1981

Processor Displacement: An Area-Time Trade-Off Method for VLSI Design

David M. DeRuyck

Lawrence Snyder

John D. Unruh

Report Number:

81-394

TR 394

PROCESSOR DISPLACEMENT: AN AREA-TIME TRADE OFF METHOD FOR VLSI DESIGN

David M. DeRuyck[*]

Lawrence Snyder

John D. Unruh[*]

Department of Computer Sciences, Purdue University,

West Lafayette, Indiana 47907

ABSTRACT

Direct VLSI implementation of pipelined (systolic) processor arrays can lead to an "over parallelized" design causing the chip to have unused or underutilized area. Processor displacement design is a methodology that provides a spectrum of designs with differing time-area trade offs. The methodology is motivated, presented in detail, and illustrated by several examples. Direct experience for the Transitive Closure and Dynamic Programming systolic arrays is presented.

The work described herein is part of the Blue CHiP Project and is supported in part by the Office of Naval Research Contracts N00014-80-K-0816 and N00014-81-K-0360. The latter is Special Research Opportunities Task SRO-100.

[*]Authors permanent address: Bell Telephone Laboratories, Naperville, IL 60566.

PROCESSOR DISPLACEMENT: AN AREA-TIME TRADE OFF METHOD FOR VLSI DESIGN

David M. DeRuyck *

Lawrence Snyder

John D. Unruh *

INTRODUCTION

Area-time trade offs for computing functions in VLSI technologies have been the subject of much study in recent years [1,2,3,4]. Although important theoretically, these results tend to be based on asymptotic analysis and employ rather coarse resource measures. To date, their impact on VLSI design and layout has been minimal.

We report on a methodology called *processor displacement design* which provides area-time trade offs for pipelined arrays of processors (systolic arrays [7]) that are useful for practical VLSI design and layout problems.

Processor displacement gives the VLSI designer a range of choices that can be balanced to conform to constraints such as "pin"

The work described herein is part of the Blue CHiP Project and is supported in part by the Office of Naval Research Contracts N00014-80-K-0816 and N00014-81-K-0360. The latter is Special Research Opportunities Task SRO-100.

*Authors permanent address: Bell Telephone Laboratories, Naperville, IL 60566.

limitations and to increase the size of the problem solved with a given chip area.

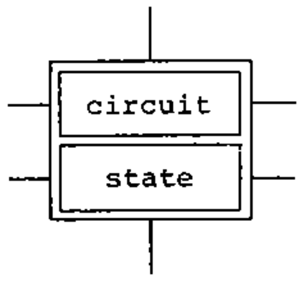
The widespread interest in systolic algorithms (see the references [5,6]) provides many opportunities to apply the methodology.

There are several benefits to processor displacement design. It provides a means of rapidly responding to the uneven improvements in process technology, e.g., when feature sizes reduce without a corresponding improvement in packaging technology. It gives a rational basis for deciding between serial or parallel data transfer on and off the chip The methodology can even be transferred to solving the problem of mapping large problems onto fixed size multiprocessor architectures.

The remainder of the paper is organized as follows. The next section gives an example of the use of the methodology as well as its benefits and liabilities. Next comes a thorough presentation of the methodology. The final section gives a summary and a discussion of some remaining issues.

MOTIVATION AND PROBLEM CONTEXT

In order to illustrate processor displacement, consider an idealized design situation. A systolic array processing element cell, visualized as containing processing circuitry and state memory, has been designed.



Four processing elements of a linear systolic array have been implemented as shown in Figure 1. (Kung and Leiserson's lower triangular banded-system solver is of this variety [8, pp. 285-288]. The state values are x, y and a, and the circuitry performs $y \leftarrow y + ax$.) We suppose that the four elements fully utilize the available chip area at $\lambda = 2x\mu m$ (for some real value $x > 0$), and that the timing is such that all processors are active on each step once the pipeline has been filled. Moreover, we assume the eight ports of the array are connected to the eight pins of our (over simplified) package. (We can ignore power, ground and clocking wires in this discussion.)

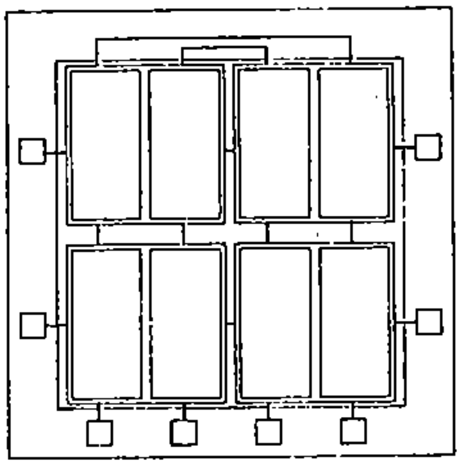


Figure 1.

Now suppose the circuit is to be fabricated with a $\lambda = x\mu m$ process. This factor of two density improvement enables the systolic array to be realized with only one-fourth the area of the previous implementation, (Figure 2). It is possible, therefore, to increase the implementation to sixteen processing elements (Figure 3). Notice that this can be done by a global reorganization of the cells without any cell redesign.

The sixteen element systolic array has twenty ports, but for the sake of this discussion, we still assume that only eight pins are available. It is

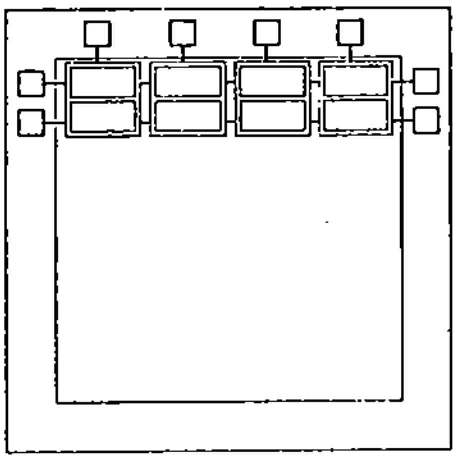


Figure 2.

possible to multiplex the pins, but doing so has a liability: *Processing elements must remain idle awaiting data.* Not only does this mean that we never have all sixteen copies of the processing circuitry active at once and thus waste silicon area, we must break open our completed cell design to add idling logic.

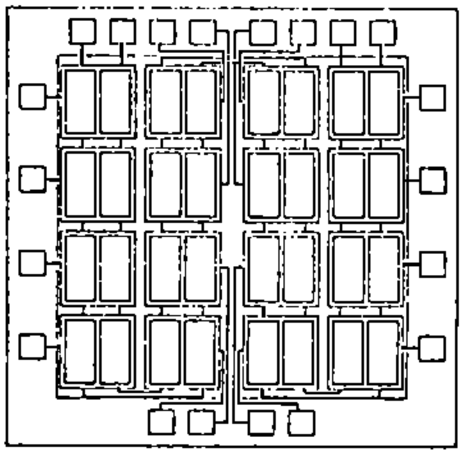


Figure 3.

Specification of sixteen processing elements without a corresponding increase in pin availability causes us to over parallelize the design. We simply have more processing circuitry than can be utilized. Although this simplified example can be fixed by adopting a larger package, it is illustrative of a general fact that cannot be ignored: ***There must be a balance between the parallel computation capability of the processing circuitry and the data transfer capability provided by the pins.*** It is this balance that processor displacement design is intended to control.

Continuing with the example, notice that in both of the $\lambda = x \mu m$ implementations, silicon area is wasted; either it is unused (Figure 2) or underutilized because of multiplexing (Figure 3). We can bring this wasted area into productive use by increasing the size of the problem solved on a chip. The idea is to reduce the amount of processing circuitry until it matches the data transfer capacity of the pins. (In this case, only four copies of the circuitry are required (Figure 4), although the situation is more complicated in general.) The remainder of the effective chip area is dedicated to state storage for processing elements that will be implemented by essentially multiprogramming the circuitry. A multiplexor is provided for this purpose. Each cluster of state storage cells and processing circuitry is called a *multiPE*. By using this processor displacement approach, we have increased the size of the implemented systolic array to 28.

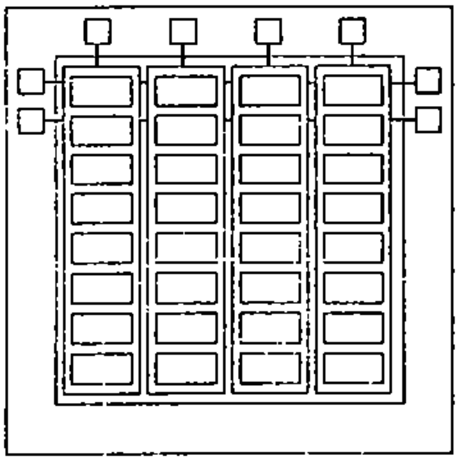


Figure 4.

This factor of seven improvement in effective density from a factor of two improvement in wire width was achieved without an increase in available pins. We paid for the improvement with a loss in time, but assessing the exact amount is difficult. The layout in Figure 2 is faster than that of Figure 1 by the speed improvements due to scaling, provided the data can be delivered fast enough. This gain is offset in the layout of Figure 3 since it is slower by a factor of four compared with Figured 2 assuming we do not multiplex the four "end" ports. Compared with Figure 3, the layout of Figure 4 looses a factor somewhat less than two under the same assumptions on multiplexing.

THE METHODOLOGY

The methodology to be described is not, as yet, a fully mechanical procedure suitable for computer implementation as a subroutine in a CAD system. It requires the designer to make judgements and estimates based on his experience. Nevertheless, the process is quite procedural and we will organize our presentation according to the six steps of the methodology.

In order to aid the reader in understanding the detailed discussion of the individual steps, we give the methodology in its entirety:

1. Develop an abstract systolic array processor (ASAP) to solve a problem of arbitrary size.
2. Design the processing element cell.
3. Figure the effective chip area and the pin count.
4. Determine if processor displacement is needed.
5. Compute the amount of processor displacement.
6. Layout the displaced processors and establish their timing protocols.

We now describe each step in detail.

1. *Develop an abstract systolic array processor (ASAP) to solve a problem of arbitrary size.* Systolic arrays are regular, locally connected arrays of one (or a small number of) processing element(s) that operate in a synchronous, pipelined manner and have external connections only at the perimeter. (See references [9,10] for characterizations by the inventors.) Three kinds of interconnection structure are typical: linearly connected, orthogonally connected, and hexagonally connected. Other connections have appeared, such as the toroidally connected Transitive Closure Systolic Array [11], and these are suitable for our mythology provided that the connections are sufficiently "local" that clustering preserves the interconnection structure.

In general, the "size" of an ASAP will be proportional to its perimeter and describes some property of the size of a give problem. For example, in the Kung and Leiserson Banded Matrix Systolic Arrays [8], it is the width of the band, not the size of the matrix, that determines the size of the array. Thus, the width is designated as the size, n, of the ASAP. In the case of the Transitive Closure Systolic Array [11], the size, n, is the number of vertices.

The ASAP will determine two functions which have the size as a parameter:

$m(n)$ = number of processing elements in an ASAP of size n,

$z(n)$ = number of inputs of an ASAP of size n.

The $z(n)$ function describes the number of "values" that must be transferred to an array of size n on each logical step, once the pipeline is full. These inputs are the candidates for multiplexing and so the function must be formulated with some care. In particular, for uniformity it may be wise to omit certain inputs from this function as was done with the four "end" wires in the example of the last section. The term "values" here refers to logical values, not bits. (See Step 3 for further discussion.) For example, the linear ASAP of the last section has $z(n) = n$.

2. *Design the Processing Element Cell.* The objective is not to design a single, monolithic cell, but rather to design two cells: a processing circuitry cell, *pc cell*, and a state memory cell, *sm cell*. Together, these two cells should define a processing element for the ASAP. But they should also define a family of cells, each one composed of one instance of the pc cell and multiple instances of the sm cells. These serve as multiPEs when multiplexing control logic is added. These conditions imply not only that the two cells have a compatible geometry, but that they are compatible with additional copies of the sm cell. (See Step 6 for a discussion of the effect of various clustering choices.) In order that "high level" manipulation of these components be possible without any internal modification, bus wires and selection lines should be incorporated into the sm cell.

Although many systolic arrays use only one kind of processing element, it is possible that several types will be required [8]. If this is the case, several pc cell types will obviously have to be designed. Several sm cells may be required too, although these tend to be the same over the entire array. When multiple element types are required, there will be geometric constraints within the multiPE as well as between multiPEs. Moreover there may be limits on the kinds of clustering possible, (see Step 6.) These considerations should obviously be assessed before designing.

There are two values that are determined by the cell design that will be needed later:

a = area of one processing element, i.e., area of a pe cell and an sm cell,

q = that fraction of a used by the sm cell, i.e., area of an sm cell/a.

Since the subsequent analysis only requires these two values and not the designs themselves, it is sufficient to have good estimates in order to proceed.

By proceeding on the basis of good estimates, information can be learned about two important design decisions. First, it is possible that given layout dimensions and certain clustering strategies can lead to multiPE geometries that do not pack well into the available chip area. This could make a processor displacement design unachievable. By estimating the area, we can determine the degree of clustering and this will allow us to infer preferred cell dimensions that will pack easily. Secondly, it may not be obvious how much parallelism is appropriate for data transmission. Since this decision will probably influence cell design, we can work through the methodology with several assumptions on the extent of parallelism and compare the results. This approach is recommended when speed is a significant consideration.

3. *Figure the effective chip area and the pin count.* Not all of the chip area is available for use by the systolic array processing elements. In addition to input/output pads, we may need area for multiplexor logic, bus wires for routing signals between the pads and the array elements, and possibly, buffers for timing (see Step 6). The area occupied by all of these overhead components should be determined (or estimated). Define the remaining area as

A = effective chip area.

We assume that A is a rectangle with dimensions that permit convenient packing of pc and sm cells.

Of the pins available on the intended package, some will be dedicated to power, ground and clocking signals. The remainder will be assigned to the data transmission activity of the systolic array. If certain ports were not included in the $z(n)$ definition (Step 1), then they must be permanently assigned to pins and the available number reduced accordingly. If there is a single output from the array, this should be included in the permanently assigned pins.

The remaining pins are available to be used by the multiPEs. If the processing elements use parallel input (and, perhaps output), then divide the available pins by the width of the parallelism. (This allows us to refer to a "pin" without reference to serial or parallel data transfer.) Now, if the ASAP produces multiple outputs, then we assume there are $z(n)$ of them and that they use the same degree of parallelism as the input. If so, divide the number of available pins by two, since for each pin assigned to the input, one must be assigned to an output. (Any other ratio can be handled analogously.) Define this result to be

p = number of available pins.

This is the number of data "values" that can be read in a single logical step (see Step 1).

4. *Determine if processor displacement is needed.* The objective of this step is to determine if there are sufficiently many pins to permit a direct implementation of a portion of the systolic array. Clearly,

$$y = \left\lfloor \frac{A}{a} \right\rfloor$$

processing elements could fit into the available chip area. By solving

$$m(n) = y$$

for n we can determine the size of the region of the ASAP that fits on one chip. This region of the ASAP will require $z(n)$ pins. Thus, if

$$z(n) \le p,$$

direct implementation of the systolic array is possible with full parallelism.

Care must be exercised in interpreting the results of the preceding test. If direct implementation is possible there is still the problem of packing the pc cell, sm cell pairs into the available area. In the following we assume that

$$z(n) > p$$

by a "substantial" amount.

5. *Compute the amount of processor displacement.* Using the previously defined functions,

$m(n)$ = number of processing elements in an ASAP of size n,

$z(n)$ = number of inputs of an ASAP of size n, and constants,

a = area of a processing element,

q = fraction of a required for state memory,

A = effective chip area

p = number of available pins,

we can derive an expression for the area occupied by a displaced processor design.

The key quantity in our analysis is the *bundling function*,

$$k(n) = \frac{z(n)}{p},$$

which describes the degree of multiplexing required of the p pins in order to deliver the $z(n)$ input required by one logical step of an ASAP of size n. Thus, in $k(n)$ processing steps the data for one logical step of a size n ASAP can be read. Since the systolic array is assumed to require this many inputs on each logical step, the parallelism of the ASAP should be reduced by a factor of $k(n)$. Thus, each multiPE should simulate $k(n)$ logical processing elements, and hence the name "bundling function."

With $k(n)$ bundling, the $m(n)$ logical processing elements can be simulated by $m(n)/k(n)$ multiPEs each containing $k(n)-1$ additional memory states. The total must thus satisfy

$$A = \left[\frac{m(n)}{k(n)} + \frac{k(n)-1}{k(n)} m(n) q\right] a.$$

Substituting for $k(n)$ and simplifying the resulting quotients yields

$$A = \left[m(n)\,\frac{p}{z(n)} + \frac{z(n)-p}{z(n)} m(n) q\right] a.$$

Further simplification gives

$$A = \frac{apm(n)}{z(n)}\left[1 + q\left[\frac{z(n)}{p} - 1\right]\right] \tag{1}$$

Since all quantities are known as a result of Steps 1-3, we can solve for n and determine the size of the ASAP that could be designed. Knowing n allows us to compute the bundling factor, k, from $k(n)$.

Example: Equation (1) can be used to derive the displaced processor design given in the second section. For the linear systolic array, $m(n) = n$ and $z(n) = n$. We used $p = 4$ before and if we use $a = 1$ and $q = \frac{1}{2}$, then $A = 16$ is appropriate. Thus

$$16 = \frac{1 \cdot 4 \cdot n}{n}\left[1 + \frac{1}{2}\left[\frac{n}{4} - 1\right]\right]$$

$$= 4 + \frac{n - 4}{2}$$

and $n = 28$. Since $k(n) = n/4$. The bundling factor is 7.

Obviously, some judgement must be employed in applying equation (1). For example, since the bundling factor describes the extent of the multiplexing and the number of displaced processors in a multiPE, the design is greatly simplified if it is an integer. Thus, one might choose the greatest n less than that determined by equation (1) such that $k(n)$ is an integer.

6. *Layout the displaced processors and establish their timing protocols.* With a bundling factor of k established in Step 5, we layout the multiPEs such that each contains a copy of the pc cell and k copies of the sm cell. The multiPEs are then laid out in the available area such that their input/output ports are connected either to input/output pads or the ports of adjacent processing elements. The layout problem, as has already been mentioned, is subject to packing difficulties when the dimensions of the available area are not multiples of the dimensions of the multiPEs. Wiring the ports of a linear systolic array should be a straightforward operation. But wiring and timing two dimensional systolic arrays presents some interesting problems.

Each multiPE will simulate a contiguous region of k logical processing elements of the ASAP. The geometry of this region significantly effects the multiplexing operation. Let the bundling factor $k = 4$ and consider two multiPEs that simulate regions of the ASAP with input on two sides, each with different geometrics.* (Refer to Figure 5.) MultiPE B simulates a 2×2 block

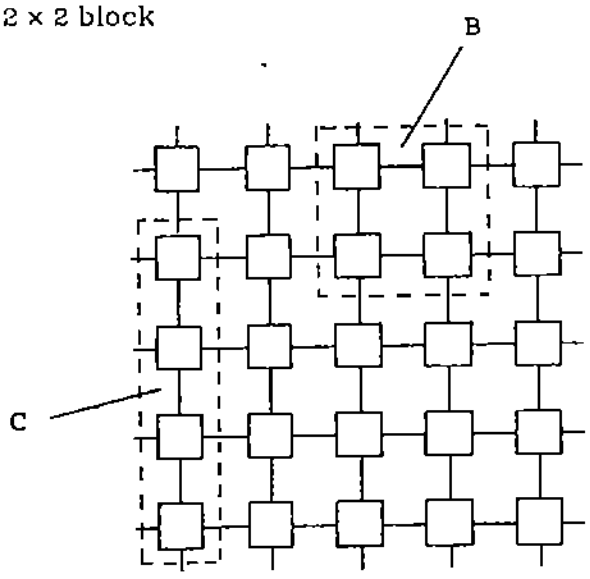


Figure 5.

of logical processing elements while multiPE C simulates a columnar region.

The key difference between multiPEs B and C is that when they appear along the perimeter (ignore the corner case for the moment) of the ASAP, they have different numbers of external ports; B has two while C has four. Since each pin will deliver four values in a logical step, each C multiPE processes exactly the amount of data provided by the pin. But each B multiPE can only use two inputs during a logical cycle, so two B multiPEs must be attached to one pin.

*Notice that we are not referring to the geometry of the pe cell and sm cell organization of the layout.

The main consequence of several multiPEs sharing one pin is that the order in which the constituent processing elements are simulated cannot be the same. For example, suppose two B multiPEs appear along the perimeter and share a pin. Then on processing step one, they cannot both simulate processing element <1,1> since that would require them both to read from the pin simultaneously. We must either introduce a buffer or change the order in which constituent processing elements are simulated, so that they are not both reading at once. Notice that linear multiPEs are not subject to this difficulty.

Handling the external input for the multiPE that simulates the corner processing element adds a bit more complexity because for either geometry, it has more inputs than the others. In either case the multiPE will be connected to two different pins. Again, changing the ordering of the simulations (now it must be done along both sides) or buffering solves the problem.

Perhaps the simplest solution is to use C multiPEs such that the corner multiPE simulates in sequential order down the column and the $k-1$ adjacent C multiPEs simulate in an order that is a cyclic shift (upward) of this sequence. (Obviously, analogous remarks apply for the output.)

EXAMPLES

We have used the processor displacement methodology to analyze two systolic array algorithms. The computed results are summarized in Table I. The processing element layouts use Mead and Conway [9] design rules. The expression "$p = z(n)$" means that pins are assumed to be unlimited.

TABLE I.

	Transitive Closure			Dynamic Programming	
	n^2		$m(n)$	n^2	
	$2n$		$z(n)$	$5n$	
params	$6mm \times 6mm$		A	$6mm \times 6mm$	
	$11286\lambda^2$		a	$285000\lambda^2$	
	0.5		q	0.25	
λ	$2\mu m$	$1\mu m$		$2\mu m$	$1\mu m$
n	28	56	$p=z(n)$	5	11
$m(n)$	784	3136		25	121
n	28	66	package	8	19
$m(n)$	784	4356	size =64	64	361
multiPEs	784	1452		16	38
n	32	71	package	9	20
$m(n)$	1024	5041	size =40	81	400
multiPEs	512	1261		11	24

For the Transitive Closure Systolic Array [11], n is the number of vertices. Since the input does not overlap with the output, the same pins are used for both operations. Notice that there is no benefit in processor displacement for $\lambda = 2\mu m$ and package size of 64 since only 56 pins are needed in addition to the four overhead pins, i.e., in this technology it is possible to have full parallelism.

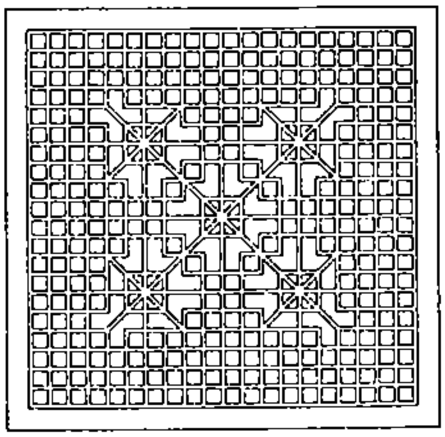
The Dynamic Programming Systolic Array solves string distance measurement using an $n \times n$ array with six bit data. Thus, for a 64 pin package $p = (64 - 4)/6 = 10$ logical pins. Each cell requires three values from the north and two from the west.

Notice that the values in Table I may be optimistic in the sense that "divisibility constraints" have been ignored. To use the pins optimally, n should be chosen so that $z(n)/k(n)$ is an integer. If it is not, some bandwidth will be wasted or the timing will be significantly complicated. A further constraint that one might require is for $m(n)/k$, the number of multiPEs used, to be an interger. Fractional numbers could be achieved by having multiPEs simulate fewer than k logical processing elements.

The Dynamic Programming array for packages of 64 pins and $\lambda = 2\mu m$ is the only table entry to satisfy both constraints. We can often reduce the problem size to enforce the divisibility constraints. For example, in the Transitive Closure, package of 40 pins, $\lambda = 1\mu m$ case, the problem size must be reduced to $n = 54$ before the divisibility constraints are met. This choice reduces the area utilization from better than 98% to about 61%. However, if the available area were about 1.5% larger (or equivalently, the cells were proportionately smaller), a problem size of $n = 72$ (with $k = 4$) would be possible. This would require redesigning the input/output pad area. In general the divisibility constraints can be controlled with several parameters and the optimal combination depends on the designer's judgement.

SUMMARY

We have presented a six step methology that allows the amount of parallelism in a systolic array to be matched to the data transfer bandwidth provided by the pins. The technique appears to be applicable to systolic arrays with a wide variety of characteristics. It provides a means of evaluating the benefits of serial *vs* parallel data transfer and for fully utilizing the available silicon.



Temple Window Lattice, Mount Omei, Szechwan, (date unknown).

REFERENCES

[1] C. D. Thompson
Complexity Theory for VLSI
Ph.D. Thesis, Carnegie-Mellon University, 1979

[2] R. P. Brent and H. T. Kung
Area Time Complexity of Binary Multiplication
Technical Report CMU-CS-136, Carnegie-Mellon University, 1979

[3] Jean Vuillemin
A Combinatorial Limit to the Computing Power of V.L.S.I. Circuitry
Proceedings of the 21*st* Annual Symposium on the Foundations of Computer Science, IEEE, 1980

[4] R. J. Lipton an Robert Sedgewick
Lower Bounds for VLSI
Proceedings of the 13*th* Annual Symposium on the Theory of Computing, 1981

[5] John P. Gray (ed.)
VLSI 81 Proceedings of an International Conference on VLSI
Adademic Press, 1981

[6] H. T. Kung, Bob Sproull, and Guy Steele (eds.)
VLSI Systems and Computations
Computer Science Press, 1981

[7] H. T. Kung and C. E. Leiserson
Systolic arrays (for VLSI)
Technical Report CS-79-103, Carnegie-Mellon University, 1979. (Also in reference [8].)

[8] Carver Mead and Lynn Conway
Introduction to VLSI Systems
Addison Wesley, 1980

[9] Charles E. Leiserson
Area-Efficient VLSI Computation
Ph.D. Thesis, Carnegie-Mellon University, 1980

[10] H. T. Kung
Why Systolic Arrays?
Computer, IEEE, January 1982

[11] L. J. Guibas, H. T. Kung and C. D. Thompson
Direct VLSI Implementation of Combinatorial Algorithms
Proceedings of the CalTech Conference on VLSI, California Institute of Technology, 1979