The Application of Systolic Architectures in VLSI Design

1986

Steven V. Manno
University of Central Florida

Find similar works at: http://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Engineering Commons

STARS Citation


This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
THE APPLICATION OF SYSTOLIC ARCHITECTURES IN VLSI DESIGN

BY

STEVEN V. MANNO
B.S.E.E., Pennsylvania State University, 1983

RESEARCH PAPER

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering in the Graduate Studies Program of the College of Engineering University of Central Florida Orlando, Florida

Fall Term
1986
ABSTRACT

Very large scale integrated (VLSI) circuit technology has offered the opportunity to design algorithms and data structures for direct implementation in integrated circuits. In order to take full advantage of this opportunity, the designer must understand the geometric limitations of the technology. Because the interconnections between components in a VLSI circuit are costly in both chip area and performance, the designer must consider the complexity of the data paths between components when selecting algorithms and architectures.

Systolic architectures offer a way to implement massively parallel special-purpose systems in VLSI circuits while minimizing costly interconnections between processing components. In a systolic system, data flows from memory in a rhythmic fashion, passing through many processing elements before it returns to memory.

Some of the architectural issues of VLSI design are presented, and the applications and advantages of systolic architectures in the design of special-purpose hardware using VLSI technology are discussed. A systolic priority queue and a systolic array for band matrix multiplication are presented as application examples.
# TABLE OF CONTENTS

## LIST OF FIGURES

iv

## INTRODUCTION

1

## CHAPTER I, ARCHITECTURAL ISSUES OF VLSI STRUCTURES

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modular and Regular Design</td>
<td>3</td>
</tr>
<tr>
<td>Concurrency</td>
<td>3</td>
</tr>
<tr>
<td>Communication and Memory Locality</td>
<td>4</td>
</tr>
<tr>
<td>The I/O Balance and Pipelining</td>
<td>5</td>
</tr>
</tbody>
</table>

## CHAPTER II, THE SYSTOLIC APPROACH

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systolic System Defined</td>
<td>6</td>
</tr>
<tr>
<td>The Systolic Device</td>
<td>6</td>
</tr>
<tr>
<td>Target Applications</td>
<td>7</td>
</tr>
</tbody>
</table>

## CHAPTER III, APPLICATIONS OF SYSTOLIC STRUCTURES

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Various Configurations of Systolic Structures</td>
<td>9</td>
</tr>
<tr>
<td>Systolic Stacks and Priority Queues</td>
<td>11</td>
</tr>
<tr>
<td>A Systolic Stack</td>
<td>12</td>
</tr>
<tr>
<td>The Stack Becomes a Priority Queue</td>
<td>16</td>
</tr>
<tr>
<td>Band Matrix Multiplication</td>
<td>19</td>
</tr>
<tr>
<td>The Inner Product Step Processor</td>
<td>19</td>
</tr>
<tr>
<td>The Hexagonal Processor Array</td>
<td>21</td>
</tr>
</tbody>
</table>

## CHAPTER IV, ADVANTAGES OF SYSTOLIC STRUCTURES

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduced I/O Bandwidth</td>
<td>23</td>
</tr>
<tr>
<td>Extensive Use of Concurrency</td>
<td>23</td>
</tr>
<tr>
<td>Modular Design</td>
<td>24</td>
</tr>
<tr>
<td>Simple and Regular Interconnections</td>
<td>24</td>
</tr>
</tbody>
</table>

## CHAPTER V, CONCLUSIONS

25

## REFERENCES

26
LIST OF FIGURES

Figure 1. Geometric Configurations of Systolic Structures .......................... 10
Figure 2. The Systolic Stack .................................................. 13
Figure 3. Sample Operation of the Systolic Stack .................................. 15
Figure 4. Sample Operation of the Systolic Queue .................................. 18
Figure 5. Band Matrix Multiplication ........................................... 19
Figure 6. The Inner Product Step Processor ..................................... 20
Figure 7. Hexagonal Array for Band Matrix Multiplication ................ 22
INTRODUCTION

Very large scale integrated (VLSI) circuit technology makes feasible the implementation of high-performance special-purpose computational systems. These special-purpose devices are useful for application specific tasks and for off-loading general-purpose computers of time-consuming computations. Because both processing elements and memory elements can be implemented in VLSI, it is possible to construct highly concurrent systems on a single silicon chip.

Because VLSI is a planar technology, the interconnections between processing elements in a highly concurrent system become very costly. These interconnections occupy large amounts of chip area and cause greatly reduced component speeds. Simple and regular interconnections lead to higher component densities and better performance.

Algorithms and data structures can be designed to take advantage of the VLSI technology while minimizing costly interprocessor communication. Systolic architectures offer a method for optimizing an algorithm for direct layout in integrated circuits. In a systolic system, data flows from the computer memory in a rhythmic fashion, passing through
many processing elements before it returns to memory. Many algorithms may be implemented as systolic arrays. A systolic array consists of a network of processors which communicate with only their neighboring processors.

The processing elements of a systolic array may be arranged in several geometric configurations. For example, a hexagonal array can implement matrix multiplication, and a tree arrangement can support searching algorithms. Systolic arrays are applicable to a wide range of applications such as fast Fourier transform, matrix multiplication, LU decomposition, priority queues, pattern matching, and relational database operations.
CHAPTER I, ARCHITECTURAL ISSUES OF VLSI STRUCTURES

VLSI technology provides the opportunity to implement high-performance devices inexpensively. In order to take full advantage of this opportunity, it is necessary to understand the limitations of the technology.

Modular and Regular Design

The cost of designing VLSI components is much larger than the cost of the components themselves. Design costs become especially significant if the component is only produced in small quantities, as is often the case in special-purpose systems. Great savings can be achieved if a design can be decomposed into a few simple building blocks which are used repetitively (Kung 1982). The reduction of a complex system into more simple blocks is similar to the methods used in software design.

If a system can be decomposed into very few highly regular parts it is likely to be modular. In a modular design the performance of the system can be adjusted simply by adding more modules. The concept of modularity is important in VLSI designs where part costs are highly dependent on chip area. The cost of parts can be made proportional to the required performance.
Concurrency

The way to build a fast computer system is to use fast components. The way to build still faster computer systems is to use concurrency. "With current technology, tens of thousands of gates can be put in a single chip, but no gate is much faster than its TTL counterpart of ten years ago" (Kung 1982). Since component density is increasing much more quickly than component speed, improvements in system performance require the use of concurrent processing.

Communication and Memory Locality

Since VLSI is a planar technology, the interconnections between the many devices on a chip may cost more in chip area and device performance than the devices themselves (Leiserson 1979). Restricting the interconnections between the components on a chip to local communication between neighboring components improves chip density and reduces parasitic interconnect delay. The complete elimination of global or broadcast signals from the design is the ideal situation.

Conventional computational systems suffer from the problem that a processor is separated from its memory. Accessing memory over long communication paths can contribute greatly to processing time. VLSI technology allows processing structures to be placed in close proximity of memory structures (Conway and Mead 1980).
The I/O Balance and Pipelining

A special-purpose system often receives data and returns results by means of a host computer. The special-purpose system is implementing a compute-bound algorithm if the number of computing operations is larger than the number of input and output elements. The I/O bandwidth between the host and the special-purpose hardware may limit the maximum rate of operations in a compute-bound situation if the special-purpose machine needs to access the host's memory for each operation. The required I/O bandwidth in compute-bound problems can be greatly reduced by pipelining. Pipelining allows the special-purpose machine to perform more than one operation per I/O access by storing intermediate results internally.
Systolic architectures have been proposed for the VLSI implementation of special-purpose hardware. The use of systolic architectures allows the VLSI designer to meet the architectural challenges discussed in the previous chapter.

**Systolic System Defined**

A systolic system consists of a collection of interconnected processing elements, each capable of performing some simple operation. Processing elements in a systolic system are typically interconnected into a systolic array or systolic tree. Information in a systolic system flows rhythmically between neighboring processing elements in a pipelined fashion. Communication between a systolic system and the outside world occurs only through the processing elements located at the boundaries of the systolic array or systolic tree.

The name systolic is derived from the word systole. Systole refers to the rhythmic contraction of the heart during which blood is forced onward through the body. The rhythmic pumping of data through a systolic system is analogous to the rhythmic pumping of blood through the body.
The Systolic Device

In a systolic system inputs and results flow into and out of the processing elements located along the boundaries of the processing array or tree. These boundaries are what make up the I/O ports of a systolic device. A systolic device can operate in a pipelined manner with input and output occurring synchronously. Systolic devices are attractive as peripheral processors attached to a host computer or as co-processors directly attached to the central processing unit of a Von Neumann machine (Leiserson 1979).

Target Applications

The basic principle of a systolic system is simple. Replacing a single processing element with an array of processing elements will result in a higher computation throughput without an increase in memory bandwidth (Kung 1982). The goal of the systolic approach is to make exhaustive use of data brought out of memory before it is returned. As the data from memory passes through the systolic system, it is available for use by each of the processing elements it encounters along the way.

Computational tasks can be classified as either compute-bound or I/O-bound. An example of a compute-bound task is matrix-matrix multiplication, because the number of multiplication and addition operations is larger than the
total number of data entries in the two matrices. Matrix addition is an example of an I/O-bound task, because the number of addition operations required is less than the number of data elements in the matrices. It is apparent that any attempt to accelerate an I/O-bound computation will require an increase in memory bandwidth. Acceleration of a compute-bound computation may often be accomplished by the systolic approach. Some specific application examples of the systolic approach are presented in the following chapter.
CHAPTER III, APPLICATIONS OF SYSTOLIC STRUCTURES

A wide variety of special-purpose functions can be implemented as systolic structures. It is the purpose of this chapter to identify some of the possible applications of systolic structures and to illustrate a few examples.

Various Configurations of Systolic Structures

A systolic structure consists of many processing elements connected to their neighboring processing elements in a regular fashion. The geometric configuration of the systolic structure is dependent on the algorithm which it supports. Different applications require different algorithms which in turn require different geometric configurations. Figure 1 illustrates a few typical geometric configurations of systolic structures and Table 1 summarizes which computational functions can be implemented using each of these configurations (Briggs and Hwang 1984).
Figure 1. Geometric Configurations of Systolic Structures.
### TABLE 1

GEOMETRIC CONFIGURATIONS AND ASSOCIATED FUNCTIONS

<table>
<thead>
<tr>
<th>CONFIGURATION</th>
<th>FUNCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear array</td>
<td>convolution, Fourier transform, sort, priority queue</td>
</tr>
<tr>
<td>square array</td>
<td>graph algorithms involving adjacency matrices</td>
</tr>
<tr>
<td>hexagonal array</td>
<td>matrix arithmetic, transitive closure, pattern match, relational database operations</td>
</tr>
<tr>
<td>tree</td>
<td>searching algorithms, parallel function evaluation, recurrence evaluation</td>
</tr>
<tr>
<td>triangular array</td>
<td>inversion of triangular matrix, formal language recognition</td>
</tr>
</tbody>
</table>

**Systolic Stacks and Priority Queues**

Two good examples of a linear systolic array are a stack and a priority queue (Guibas and Liang 1982). First, the design of the systolic stack will be presented. Then, a modification to the stack design will be shown to result in the creation of a systolic priority queue.
A Systolic Stack

The basic function of the stack is to provide memory storage where data elements are inserted and deleted in a first in last out fashion. The primary design goal of the stack is to facilitate insertions and deletions of data elements in a constant amount of time regardless of the depth of the stack.

The stack is a linear array of simple processors, each communicating directly with its left and right neighbors. Each processor has a memory cell which may be either empty or occupied by a data element. Elements are inserted into the leftmost cell and retrieved or deleted from the cell next to the leftmost cell. Figure 2 shows the state of the first six memory cells of the stack when it is ready for an insert or delete operation. The black boxes indicate occupied memory cells containing elements ordered such that the element ready for deletion was the last element inserted into the stack.
In order to support a sequence of insertions and deletions, the stack must be cycled. During a cycle operation, neighboring processors may exchange the elements contained in their memory cells among each other. Elements are exchanged according to the following two rules:

1. If there are ever two consecutive occupied cells followed by an empty cell, then the element in the rightmost of the two occupied cells will move one over to the right into the empty cell.

2. If there are ever two consecutive empty cells followed by an occupied cell, then the element in the occupied cell will move one left into the rightmost of the two empty cells.
The systolic stack will be ready for an insert or delete operation if the previous insert or delete operation is followed by three cycle operations.

Figure 3 shows the operation of the systolic stack as it responds to a sample input sequence of three insertions and two deletions. The numerals indicate the order in which the data elements have been inserted into the stack, and the letters I, D, and C represent insert, delete, and cycle operations respectively. Note that the number of cycles needed to ready the stack is independent of the length of the stack.
Figure 3. Sample Operation of the Systolic Stack.
The Stack Becomes a Priority Queue

The basic function of the priority queue is to provide a way for records to be inserted unordered into a set and to be retrieved from that set ordered. The records are ordered by key such that the first record retrieved has the smallest key.

The queue is constructed in the same manner as the stack with the exception of a few modifications. In the queue, the memory cells store records with a field for a key value rather than data elements as in the stack. The processors of the queue must follow an additional rule during a cycle operation.

During a cycle operation, Records are exchanged according to the following three rules:

1. If two adjacent cells are occupied, the records contained in these cells will be swapped if necessary to arrange the records such that the record with the smallest key will reside in the left cell. This rule always takes priority over the following two rules.

2. If there are ever two consecutive occupied cells followed by an empty cell, then the record in the rightmost of the two occupied cells will move one over to the right into the empty cell.

3. If there are ever two consecutive empty cells followed by an occupied cell, then the record in the occupied cell will move one left into the rightmost of the two empty cells.
Note that the second and third rules are identical to the rules for the stack. The systolic queue will be ready for an insert or delete operation if the previous insert or delete operation is followed by four cycle operations.

Figure 4 shows the operation of the systolic priority queue as it responds to a sample input sequence. The numerals indicate the key value of the records in the queue, and the letters I, D, and C represent insert, delete, and cycle operations respectively. As with the systolic stack, the number of cycles needed to ready the queue is independent of the length of the queue.
Figure 4. Sample Operation of the Systolic Queue.
Band Matrix Multiplication

A hexagonal systolic array of processors can be used to implement band matrix multiplication (Conway and Mead 1980). In a band matrix, all of the nonzero entries form a diagonal band of some width. Figure 5 shows an example of the multiplication of two band matrices $A$ and $B$. The result of the multiplication is the band matrix $C$.

\[
\begin{bmatrix}
a_{11} & a_{12} & 0 \\
a_{21} & a_{22} & a_{23} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
0 & \ddots & \ddots & \ddots \\
\end{bmatrix}
\begin{bmatrix}
b_{11} & b_{12} & b_{13} & 0 \\
b_{21} & b_{22} & b_{23} & b_{24} \\
b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\
0 & \ddots & \ddots & \ddots & \ddots \\
\end{bmatrix}
= 
\begin{bmatrix}
c_{11} & c_{12} & c_{13} & c_{14} & 0 \\
c_{21} & c_{22} & c_{23} & c_{24} & \ddots \\
c_{31} & c_{32} & c_{33} & c_{34} & \ddots \\
c_{41} & c_{42} & \ddots & \ddots & \ddots \\
0 & \ddots & \ddots & \ddots & \ddots \\
\end{bmatrix}
\]

Figure 5. Band Matrix Multiplication.

The Inner Product Step Processor

The processor for the matrix multiplication must be able to perform an operation called the inner product step. An inner product is generated by the multiplication of a single entry from each of two matrices. Groups of inner
products are accumulated in order to determine the value of an entry in the result matrix.

Figure 6 shows the inner product step processor. The processor contains three registers, Ra, Rb, and Rc. During each cycle, the processor clocks the data on its inputs, A, B, and C, into registers, Ra, Rb, and Re, and computes the inner product step,

\[ Rc <- Rc + Ra \times Rb. \]  \hspace{1cm} (1)

The new contents of registers, Ra, Rb, and Rc, are passed to neighboring processors through outputs A, B, and C.

Figure 6. The Inner Product Step Processor.
The Hexagonal Processor Array

A hexagonal array of inner product step processors can be used to perform the band matrix multiplication of Figure 5. The size of the array is determined by the band widths of the two matrices to be multiplied. In this case, a four by four array is needed because both of the matrices, A and B, have band widths of four. Notice that the size of the array is independent of the overall dimensions of the matrices. Figure 7 illustrates how data flows through the array. Each entry in matrix C is initially zero as it enters the array. As the entries of matrix C pass through the array, they accumulate inner product terms. The data flows through the array are timed such that the correct entries of matrices A, B, and C intersect to form the inner products.
Figure 7. Hexagonal Array for Band Matrix Multiplication.
CHAPTER IV, ADVANTAGES OF SYSTOLIC STRUCTURES

Having described a few application examples of systolic structures, some of the advantages of these structures become apparent. The following sections outline the basic features of systolic designs which make them advantageous for implementation in the VLSI technology.

Reduced I/O Bandwidth

Because systolic designs make multiple use of each input data item, systolic systems can provide high throughput with relatively small I/O bandwidth requirements. Data in a systolic system is used repetitively at each processor, so I/O activity is restricted to the input of data and the output of results. Intermediate results are stored internally to the systolic system and can be excluded from I/O activity.

Extensive Use of Concurrency

A systolic system achieves concurrency by pipelining and or multiprocessing. Pipelining allows the computation of results to be performed in assembly line fashion, and multiprocessing allows many results to be computed in parallel. The processing power of a systolic system comes from concurrent use of many simple processors rather than sequential use of one powerful processor.
Modular Design

In a systolic system, there are only a few types of simple processing elements. A VLSI device may contain huge amounts of circuitry in order to achieve a high level of performance. Design and implementation costs are reduced in systolic systems, because only a few simple processors need be designed regardless of the performance requirements. Higher levels of performance in a systolic system are achieved by simply including more processing elements into the system.

Simple and Regular Interconnections

Long and irregular interconnections for data are completely eliminated in a pure systolic system. The only global communication required in a systolic system is for the system clock and for power and ground connections. VLSI devices of small physical size generally cost less and out perform larger devices implemented in the same technology. Simple and regular interconnections lead to area-efficient layout on VLSI devices. Short wires on VLSI devices have reduced parasitic capacitances which lead to higher operational frequencies.
CHAPTER V, CONCLUSIONS

The examples of systolic designs discussed previously represent a small fraction of the possible applications of systolic architectures in VLSI design. Systolic designs apply in general to any compute-bound problem that is regular (Kung 1982). A regular problem is one where repetitive computations are performed on a large set of data. Many compute-bound problems are inherently regular in that they can be defined in terms of simple recurrences.

The ability to embed logic in memory, which VLSI technology brings, will change the way in which things are computed. It will become insufficient to evaluate algorithms by using computational models which are based on the Von Neumann architecture (Leiserson 1979). New models for parallel computation will need to be considered.
REFERENCES


