Generalized Signed-Digit Multiplication and Its Systolic Realizations

Ching Yu Hung and *Behrooz Parhami*

Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560, USA

Abstract — A generalized signed-digit (GSD) number system is a fixed-radix number system, with radix r and digit set $\{-\alpha, -\alpha+1, \ldots, \beta-1, \beta\}$, where $\alpha \geq 0$, $\beta \geq 0$, and $\alpha+\beta+1 > r$. The redundancy of GSD number systems allow digit-parallel addition, based on which linear-time algorithms for multiplication are devised. Systolic and semisystolic schedules derived from these algorithms lead to the design of one-dimensional and two-dimensional array multipliers that have $O(n)$ latency for n-digit operands.

Introduction

In applications requiring arithmetic operations with long operands, signed-digit number systems offer the advantage of limited carry propagation by exploiting the redundancy in the representation. With limited carry propagation, operations can be performed in parallel for all digits of a number. Consequently, the computation can be completed much faster in a parallel system. Limited carry propagation also enables most-significant-digit-first computation, which can be applied to digit sequential systems to reduce the latency. A conventional nonredundant representation, on the other hand, slows down the computation by requiring the carry to propagate from the least significant to the most significant digit, which in the worst case takes $O(n)$ time for n-digit operands.

Signed-digit (SD) number systems and associated arithmetic algorithms were first proposed by Avizienis [1]. Ercegovac and Lang [5] have proposed a systematic method to design on-line arithmetic algorithms with SD representation. Addition, multiplication, division, and square-rooting algorithms have been presented. Preparata and Vuillemin [9] discuss division algorithms and propose systolic implementations. All these works use a symmetric digit set for the SD number system, i.e., the digit set has the form $\{-\alpha, \ldots, 0, \ldots, \alpha\}$, and the *redundancy index* ρ, defined as the size of digit set minus the radix, is bounded by $0 < \rho < r$.

Recently Parhami [8] has considered a generalized signed-digit (GSD) number system in which the digit set need not be symmetric, and the redundancy index has no upper bound. GSD representations cover conventional SD number systems as well as several other redundant representations. GSD addition algorithms are also discussed fully in [8].

In this paper, four algorithms for carry-free and limited-carry GSD multiplication are presented. They differ in the information passed between adjacent digit positions after each product accumulation step. The carry-free algorithm requires a transfer digit, the estimate-transfer algorithm requires an estimate digit then a transfer digit, and the two-phase-transfer and parallel-transfer algorithms both require two transfer digit exchanges. The two-phase-transfer algorithm does so serially, while the parallel-transfer algorithm does so in parallel. To show how these algorithms can be implemented, we present a semisystolic and a systolic schedule for each algorithm and derive the latency. Additionally, we discuss two-dimensional and one-dimensional array processors for carry-free multiplication.

Carry-Free Multiplication

The following is the carry-free algorithm:

1. $z_j := 0$ for each j
2. For $i = n-1$ downto 0 do begin
3. For $j = 0$ to $2n-1$ do begin
4. If $i \leq j \leq i+n-1$, then $p_j := z_j + x_i y_{j-i}$
5. Otherwise $p_j := z_j$
6. end
7. Find t_{j+1} and w_j s.t. $p_j = rt_{j+1} + w_j$ for each j
8. $z_j := w_j + t_j$ for each j
9. end

The algorithm accepts two n-digit GSD numbers x and y, and computes their $2n$-digit product $z = xy$. The structure of the algorithm is just a standard serial-parallel multiplication augmented with transfer steps (lines 7 and 8) to absorb partial products generated in each digit positions. On line 7, a transfer digit t_{j+1} and remainder digit w_j are found based on p_j. Line 8 adds the left-shifted t_j to w_j and produces the partial product digit z_j.

We have found that this algorithm always requires a larger partial product digit set, denoted as $[-\alpha', \beta']$, than the input product set, $[-\alpha, \beta]$. These parameters are constrained by:

$$\alpha' + \beta' + 1 - r \geq \lceil \frac{\alpha\beta}{r-1} \rceil + \lceil \frac{\max(\alpha^2, \beta^2)}{r-1} \rceil$$

Let $\rho \equiv \alpha + \beta - r + 1$ be the redundancy index of the input digit set. Under the normal assumption that $\rho < r$, the partial product digit set is up to twice as large as the operand digit set. The incurred increase of one bit in logic or computation time is not significant.

In the end a carry-free correction step is required to convert the product digit set to the input digit set. This correction need only be performed once for each multiplication, so it is not a significant drawback.

Limited-Carry Multiplication

The transfer steps of the estimate-transfer algorithm for GSD multiplication are as follows. The rest of the algorithm is identical to the carry-free algorithm.

7. Compute e_{j+1}, an estimate for t_{j+1}, for each j
7a. Knowing e_j, find t_{j+1} and w_j s.t. $p_j = rt_{j+1} + w_j$ for each j
8. $z_j := w_j + t_j$ for each j

On line 7, an estimate e_{j+1} for the transfer digit t_{j+1} is found based on the value of p_j, where $e_{j+1} \in \{1, 2, \ldots, E\}$. On line 7a, e_j is shifted one digit position to the left, and the actual transfer digit t_{j+1} and the remainder digit w_j is chosen so that $p_j = rt_{j+1} + w_j$. Finally on line 8, t_j is added to w_j to form z_j.

The estimate e_{j+1} is determined by comparing p_j against some thresholds $T_1 < T_2 < \cdots < T_{E+1}$, and $T_i \leq p_j < T_{i+1}$ indicates $e_{j+1} = i$. After an estimate $e_{j+1} = k$ is sent out, the actual transfer digit t_{j+1} must comply with the estimate by being in a predetermined subrange $[-t_k^-, t_k^+]$. Due to the length limitation, we cannot present detailed derivation of T_i, t^-, and t^+. Instead, some important intermediate results are shown.

On line 7, the remainder digit w_j must have a range no smaller than r, since r is the divisor. On line 8, in order for z_j to fall into the digit set $[-\alpha, \beta]$, the range of t_j must be no larger than $\alpha + \beta + 1 - r = \rho$. Therefore we have

$$t_k^+ + t_k^- + 1 \geq \rho,$$

which dictates the size of each subrange of transfer digit. These subranges must also overlap, so that proper margin is left on p_j for the incoming transfer digit. We have derived the requirement

$$t_k^- + t_{k-1}^+ \geq \lceil (t_E^+ + t_1^- - (\alpha + \beta + 1))/r \rceil,$$

which dictates the amount of overlap between the subranges, where $t_1^- = \lceil \alpha\beta/(r-1) \rceil$ and $t_E^+ = \lceil \max(\alpha^2, \beta^2)/(r-1) \rceil$. By requiring that the total range $[-t_1^-, t^E]$ be covered by the union of all subranges, we have derived the minimum cardinality of estimate, $E_{\min}$, as

$$E_{\min} = \lceil \frac{t_E^+ - t_1^+}{t_k^+ - t^+ k - 1} \rceil + 1 = \lceil \frac{t_E^+ + t_1^- - \rho}{\rho - t_k^- + t_{k-1}^+} \rceil.$$

Finally, the threshold constants for determining e_{j+1} are chosen in the intervals:

$$-rt_k^- - \alpha + t_1^- \leq T_k \leq rt_{k-1}^+ + \beta - t_E^+ + 1.$$

To ease the task of determining e_{j+1}, we would like to choose threshold constants to be numbers that are easy to compare with. In most practical cases where r is a power of 2, we may, for example, choose the thresholds to be multiples of r. Doing so might incur a slight increase in E in some cases.

CH 3381-1/93/$01.00 ©1993 IEEE

Variations on Limited-Carry Multiplication

Two-Phase-Transfer Algorithm

The two-phase-transfer algorithm uses the following transfer steps:

7. Find t_{j+1} and w_j s.t. $p_j = rt_{j+1} + w_j$ for each j
7a. $q_j := w_j + t_j$ for each j
7b. Find u_{j+1} and v_j s.t. $q_j = ru_{j+1} + v_j$ for each j
8. $z_j := v_j + u_j$ for each j

The transfer steps are basically the transfer steps of carry-free algorithm performed twice. We have analyzed the algorithm and proven that for the algorithm to be applicable, we must have

$$\max(\alpha, \beta) \leq \frac{\rho(r^2-1) - 2r + 4}{\rho + r - 1},$$

which is satisfied by most GSD number systems.

Parallel-Transfer Algorithm

The parallel-transfer algorithm uses the following transfer steps:

7. Find t_{j+2}, u_{j+1} and w_j s.t. $p_j = r^2 t_{j+2} + ru_{j+1} + w_j$ for each j
8. $z_j := w_j + t_j + u_j$ for each j

Line 7 generates two transfer digits, t_{j+2} and u_{j+1}, carrying weights r^2 and r respectively. Then, they are shifted to the left, two digit positions for t_{j+2} and one digit position for u_{j+1}, and added to the remainder w_j to obtain partial product z_j. We have shown that the partial product digit set must be at least $2r+1$ in size. Consequently, in almost all cases, a final correction is required to convert the product digit set to the input digit set.

Sparse-Transfer Variation

Since both the two-phase-transfer and the parallel-transfer algorithms are able absorb a larger partial product digit, a sparse-transfer variation has been conceived. Instead of performing n transfer steps, only $\lceil n/m \rceil$ transfer steps are carried out. For example, in a minimally redundant signed-digit system, i.e., $\rho = 1$, using the parallel-transfer method, m can be 4 for $r = 2$ and 5 for $r > 2$.

Mapping Algorithms onto Arrays

The mapping from algorithms to array processors is well covered in [6] and has become a standard technique. In [4], Chen constructs 18 multiplier designs from the long multiplication algorithm. In this section, we show some examples of how the algorithms presented in the previous sections can be implemented.

We derive systolic and semisystolic schedules for each the four algorithms presented. Schedules for the sparse-transfer variant of each algorithm can be derived similarly and are thus omitted. Next, from the semisystolic schedule for carry-free algorithm, we derive a linear array and a two-dimensional array. Our procedure and terminology in the development of schedules and array processors follow those of [6].

Semisystolic Schedule for Carry-Free Multiplication

The most natural schedule of the carry-free multiplication is the digit-parallel schedule. Figure 1 shows a schedule for a four-by-four-digit multiplication.

There are two kinds of nodes in Figure 1, represented by circles and squares. Each circle performs lines 8, 4, and 7, and each square performs lines 8, 5, and 7. The exceptions are: the top row only executes lines 4 or 5, and 7; and the bottom row only executes line 8. The arcs represent dependencies in the algorithm. The vertical and diagonal arcs represent the transmission of w_j and t_j values, respectively, between the operations on lines 7 and 8.

A semisystolic schedule is also specified on the figure, by *equitemporal hyperplanes*, the horizontal dashed lines, and by the *schedule vector* $\vec{S}$. Nodes on the same hyperplane are computed at the same time. The schedule vector represents the advancing of time. The interpretation of $\vec{S}$ in Figure 1 is that the nodes in the top row of nodes are computed first, and those in the bottom row last.

This schedule is called semisystolic because the operand digits x_i need to be broadcast among processors in the same row. When the number of digits becomes large, physical dimensions of array grow accordingly, and the communication delay might slow down the computation significantly.

From this schedule, one-dimensional and two-dimensional array processors can be derived. The two-dimensional array, shown in Figure 2, has the same topology as the schedule. Arcs have been added for the communication of x_i and y_i values. Dots in the figure represent single clock cycle delays. The array processor is pipelined for full utilization. As the x and y operand streams are piped into the array, one multiplication operation is completed at each clock cycle. Figure 3 shows the linear array of processors derived from the schedule in Figure 1. The latency of both arrays, determined by the schedule, is $n+1$ clock cycles for n-by-n-digit multiplications.

Systolic Schedule for Carry-Free Multiplication

For most-significant-digit-first computations, another schedule, shown in Figure 4, can be used. In this schedule, equitemporal hyperplanes run diagonally through the graph. Because nodes in the same row are not computed at the same time, operand digits x_i have time to pass from one node to the next, and consequently no broadcast of data is required. This schedule is therefore a *systolic* one.

Here we omit figures for two-dimensional and one-dimensional arrays that can be easily derived from the schedule. The latency of both arrays is $4n$ for n-by-n-digit multiplications; it takes $2n+1$ clock cycles to reach the lower-left corner node, and $2n-1$ more to reach the lower-right one. The two-dimensional array is pipelined and able to complete one multiplication per clock cycle. The latency, in terms of clock cycles, is about four times that of the semisystolic arrays. However, the systolic design does not necessarily have four times longer latency. Duration of clock cycle depends on the logic and wiring delay between clock ticks; systolic design supports much faster clocking because it requires only local communication.

Estimate-Transfer and Two-Phase-Transfer Multiplication

The limited-carry algorithm and its two-phase-transfer variant have the same pattern of logical dependency; they both require two inter-digit transfers in series. Therefore, they have the same topology in their schedules. The logical dependency for both algorithms is shown in Figure 5.

For the estimate-transfer algorithm, the dotted circular nodes in the figure represent lines 8, 4, and 7, the white circular nodes represent lines 8, 5, and 7, and the square nodes represent line 7a, with possible exclusions in the top and bottom rows.

For the two-phase-transfer algorithm, the dotted circular nodes in the figure represent lines 8, 4, and 7, the white circular nodes represent lines 8, 5, and 7, and the square nodes represent lines 7a and 7b, again with possible exclusions in the top and bottom rows.

A semisystolic schedule is also shown in Figure 5. Equitemporal hyperplanes run horizontally, and reflect the digit-parallel computation. The passing of operand digits x_i requires broadcast among circular nodes in the same row. Again, two-dimensional and one-dimensional array processors can be derived from the schedule. The latency of both designs is $2n+1$ clock cycles. The two-dimensional array is again pipelined and able to complete one multiplication per clock cycle.

Next, we show a systolic schedule in Figure 6. It implies a most-significant-digit-first computation, and is useful when broadcasting is to be avoided, or when digit sequential computation is favored.

In the figure, equitemporal hyperplanes run diagonally through the graph, and there is a two-unit time difference between two adjacent rows. The latency imposed by the schedule is $6n$; it takes $4n+1$ clock cycles to reach the lower-left corner node, and $2n-1$ more to reach the lower-right one. The latency, in terms of clock cycles, is about three times that of the semisystolic schedule.

Parallel-Transfer Multiplication

The semisystolic schedule for the parallel-transfer algorithm is similar to that for the carry-free (single-transfer) algorithm. With the parallel transfer of two transfer digit, we have extra arcs connecting digits two positions away, as shown in Figure 7. The latency of the schedule is $n+1$, same as the semisystolic schedule of carry-free algorithm.

The systolic schedule is shown in Figure 8. It requires longer latency compared to that of the carry-free multiplication. With longer arcs carrying the transfer digits, we have extra delay between rows; 3 clock cycles instead of 2. Consequently, the latency is $5n$ clock cycles.

Comparison of Algorithms

To compare the algorithms, in Table 1 we list the latencies in semisystolic and systolic schedules and the time and space required in each clock cycle. The latter can only be estimated because there are too many parameters to consider.

Time estimates are based on approximating each computation step in the algorithms with the number of additions and multiplications. An addition costs one A in space and one A in time, while one multiplication costs one M in space and one M in time. The estimates assume that r is a power of 2, as should be in most practical cases. In terms of space-time tradeoff, we favor minimum time and then optimize space requirements under the minimum time constraint.

Table 1: Comparison of algorithms

Algorithm		carry-free	estimate	2-phase	parallel
Latency	Semisys.	$n+1$	$2n+1$	$2n+1$	$n+1$
	Systolic	$4n$	$6n$	$6n$	$5n$
Time/space	4	$A+M/A+M$	$A+M/A+M$	$A+M/A+M$	$A+M/A+M$
in each line	7	$2A/2A$	$(E-1)A/A$	$2A/2A$	$2A/3A$
	7a		$2A/2A$	A/A	
	7b			$2A/2A$	
	8	A/A	A/A	A/A	A/A
Time/space	circle	$4A+M/2A+M$	$(E+1)A+M/A+M$	$4A+M/2A+M$	$4A+M/3A+M$
clock cycle	square	$2A/2A$	$2A/2A$	$3A/2A$	$2A/3A$

- The product accumulation step takes $A+M$ in time and $A+M$ in space.
- Finding a transfer digit t_{j+1} and a remainder w_j from a position sum p_j takes $2A$ in time and $2A$ in space. First, t_{j+1} and w_j are assigned with p_j div r and p_j mod r by taking the upper and lower portions of p_j (routing takes no time). Next, w_j is compared against the allowed upperbound, and adjustment is made if necessary. The comparison is counted as one addition. The adjustment adds one to t_{j+1} and subtracts r from w_j, so is counted as two parallel additions.
- Finding t_{j+2}, u_{j+1}, and w_j in the parallel-transfer algorithm is assumed to take $2A$ in time and $3A$ in space. First, the three variables are assigned with the upper, middle, and lower portions of the number p_j. Then, u_{j+1} and w_j are compared with their upper bounds simultaneously, and finally adjustment is made in parallel to the three variables if necessary. Here we assume that the range of u_j has enough redundancy to absorb the the adjustment.
- Finding an estimate digit e_{j+1} is assumed to take $E-1$ sequential comparisons with the threshold constants T_k. Since E is a small number, this approach is usually more efficient than binary search. We need $(E-1)A$ in time and A in space.

From Table 1, we observe that when $E=2$, the estimate-transfer algorithm requires the least amount of time in a clock cycle, and thus might lead to the design with the fastest clock. In terms of latency, the carry-free algorithm, with either semisystolic or systolic schedule, is the best. However, there are other considerations in design: digit set size, input/output conversion, and communication delay. The sparse-transfer variations of the algorithms should also be considered in search of an efficient design.

Conclusions

Algorithms for multiplying generalized signed-digit numbers have been discussed. The first algorithm, carry-free multiplication, requires a transfer digit between accumulation of product terms. To apply it to GSD, we have to use a larger digit set for the partial product than for the input operand digits. This is not a significant drawback, since a conversion step can be performed after the multiplication to bring the product digits back to the input digit set. Such conversions are special cases of more general schemes currently under investigation [7].

The second algorithm, estimate-transfer algorithm, requires first a binary estimate digit, then a transfer digit, between accumulation of product terms. With the maximally redundant symmetric SD number system, the algorithm requires only a binary estimate, and is therefore easiest to implement.

The third algorithm, two-phase-transfer algorithm, requires two passes of transfer digits between product accumulation steps. The amount of information transferred is greater than that required by the second algorithm. This algorithm is applicable to almost all GSD number systems.

The fourth algorithm, parallel-transfer algorithm, generates and processes in parallel two transfer digits carrying weights r^2 and r. It has even larger capacity than the third algorithm. In this algorithm, the partial product digit set must be at least $2r+1$ in size, and thus output conversion is almost always required.

The sparse-transfer variants of the algorithms perform several accumulation steps between transfer stages. The number of transfer stages is therefore reduced. This is particularly useful for the parallel-transfer algorithm, which has the largest capacity in a transfer stage.

Semisystolic and systolic schedules for the algorithms were presented. The latencies of all algorithms are $O(n)$; the constant factors differ. The first algorithm has the least latency in terms of clock cycles, while the second one leads to the fastest clock rate if a binary estimate is used. One-dimensional and two-dimensional arrays can be designed from both semisystolic and systolic schedules. It is also possible to have finer pipelining in the two-dimensional arrays to increase the clock rate.

As a suggestion for further study, detailed modeling of the multipliers, including logic cost, communication delays, and logic delays as functions of the digit set can be performed. With detailed modeling, the sparse-transfer approach can be optimized for each algorithm. Additionally, we may study application of the algorithms to division of GSD numbers and to larger problems such as vector and matrix computations.

References

[1] A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, EC-10:389–400, 1961.

[2] P. R. Cappello and K. Steiglitz. Unifying VLSI array design with linear transformations on space-time. *Advances in Computing Research*, 2:23–65, 1984.

[3] I-Ngo Chen and R. Willoner. An $O(n)$ parallel multiplier with bit-sequential input and output. *IEEE Transactions on Computers*, 28(10):721–727, October 1979.

[4] M. C. Chen. The generation of a class of multipliers: Synthesizing highly parallel algorithms in VLSI. *IEEE Transactions on Computers*, 37(3):329–338, March 1988.

[5] M. D. Ercegovac and T. Lang. On-line arithmetic: A design methodology and applications in digital signal processing. *VLSI Signal Processing*, III:252–263, 1988.

[6] S. Y. Kung. *VLSI Array Processors*. Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[7] B. Parhami. Systolic converters for number radices and digit sets. Paper in preparation.

[8] B. Parhami. Generalized signed-digit number systems: A unifying framework for redundant number representations. *IEEE Transactions on Computers*, 39(1):89–98, January 1990.

[9] F. P. Preparata and J. E. Vuillemin. Practical cellular dividers. *IEEE Transactions on Computers*, 39(5):605–614, May 1990.

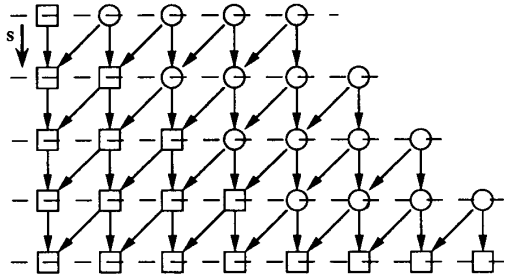

Figure 1. Semisystolic schedule for the carry-free algorithm

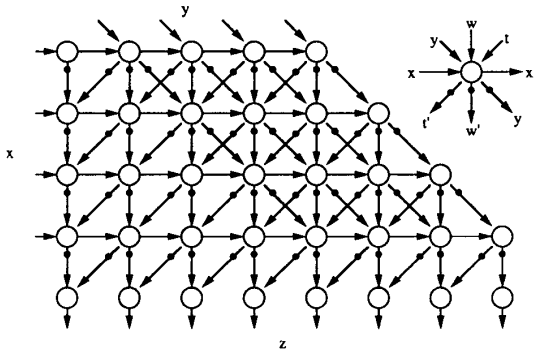

Figure 2. Two-dimensional array processor from the schedule in Figure 1

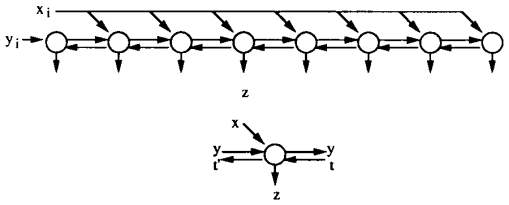

Figure 3. One-dimensional array processor from the schedule in Figure 1

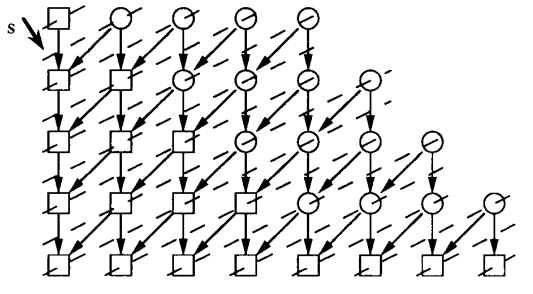

Figure 4. Systolic schedule for the carry-free algorithm

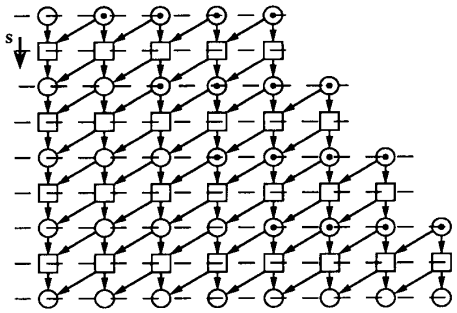

Figure 5. Semisystolic schedule for the limited-carry algorithm

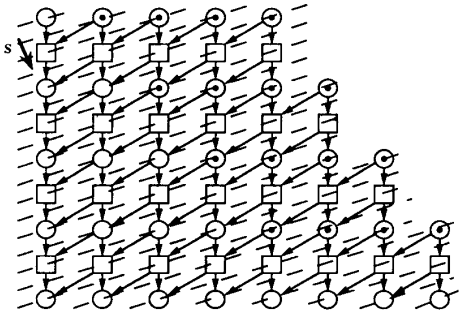

Figure 6. Systolic schedule for the limited-carry algorithm

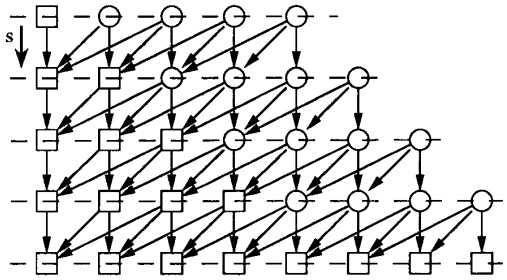

Figure 7. Semisystolic schedule for the parallel-transfer algorithm

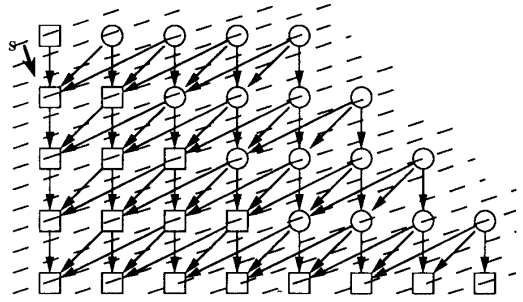

Figure 8. Systolic schedule for the parallel-transfer algorithm