifts out of the cell, new values of a and b can enter starting the next computation. A stream of multiplier values, a, passing through the cell will be referred to as the A data path. Similarly, the stream of multiplicands, b, will be called the B data path and the outputs, s, the S data path.

As all the bits of b are required for the multiplication, they must all be present before the first bit of a arrives. This is not quite true, as section 5 discusses means for breaking b up into smaller pieces. For the time being, it is assumed that each cell has a latch large enough to accommodate all of b. The number of bits, w, in this latch is referred to as the *cellwidth*. b is sent bit-serially, starting w clock cycles in advance of when it is needed, shifted in, and latched at the appropriate time. The transmission of the new b value is done concurrently with the arithmetic being performed on the current b value.

Consider using two of these multipliers to compute

$$s = a_1 \times b_1 + a_2 \times b_2.$$

The product $a_1 \times b_1$ streams bit-serially from the first multiplier ($M1$) and must be added to the output $a_2 \times b_2$ of the second ($M2$). The accumulation of the product can be pipelined if $M2$ is synchronized so that it begins only as soon as the first bit of output from $M1$ is available. When $M2$ receives the lsb of $a_1 \times b_1$ from $M1$, it can then compute the lsb of $a_2 \times b_2$ and add them, giving the lsb of s. This is then repeated for each of the successive bits of s. This can be implemented by feeding the answer coming from $M1$ directly into the low-order carry bit of $M2$'s accumulator (Figure 1). A row of cells inter-connected as described will solve recurrences of the form

$$s_{i+1} = s_i + a_i \times b_i \quad (R1).$$

A recurrence relation very similar to $R1$ is

$$s_{i+1} = a_i + s_i \times b_i \quad (R2).$$

Essentially, both calculations are identical except for the

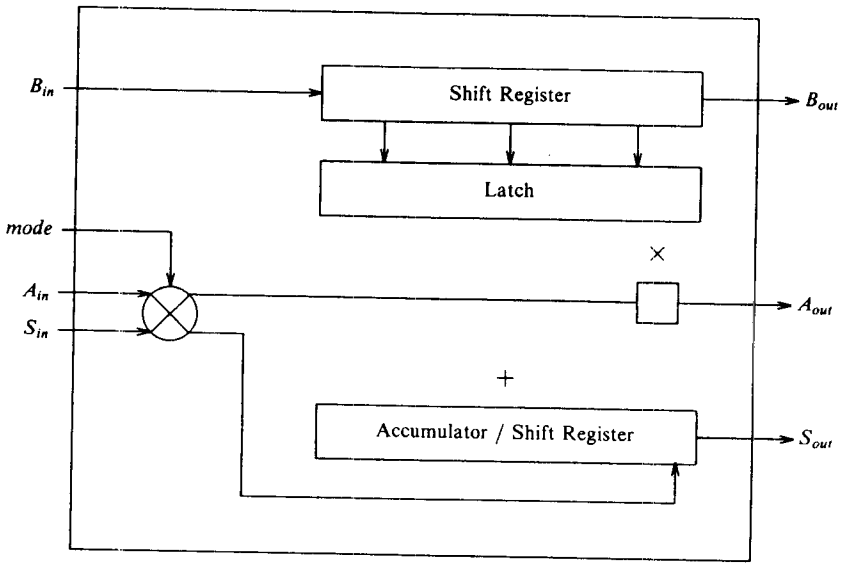

Figure 1. Cell Block Diagram

reversal of roles of s_i and a_i. A *mode* switch is implemented which, under program control, can reverse the roles of S_{in} and A_{in} to allow either $R1$ or $R2$ to be computed. Looking ahead, setting *mode = matrix* ($R1$) allows matrix computations to be performed and setting *mode = poly* ($R2$) allows polynomials to be evaluated.

3. Systolic Polynomial Evaluation

An arrangement of interconnected cells can be used to evaluate a set of K polynominals. Each polynominal has N coefficients C, and is to be evaluated for a set of M different X values.

$$f_j(X_i) = C_{j,1} X_i^{n-1} + C_{j,2} X_i^{n-2} + \cdots + C_{j,n-1} X_i + C_{j,n}$$

$$i = 1,M;\ j = 1,K$$

A matrix of coefficients can be used to represent the polynominals with each row in the matrix representing a separate polynominal. The X values can be represented as a vector.

Figure 2 illustrates the arrangement of cells required to perform polynomial evaluation. Each cell has *mode = poly* to perform the recurance relation R2, as described in the previous section. For clarity, the figure depicts the data moving in parallel along full width data paths whereas in the actual implementation all bits travel serially.

The bits of the coefficients C and of the result stream along the A and S data paths respectively, with P bits per element and one bit per clock cycle. The bits of the X values also travel sequentially along the B data path and contain w bits per element (where w is the *cellwidth* as will be described in section 5). They however, must be sent w clock cycles in advance of their use so as to be latched by each cell when needed. Each column of cells along the B data path all latch simultaneously with the latching occuring P clock cycles apart. The latching of adjacent columns is displaced by one clock cycle. Simple counters can be placed on chip to generate the latching signals required.

This arrangement of cells implements the well known Horner's algorithm for polynominal evaluation.

$$C_{j,1} X^{n-1} + C_{j,2} X^{n-2} + \cdots C_{j,n} =$$

$$(\ldots((C_{j,1} X + C_{j,2}) \times X + C_{j,3}) \times X + \cdots + C_{j,n})$$

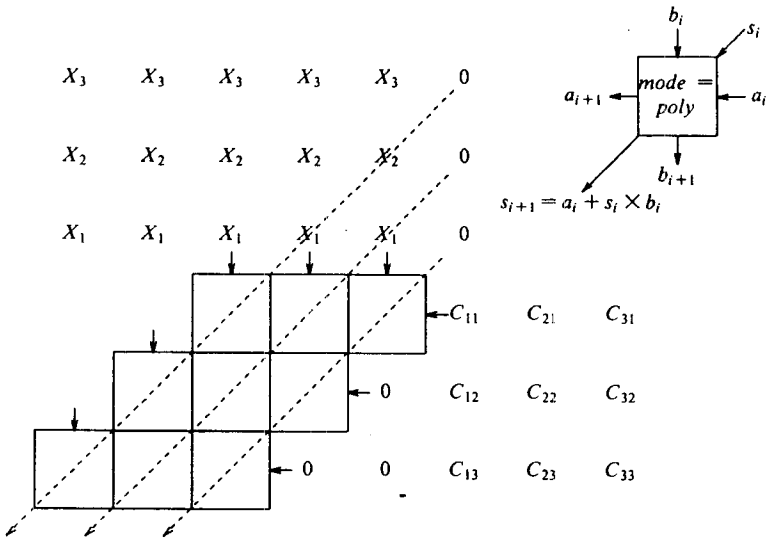

Figure 2. Polynomial Evaluation

On the first clock cycle the top right hand cell in Figure 2 evaluates $C_{1,1} + X_1 \times 0$. In the next cycle the cell diagonally below it evaluates $(C_{1,1} + X_1 \times 0) \times X_1 + C_{1,2}$. This continues until the result $f_1(X_1)$ emerges from the bottom cell of the rightmost column. Thus the results flow diagonally through the cells along the dashed arrows emerging from the bottom.

It is necessary for the X values to be fed in twice in order for all of the polynominals to be evaluated for all values of X. The coefficients are only fed in once. Dummy elements are used to pad the data when the number of polynominals and the number of X values differ.

If there are K polynominals of N coefficients each with M values of X, each of precision P, then the number of cells needed in this arrangement is $N \times Max(K,M)$ and the evaluations can be done in a time of $(2 \times Max(K,M) - 1) + (K \times P)$ clock cycles.

4. Systolic Matrix Multiplication

If $A = (a_{ij})$ and $B = (b_{ij})$ are two $N \times N$ matrices, then their product $S = (s_{ij})$ is given by the equation

$$s_{ij} = \sum_{k=1}^{N} a_{ik}\, b_{kj} \qquad i, j = 1, N$$

This is really a calculation of inner products and can be performed by a 2-dimensional array of cells with *mode = matrix* (*R*1). For band matrix multiplication, the interconnection format is the same as that used in [2] but here the complexity of the individual cells is greatly reduced. This is illustrated in Figure 3 with a band width of three. As in polynominal evaluation, the figure shows the data paths as full width for reasons of clarity. They are in fact implemented using the pipelined bit-serial manner described in section 2.

For $N \times N$ band matrices of band widths W_1 and W_2 and precision P, the multiplication can be done using $W_1 \times W_2$ cells in $P \times N$ clock cycles. Start-up costs were not included in this timing, as they can be reduced by pipelining one computation after another.

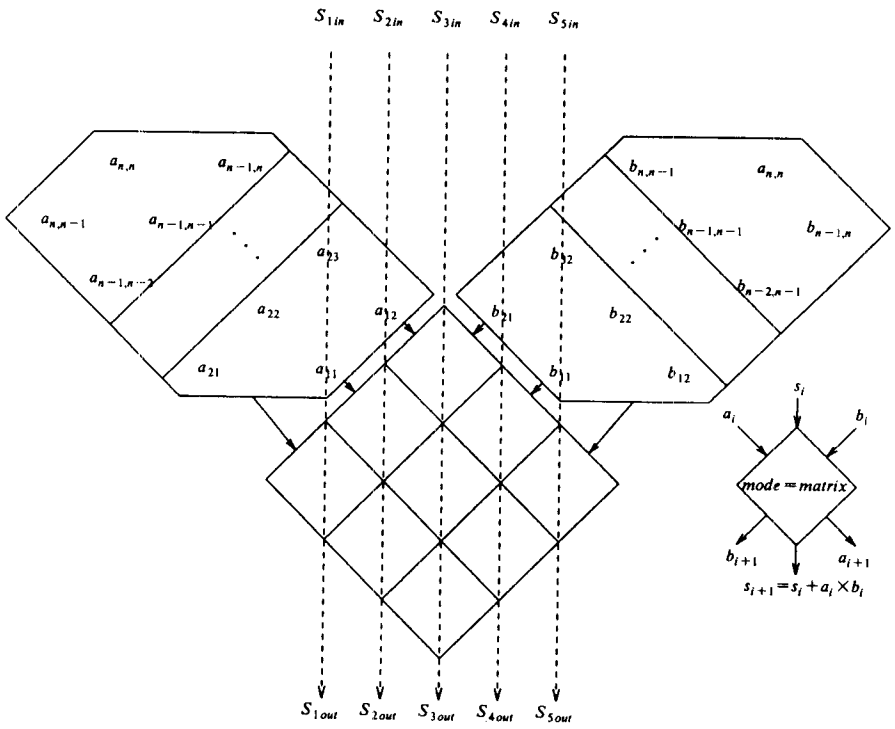

Figure 3. Band Matrix Multiplication

Full matrix multiplication can also be achieved by routing output results back into the top of the cell arrangement. The details of this scheme are described in [6].

As in polynominal evaluation, padding with dummy or zero elements is necessary when non-square matrices are involved. Very large matrices may need to be broken into smaller matrices and the results mathematically reassembled if the configuration of cells is not large enough.

5. Arbitrary Precision

In section 2 it was described how a and s were pipelined bit-serially through an array of cells. There were no constraints placed on the length of these bit streams. However, the precision of b was governed by the size of the cell latch. Since only a fixed, finite number of bits can be stored per cell, it is necessary to break b up into smaller bit streams to allow arbitrary precision. We store w bits of b per cell and arrange for cells to abut together to accommodate greater precision. This parameter w was referred to in section 2 as the *cellwidth*.

Assume b requires P bits of precision, $P > w$. Then b can be broken into $\left\lceil \frac{P}{w} \right\rceil$ groups of w consecutive bits. The low-order w bits of b can be multiplied by a and added to s. Systolic arrays of cells can be cascaded together to achieve arbitrary precision as illustrated in Figure 4. There are $\left\lceil \frac{P}{w} \right\rceil$ arrays of cells; each using w bits of b. When s comes bit-serially out of the first set of cells, the low order w bits are part of the final answer. They can therefore be extracted at this point and the remaining bits allowed to propagate into the second set of cells as their initial S_{in}. These cells then perform the calculations using the next significant group of w bits of b. Actually, to allow proper accumulation of signed numbers (see below), each set of cells should use only $w-1$

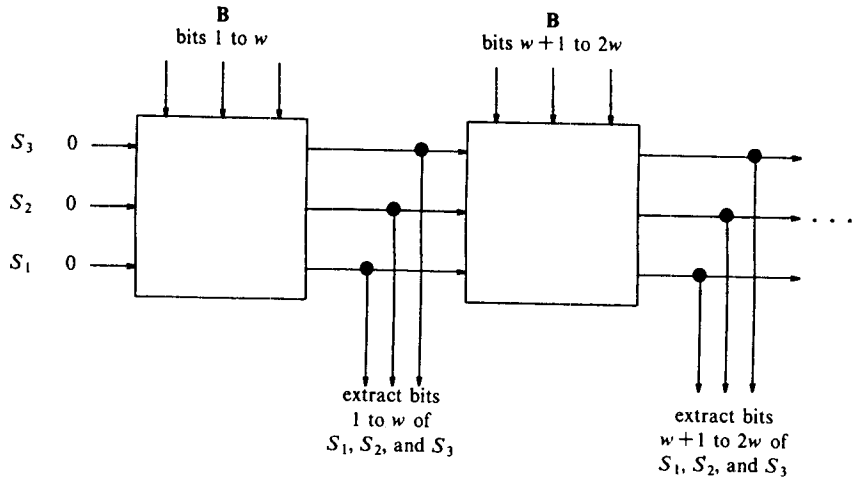

Figure 4. Cascading Chips Together

bits of b with bit w being set to the proper sign of b.

Arbitrary precision for b can be distributed over time instead of space. The data can be fed through one set of cells repeatedly, each time varying the input bits of b, extracting w bits of s, saving the high-order bits, and then feeding them back into the array. However, additional extra storage is required for holding intermediate results.

The algorithms described to this point work only for positive integers. Two minor modifications to the cell are required to accommodate two's complement arithmetic. The first change is to enhanced the accumulator to contain $w+2$ bits. During the addition step the high-order (sign) bit of b is added to bits w, $w+1$, and $w+2$ of the accumulator allowing proper sign-extension. The carry out signal of the high order bit of the accumulator can be ignored. During the shift step the high order bit $(w+2)$ of the accumulator remains unchanged (sign extension).

After a multiplication involving negative numbers is complete and the next one about to begin, there may be non-zero bits left in the accumulator. The second modification to the cell requires that when b is latched, the accumulator be reset to zeros. The implementation details of both modifications are left for the next section.

6. Implementation

Figure 5 shows a block diagram of a single cell. A two-phase clock controls the operation of the chip. During the first phase $\phi 1$, the cells perform the additions and during $\phi 2$, they shift right. The latching of the bits of B in the shift register into the multiplicand register is signaled by the latch pulse $b-latch$. A column of cells always latches at the same time, 1 clock cycle after its left neighbor. The additional shift register necessary to have the $B-latch$ signal shift from column to column is not shown.

In laying out a systolic array of cells on a chip, pin-out becomes the major constraint. There are 7 pins required for overhead in our design; V_{dd}, GND, $\phi 1$, $\phi 2$, $b-latch$, $b-latchout$ and $mode$. Multiplexing of any of the remaining pins was not considered to avoid adversely affecting the data rate and to minimize the extra control circuitry required.

To maximize the number of cells that can be placed on a chip, the cells are arranged in a hexagonal grid. A hexagon with M cells per side will contain $3M(M-1)+1$ cells. $4M-2$ pins are required for each of A, B, and S, giving a total pin

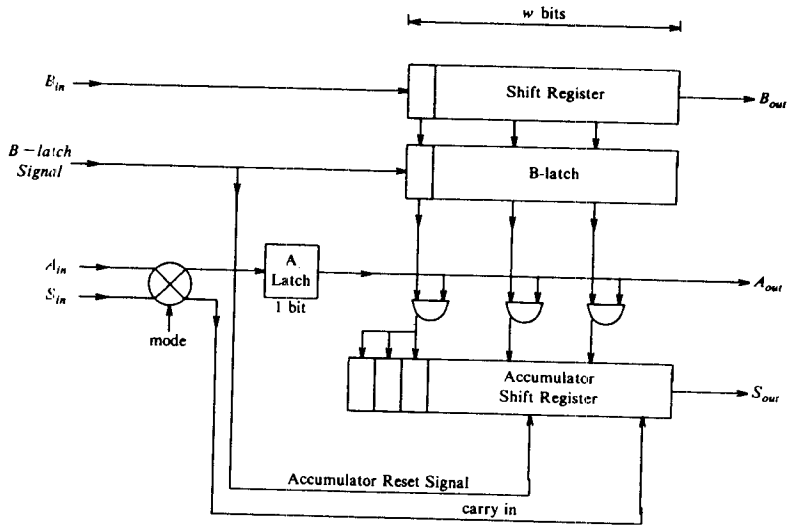

Figure 5. Cell Layout

requirement of $12M-6$. Thus 19 cells can be accommodated in a 40 pin package, 61 in a 64 pin package, and 169 in a 100 pin package. As N the number of pins available increases the number of cells on a chip can increase by $O(N^2)$. In implementing such arrays of cells the *cellwidth* would be chosen so as to utilize all of the area of the chip. This would have no effect on the number of pins needed.

Chip-to-chip communication is the constraining factor on the operating speed of the chip. This is typically less than 200 nanoseconds per minor clock cycle for current technologies. If large arrays of cells are used, clock skew must also be taken into account.

As an example, if 100 polynomials of 100 coefficients are to be evaluated at 100 data points with 32 bits of precision and using 40 pin chips (19 cells of 32 bit *cellwidth* per chip), then 400 chips would be needed and the entire matrix multiply would take 1.28 milliseconds ($32 \times 100 \times 2 \times 200 nanoseconds$). Using 100 pin chips the number of chips reduces to 49. If the problem is band matrices instead, then significantly less chips would be needed (depending on the width of the band) but the time would be the same.

As in any systolic array of chips, synchronization of the input and output data is important. With a large number of chips the combined bandwidth of all of the data paths may exceed the memory bandwidth of most machines. We envision the data being fed in and retrieved from the chip array using a set of high speed sequencing ram chips placed around the perimeter of the array. These ram chips would be loaded prior to the actual calculations.

7. Conclusions

This paper has presented the design and implementation of a polynomial evaluator and matrix multiplier chip. Two of the important features of the design are the simplicity of the cell and the arbitrary precision of the calculations. Both of these are the direct result of the bit-serial approach to both the calculations and the transmission of data. Many of the published systolic algorithms have been designed using parallel data paths. Arguments given against the use of bit-

serial transmission of data in systolic structures have been based on the assumption that individual cell operations are performed on full words. Once a bit-serial cell design is chosen, bit-serial transmission of data becomes the natural choice. This simplifies the cell logic, eliminates pin-out problems, minimizes area loss due to routing, and maximizes the flexibility of the design. With these major obstacles overcome, the actual implementation is considerably simplified. The cell design presented in this paper is easy to build and small enough to accommodate a large number on a single chip. It is these two attributes which are the essential ingredients of any implementable systolic algorithm.

8. Acknowledgements

The authors would like to thank Dr. Cliff Addison for his valuable comments and suggestions. The support of Dr. Tony Marsland and Dr. John Tartar is also appreciated. Special thanks to Mui-Hua Lim for her assistance and patience.

9. References

(1) H.T. Kung, Lets Design Algorithms for VLSI Systems, *Proceedings of the Caltech Conference on Very Large Scale Integration*, January, 1979, Charles Seitz (ed.), 65-90.

(2) U. Weiser and A. Davis, A wavefront notation tool for VLSI array design, *CMU Conference on VLSI Systems and Computations*, October 19-21, 1981, Computer Science Press, H.T. Kung, Bob Sproull, Guy Steel (ed.), 226-234.

(3) M.R. Buric and C.A. Mead, Bit Serial Inner Product Processors in VLSI, *Proceedings of the Second Caltech Conference on Very Large Scale Itegration*, January 19-21, 1981, Charles Seitz (ed.), 155-164.

(4) I.N. Chen and R. Willoner, An O(n) Parallel Multiplier with Bit-Sequential Input and Output, *Transactions on Computers* **C-28**(10), October, 1979, 721-727.

(5) R. Gnanasekaran, On a Bit-Serial Input and Bit-Serial Output Multiplier, *Transactions on Computers* **C-32**(9), September, 1983, 878-880.

(6) D. Makarenko and J. Schaeffer, A VLSI Multi-Precision Matrix Multiplier and Polynomial Evaluator, TR 84-11, Department of Computing Science, UNiversity of Alberta, November, 1984.