The Perfect k-Shuffle: a broad purpose architecture for parallel computation

Alberto Negro
Dipartimento di Informatica ed Applicazioni
Universita' di Salerno
Salerno, Italy

and

Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

86-50-1

The Perfect k-Shuffle: a broad purpose architecture for parallel computation

Alberto Negro
Dipartimento di Informatica ed Applicazioni
Universita' di Salerno
Salerno - Italy
and
Department of Computer Science
Oregon State University
Corvallis - Oregon

February 5, 1986

1 Introduction

The PERFECT SHUFFLE (PS) has been the first proposed *broad purpose* network for fast parallel processing. Defined in 1971 by Stone [1] as a substitute for the hypercube machine, it effectively supports a class of algorithms with polylogarithmic time complexity exhibiting an optimal layout for several of them.

Natural competitor of the perfect shuffle is the CUBE-CONNECTED CYCLES (CCC). It was defined in 1979 by Preparata and Vuillemin in order to efficiently implement algorithms in two highly parallel classes: the ASCEND and the DESCEND [3]. Though presenting the same computational properties of the PS, the CCC shows a very regular layout and the optimal $area\,time^2$ (*or* AT^2) measure for problems like permutation, odd-even merge, fast fourier transform, etc. for the entire range of $T\ [O(log\ n), O(\sqrt{n})]$. Furthermore the DESCEND class has the property to efficiently emulate the computational paradigms of different architectures like the perfect-shuffle, the perfect-unshuffle, and the n-cube. [3]

The MESH of TREES (also Orthogonal Trees [15]) well fits in the fast parallel processing for the interesting properties it exhibits. Not a cube emulator it specifies a paradigm based on the multiplex-demultiplex characteristics rather than on the data adjacencies. It has been defined several times by different authors [15,17], a complete analysis is present in [15] on different VLSI models

of computations. Using this interconnection scheme problems like sorting, FFT, matrix multiplication, connected components are effectively solved.

The PS, the CCC, and the MOT are known as *broad purpose* parallel architectures because of their capability to efficiently run *classes* of algorithms. On the other side several special purpose architectures have been proposed to optimally solve specific problems. The shape of this *ad hoc* architectures is almost always very complex and is the result of the composition of basic network like PS, CCC, Mesh etc. For this reason they are sometimes called *hybrid*.

The aim of this paper is to propose two parametrized architectures and to study their computation models. Based on the SHUFFLE network [1] and on its generalized version, the k-SHUFFLE [11,4], they join the characteristics of the Hybrid Architectures, of the cube emulators (PS and CCC) and of the pipeline exhibiting high performances and an acceptable VLSI realization. In particular the Shuffle-Connected Arrays (SCA) and the Perfect k-Shuffle (P_kS) will be defined.

The SCA is composed by a fast architecture (the PS) and a slow architecture (the Linear Array (LA)). Even though the overall organization of the network depends on the size of the LA, the computation paradigm is always the same. It exhibits optimal AT^2 complexity for the Ascend-Descend algorithm in a wide range of time complexities.

The P_kS is composed by the generalized PS and the Binary Tree. Shown to be the hardware realization of an highly parallel scheme of computation: the Generalized Recursive Combination, it spans the emulation capabilities from the Binary Tree to the Perfect Shuffle passing throught the Mesh Of Trees (MOT).

Shuffle-like architectures AT^2 optimal in a wide range of T are already known from the previous literature. The mesh of CCC and the mesh of PS are shown to be AT^2 optimal for the merging in the range of $T\ [O(log\ n), O(\sqrt{n})]$ in [13], also Bilardi [21] in his thesis use the cycles and meshes to achieve the optimal AT^2 tradeoff for sorting in the range of $T\ [O(log\ n), O(\sqrt{n})]$.

Proposed as a permutation network, the k-SHUFFLE was used by D. Lawrie [4] for problems of efficient access to memory, Wu in [10] discusses carefully the full set of permutation it is possible to perform with the shuffle network. In [11] the generalized shuffle is introduced and discussed as an efficient interconnection network for SIMD machines.

Our main results are:

1. The SCA in the pipelined version *is* the CCC if the size of the arrays is $\log n$ or more and *is* the PS for s=1. If $s < log\ n$ but $s = O(log\ n)$ the SCA is still an AT^2 optimal architecture for function like odd-even merge, FFT etc. Furthermore the SCA seems to be the only architecture that is the PS *and* the CCC.

2. the P_kS can emulate the MOT for $k = \sqrt{n}$ without significant degradation of time complexity, *is* the PS for $k = 2$, and *is* the binary tree for $k = n$.

Hence the paradigm naturally running on the P_kS, that is the Generalized Recursive Combination (GRC), efficiently support the paradigms of all the above mentioned network.

3. all the algorithms can be easily described with an algol-like high level language.

In the following we will call PS or SEN the architecture defined in [1]. The binary tree will have two kind of processors: the Leaves and the Internal Nodes with different purposes. In fact while the internal nodes can perform only very simple operations like masking and logic operations, we consider the Leaves to have the same computational capabilities as a microprocessor. All the wires allow data exchange and the transmission time is not dependent on the wire length (constant delay model of computation). (for a complete discussion about the computation models see [6,7,8,9,20]).

2 The Shuffle-Connected Arrays

In this section $n = 2^d$, $s = 2^r$ and $N = n/s$. If $V[i], i = 0, 1, ..., N-1$ are N processors, the the Transfer Edges (TE) and the Shuffle Edges (SE) of the Shuffle Exchange Network are defined as follows:

$$TE = \{(V[l], V[2l \ mod(N-1)]), l = 0, 1, ..., N-2\}$$
$$\cup\{(v[N-1], v[N-1])\}$$
$$EE = \left\{(V[2l], V[2l+1]), l = 0, 1, ..., \frac{N}{2} - 1\right\}$$

If we replace each $V[i], i = 0, 1, ..., N-1$ with a linear array of $s = 2^r$ elementary processors $v[i, j], j = 0, 1, ..., s-1$, the Shuffle-Connected Array (SCA) is defined as follows:

$$TE = \{(v[l, 0], v[2l \ mod(N-1), 0]), l = 0, 1, ..., N-1\}$$
$$\cup\{(v[l, m], v[l, m+1]), m = 0, 1, ..., s-2; l = 0, 1, ..., N-1\}$$
$$EE = \{(v[2l, 0], v[2l+1, 0]), l = 0, 1, ...N-1\}$$

The network for N=32 and s=4 is shown in fig. 2. The SCA computation paradigm is a sequence of $\log N$ (OPER; SHUFFLE) steps and s input steps. An (OPER; SHUFFLE) step is:

---

procedure $OPER(N, j)$
begin

```
        foreach l : (0 ≤ l ≤ N/2 − 1)
        pardo
                oper (v[2l, 0], v[2l + 1, 0], l, j)
        odpar
end
procedure SHUFFLE(N)
begin
        foreach l : (0 ≤ l ≤ N − 1)
        pardo
                v[l, 0] → v[2l mod(N − 1), 0]
        odpar
end
```

where the operation $\rightarrow$ is a simple content transfer and the $oper(A, B, l, j)$ is the function computed by the couple of processors A and B; it may also depend on l and on the computation step j. The parallel Algol-like statement: *foreach var* : $P(var)$ *pardo "body" odpar* means that *"body"* must be performed in parallel on all the values of *var* such that $P(var)$ is true. The final paradigm, shown in table 1, needs the input procedure that is a sequence of transfer operation through the size s arrays.

Each step of the MAIN loop has time complexity $O(log\ N)$, the SHIFT step has time complexity $O(s)$. The resulting time complexity is $O(log\, N + s)$ for problems that need only a constant number of complete steps (i.e. FFT, odd-even merge, etc.) assuming that $oper(A, B, l, j)$ can be performed in time $O(1)$. It is possible to prove that the layout area of the circuit is $A = O\left(N^2/s^2\right)$ starting from the optimal layout for the PS with N processors, hence the network achieves an optimal AT^2 complexity for that class of problems.

Before introducing other architectures we would like to mention a natural composition of SCAs. If we consider s SCAs whose $i - th$ arrays $i = 0, 1, ..., N$ are composed as a square mesh of side s, we obtain an architecture we call Shuffle-Connected Meshes (SCM). It has similar properties of the SCA and exhibits $AT^2 = O(n^2)$ for T in the range $[O(log\, n), O(\sqrt{n})]$ and s in the range $[1, O(\sqrt{n})]$.

An other important measure that particularly fits with pipelined architectures is the period P [8]. In fact the use of the SCA in a pipelined mode is strongly affected by the period $P = O(p(log\, n))$ where $p(log\, n)$ is a generic polynomial in log n. As well as the first data wave enters the last processors column, i.e. the column operated by the PS, new data cannot be accepted before the processing is finished. Assuming that the processing time for each wave is $O(log\ n)$, the above defined SCA will process $O(log\ n)$ data waves in $O\left(log^2 n\right)$ time when running in pipelined mode.

A known alternative presentation of the PS [5] is shown in fig.1. It has log n stages and is suitable for pipelined processing. Its period is $O(1)$ and as well as

$s \geq log\ n$ this network can be embedded on the last log n columns of the SCA substituting the one stage PS.

Let us define also the Partial Perfect Shuffle (PPS[j]) as a j-stage ($j <$ log N) Perfect Shuffle (fig.3). If a data pattern straightly ran on the circuit only the first j steps of the perfect shuffle would be performed. If we connect by means of a PS the $(j-1)$st and the first stages and let the data cycle in the network, all the steps of the SEN can be computed in $O(log\ N)$. The period is $O(1)$ for sets of j waves, i.e. the first set of j waves enter the network with period 1, the second set of j waves must wait $O(log\, N/j)$ steps to assure a perfect behaviour.

The pipelined Shuffle Connected Arrays is then defined as follows:

$$TE = \{(v[l,p], v[2l\ mod(N-1), p-1\ mod(s)) : 0 \leq p < s\ and\ 0 \leq p < log\, n\}$$
$$\cup \{(v[l,p], v[l,p-1]) : log\, n < p < s\}$$

$$EE = \left\{(v[2l,p], v[2l+1,p]) : 0 \leq l \leq \frac{N}{2}\ and\ 0 \leq p < s\ and\ 0 \leq p < log\, n\right\}$$

The (OPER; SHUFFLE) step for this network will be:

```
- - ---- - ----- --
procedure OPER(n, s, j)
begin
        foreachl : (0 ≤ l < N/2 − 1)
        foreachp : (0 ≤ p < log N) and (p < s)
        pardo
                oper (v[2l, p], v[2l + 1, p], l, j)
        odpar
end
procedure SHUFFLE(n, s)
begin
        foreach l : (0 ≤ l < N − 1)
        pardo
                foreach p : (0 < p < log N) and (p < s)
                pardo
                        v[l, p] → v[2l mod (N − 1), p − 1 mod(s)]
                odpar
                foreach p : (log N ≤ p < s)
                pardo
                        v[l, p] → v[l, p − 1]
                odpar
        odpar
end
```

Indeed it is possible to prove the following

Claim 1

The pipelined SCA with $n = 2^d$ processors and array size $s = 2^r$ has optimal layout for $s = n/h$ for a constant $h > 1$, $AT^2 = O(n^2)$ for s in the range $[O(\log n), \sqrt{n}]$ and $AT = O(n\sqrt{n})$ when $s = \sqrt{n}$.

Proof:

If $s = n/h$ then the SCA is composed of $O(1)$ arrays of size $O(n)$. Each array can be laid out on area $O(n)$ and the PS on a constant number of processors only needs an area $O(1)$.

If $\log n \leq s \leq O(\sqrt{n})$ than the SCA is composed by $O(n/s)$ arrays of size s. The area to laid out the arrays is $O(n)$ and the PS on the last $\log(n/s)$ columns of the $O(n/s)$ arrays is laid out on area $O(n^2/s^2)$ in a direct way. In fact since the wires' paths for the TE are $O(n/s)$ and are all distinct, we can organize them as shown in fig.8. The total area will be $O(n^2/s^2)$. The time will be $O(s)$ so that $AT^2 = O(n^2)$. In the case of $s = \sqrt{n}$ $AT = O(n\sqrt{n})$ that is optimal [6,8] for functions like odd-even merge, FFT, etc.

The last case is when $s < \log n$. We cannot directly embed a PS on the SCA because we do not have a sufficient number of columns, but it is possible to use the PPS[s]. If $s = O(\log n)$ the total area will be $O(n)$ for the arrays and $O(n^2/s^2)$ for the PS between the last and the first column. Since the time complexity is $O(s)$ also in this case the circuit has an $AT^2 = O(n^2)$. Δ

It is interesting to note that when s is in the range $[\log n, O(\sqrt{n})]$ the SCA has the identical topology of the CCC. If we assume (as we stated in the introduction) that all the wires can support data exchange also the computation paradigm is the same. The obvious implication is the complete equivalence between the Ascend-Descend and the pipelined (OPER;SHUFFLE) classes.

3 The Generalized Recursive Combination

The Recursive Combination (RC) played a very important role in the VLSI design of broad and general purpose architectures. Directly obtained by the well known *divide and conquer* sequential technique, it presents the following attractive properties:

1. The problems that can be solved with a *divide and conquer* technique are of interest and in a great number,[12]
2. The natural parallel architecture supporting it is the n-cube, a clear and symmetric architecture. [1,3]

3. The description of the algorithms can be easily done by the means of an *algol-like* high level language.[3,14]

Let the problem with the input in an array T[0:n-1] and the output in the same array be solved by means of a call to the procedure RC(0,n-1) where RC is:

---

procedure $RC\,(l, r-1)$
begin
$RC\left(l, l+\frac{r-l}{2}-1\right)$
$RC\left(l+\frac{r-l}{2}, r\right)$
$foreach\ j : \left(0 \le j \le \frac{r-l}{2}-1\right)$
pardo
$oper(l+j, l+\frac{r-l}{2}+j)$
odpar
end

Problems like FFT, bitonic merge and convolution admit algorithms that have the form of RC. More complex combinations of it can be used for problems like sorting, calculation of symmetric functions, graphs problems etc.

In this section the Generalized Recursive Combination paradigm (GRC) is discussed. If T[0:n-1] is the I/O array the GRC can be described by the following procedure:

---

procedure $GRC(l, r-1)$
begin
$GRC(l, l+\frac{r-l}{k}-1)$
$GRC(l+\frac{r-l}{k}, l+\frac{2(r-l)}{k}-1)$
....
$GRC(l+\frac{(k-1)(r-l)}{k}, r-1)$
$foreach\ j = 0, \frac{r-l}{k}-1$
pardo
$oper(l+j, l+\frac{r-l}{k}+j, ..., l+\frac{(k-1)(r-l)}{k}+j)$
odpar
end

the oper statement represents the parallel operation on k elements. The natural architecture which support this paradigm is the recursive network shown

in fig.5. The graph connecting the horizontal nodes in the figure is called the $k-edge$ and it is used for the *oper* statement. It is easy to verify that GRC has time complexity $O(T_k log_k\, n)$ on such a network where T_k is the time to perform a single *oper* statement.

Suppose that the n node indexes are represented in radix k. Let us define the $k-edge$ as the graph involving the k nodes satisfying the property that one and only one $p : 0 \leq p \leq (log_k n) - 1$ exists such that the node indexes representation differs only in position p. Thus the topology of the multidimensional $k-ary$ cube can be recognized. In fig.5 the graph for $n = 9$ and $k = 3$ is shown. Anyway also if the k-cube has these important computational properties it cannot be directly used for a VLSI implementation mainly because the number of links for each node is proportional to $log_k n$. The emulation of the k-cube will be discussed in the next section.

4 The Perfect k-Shuffle

In the following $n = k^d$, $s = k^r$, $N = n/s$. Let $V[j]$ denote the vertex where $T[j]$ is at the starting time $(0 \leq j \leq n = k^d)$. The Transfer and the Exchange edges of the P_kS are:

$$TE = \left\{ \left(V[j], V[kj \ mod(k^d - 1]\right), j = 0, 1, .., k^d - 1 \right\}$$

$$EE = \{^n k^{d-1} \ Binary \ Trees \ with \ the \ k \ leaves \\ (k(i-1), ki-1)^n i = 1, 2, ..., k^{d-1}\}$$

In fig.6 the P_kS is shown for $n = 16$ and $k = 4$.

Claim 2

The P_kS on $n = k^d$ processors supports GRC in $O(T_k log_k n)$ where T_k is the computation time of the *oper* statement on a binary tree.

Let us prove the following

Lemma 1:

At step i $(0 \leq i \leq d-1)$ the k leaves of each tree in a P_kS will contain data whose distance in the starting position is K^{d-i}.

Proof of the lemma

If all indexes $j (j = 0, 1, ..., k^{d-1})$ are expressed in *radix k* notation, they can be considered of the form $j = Ul$ where $U = u_{d-1}u_{d-2}...u_1$ and l is a $k-ary$ digit. A tree will be called the tree U $(U = 0, 1, ..., k^{d-1})$ if its leaves have indices Ul for each l. It is easy to verify [11] that the data in position $j_i = u_{d-1}u_{d-2}...u_1 l$ at step i will be transferred to position $j_{i+1} = u_{d-2}...u_1 l u_{d-1}$ at step i+1. But if at step i the data is in position $j_i = Ul$, at step $i-1$ it was in position $j_{i-1} = lU$ and at step 0 in position $j_0 = u_i u_{i-1}...u_1 l u_{d-1}...u_{i+1}$. Since at step

i data in position Ul $(l = 0, 1, ..., k-1)$ are in the same tree, the lemma is proved.

Proof of the claim

As it is possible to see by simple inspection the recursive calls of GRC are performed on data whose distance depends on the steps in the same way as in the lemma. If the tree can perform the OPER statement in time T_k than GRC can run in $O(T_k d)$ steps on the $P_k S$.$\triangle$

The Claim 3 suggests how the GRC can be implemented on the $P_k S$. The algorithm to emulate the GRC on the $P_k S$ is:

```
- - - - - - - - - - - -
procedure P_kS(k, d)
begin
        for i = 0 to d - 1
        do
                k - SHUFFLE
                OPER
        od
end
```

Where the procedure $OPER$ and $k - SHUFFLE$ are:

```
- - - - - - - -
procedure OPER(k, d, j)
begin
        foreach l : (0 ≤ l ≤ k^(d-1) - 1)
        pardo
                oper (v[kl], v[kl + 1], ..., v[kl + k - 1], l, j)
        odpar
end
procedure k - SHUFFLE(k, d)
begin
        foreach l : (0 ≤ l ≤ k^d - 1)
        pardo
                v[l] → v[klmod(k^d - 1)]
        odpar
end
```

In this way the $P_k S$ is a k-cube emulator in $O(d)$ steps.

The value of k directly affects the P_kS performances and changes its global behaviour. In fact it is possible to state a correspondence between k and the kind of architecture the P_kS can emulate. The most relevant cases are enumerated in the following:

1. when $k = 2$ the procedure k-SHUFFLE and the procedure OPER have the same behaviour of the SHUFFLE and OPER of the PS without any change in the time complexity.
2. when $k = \sqrt{n}$, $d = 2$ and the P_kS procedure run in $O(T_k)$ time. From the lemma 1 at step 0 in the tree U there are data whose distance is 1 and at step 1 in the same tree there will be data whose distance is k. In two steps the procedure P_kS emulates the adjacences of the mesh of trees. In fig.7 all the process is shown. If Oper is a generic operation on the binary tree, the MOT can perform it in parallel on the row trees and on the column trees with time complexity $O(T_k)$. On the P_kS the same processing can be performed with the same time performances by $P_kS(\sqrt{n}, 2)$.
3. when $k = \sqrt[m]{n}$, $d = m$ and the procedure P_kS runs with $O(T_k m)$ time complexity. In the significant case of m=3 the P_kS can emulate in 3 steps the three dimensional orthogonal trees in a way completely equivalent to the case of m=2.
4. when $k = n$, $d = 1$ and the procedure P_kS will perform the only OPER statement on the binary tree with n leaves.

5 The layout for the P_kS

Attempts in laying out the PS have interested computer scientists for a long time. Thompson in his thesis showed that any layout for an n-nodes PS requires $A = O(n^2/log^2 n)$ and proposed an $A = O(n^2/\sqrt{log\ n})$ layout. Hoey and Leiserson found a $A = O(n^2/log\ n)$ layout using an interesting technique [18]. Rodeh and Steinberg [19] lowered this upper bound to $A = O(n^2/log^{3/2} n)$ and finally Kleitman, Leigthon, Lepley and Miller discovered the layout achieving the lower bound [2]. In this section we show how to layout the P_kS in area $A = O(n^2\ log\ k/k\ log_k n)$.

Let us recall some notations:
In a P_kS the nodes are represented as numbers in k-ary notation. The shuffle operation is defined as the left rotation of the digits and the *necklace* is an equivalence class defined on the set of indexes $[0, k^{d-1}]$ where the equivalence relation is: two indexes are equivalent if it is possible to transform the first in the second by repeated application of the shuffle operation on its k-ary representation. The *length* af a necklace is the cardinality of the class of equivalence.

It is intuitive that the maximum length is $log_k n$. The following theorem, (known as the Fermat little theorem), holds:

Theorem 2
If $n = k^d$ and d prime, $\frac{k^d-k}{d}$ is an integer.

A corollary of the theorem 2 is that if $n = k^d$ and d is prime the number of necklaces is $O(n/log\, n)$. In [18] it is shown that the corollary holds also if d is not prime.

If we organize the necklaces as vertical cycles of length $log_k n$ than we need at least a dimension $O(n/log\, n)$, i.e. the number of the cycles, and $O(\frac{n\, log_k n}{k})$ horizontal lines to embed the n/k trees, then the needed area is:

$$A = O\left(n^2 \frac{log\, k}{k\ log_k n}\right)$$

The time complexity for algorithms requiring only a constant number of complete loops is $T = O(T_k log_k n)$ than

$$AT^2 = O\left(\frac{n^2 log\, n}{k} T_k^2\right)$$

If $k = T_k^2$ than the Hoey Leiserson bound is reached; for $T_k = O(log\, k)$ the equation $k = log^2 k$ has two solutions for k=4 and k=16.

When $k = \sqrt{n}$ the area bound is not satisfactory. In fact it is possible to layout the $P_k S$ for $k = \sqrt{n}$ in $A = O(n\, log^2 n)$. In this case the TE perform a matrix transposition. Whence each processor is connected with its symmetric respect to the principal diagonal. If we simply fold the processors matrix joining the elements in symmetric positions the obtained layout will have area $O(n\, log^2 n)$.

Also for the $P_k S$ it is possible to define a pipelined version. It exhibits a time complexity $O(T_k log_k \frac{n}{s})$ where n is the processors number, s is the number of stages and T_k the processing time on the binary trees. With the same technique used for the SCA the $P_k S$ embedding has $A = O(\frac{n^2}{d^2} T_k log\, k)$.

6 Conclusion

Two architectures based on the shuffle network have been introduced. They span the performances in a wide range of time complexities and over different computational paradigms. The AT^2 complexity is optimal for the SCA and near optimal for the $P_k S$. The related computational paradigms can be easily adapted to emulate the most relevant broad purpose architectures as the CCC, the PS, the Mesh Of Trees. Parametrized architectures, i.e. architectures changing performances depending on a parameter, are important in defining a unified form for wider algorithms classes. In our case the algorithms running on the CCC, on the PS and on the MOT are shown to be instances of the same algorithm form: The $P_k S$ paradigm.

Acknowledgements

The author would like to greatfully acknowledge the referees suggestions and comments that permitted a substantially improved version af this work. A thanks also to Prof. Paul Cull for his helpful comments and technical assistence.

REFERENCES

1. Stone H.S.,Parallel Processing with the Perfect Shuffle, IEEE trans. on Comp.,C-20,2, febr.1971 pp.153-161.

2. Kleitman D., Leighton F.T., Lepley M., Miller, New layouts for the Shuffle-exchange graph", Proceedings of the 13th Ann. Symp. on Th. of Comp., Milwaukee, Wisconsin 1981.

3. Preparata F.P. and Vuillemin J., The cube-connected-cycles: a versatile network for parallel computation. Comm. of ACM, vol.24, pp.300-309, May 1981.

4. D. Lawrie, "Access and Alignment of Data in an Array Processor," IEEE Trans on Comp., C-24, 12, 1145-1155 (1975)

5. Batcher J.L., Sorting network and their applications, Proc. AFIPS SJCC, vol 32,Apr.1968, pp. 307-314.

6. Thompson, C.D. Area-Time complexity for VLSI. Proc. 11th Annual Symp. on Theory of Computing, Atlanta, GA, May 1979, 81-88.

7. Bilardi G., Pracchi M., Preparata F.P., A critique and appraisal of VLSI models of computation, in Proc. CMU conf. VLSI Syst. Comp., Oct. 1981. pp. 81-88

8. Vuillemin, J., A combinatorial limit to the computing power of VLSI circuits. Proc. 21st Symp. on Foundations of Computer Science, Syracuse, NY, Oct. 1980 294 300.

9. Preparata F.P. and Vuillemin J.E., Area-Time Optimal VLSI Network For Multiplying Matrices, Inf. Proc. Lett., vol 11, pp. 77-80, Oct. 1980.

10. Wu C., The Universality of the Shuffle Exchange Network, IEEE Trans on Comp., C-30, 5, 325-332 (1981).

11. Keutzer K. and Robertson E., the M-shuffle as an interconnection network for SIMD, Proc of 20th Annual Allerton Conference, Control and Computing, Monticello, IL, pp 264-271; Oct. 1982.

12. Knuth D., The art of computer programming, Vol. 1,2,3, Addison-Wesley, Reading, MA, 1973.

13. Aggarwal, On I/O placement in VLSI Circuits, Proc. 21st Annual Allerton Conference, pp 236-243, Oct. 1983.

14. Preparata F.P., Algorithm Design and VLSI Architecture, Proc. Symp. on Vector Processors and Scientific computation, Rome, March 1982.

15. Nath et al., Parallel Processing Based On Orthogonal Trees, IEEE trans. on comp. vol. c-32 no. 6, june 1983.

16. Muller D.E. and Preparata F.P., Bounds to complexities of networks for sorting and switching, J. ACM , vol 22, pp.195-201, Apr. 1975.

17. Leigthon F.T., New lower bound techniques for VLSI, Proc. 22nd Ann. IEEE Symp. Found. Comp. Sci., Oct. 1981, pp. 1-12.

18. Hoey D., Leiserson C.F., A layout for the Shuffle-Exchange Network, Proc. 1980 Int. Conf. on Parallel Processing.

19. Rodeh M. and Steinberg D., A layout for the Shuffle-Exchange Network with $O(n^2/log^{3/2}n)$ area. IEEE trans. on comp. vol. c-30 no.12 p.977.

20. Brent R.P., Kung H.T., The chip complexity of binary arithmetic, Proc. 12th Annu. ACM Symp. on Th. of Comp., May 1980, pp. 190-200.

21. Bilardi G., The Area-Time Complexity of Sorting, Ph.D. Dissertation, Dep. of Computer Science, University of Illinois at Urbana.

```
{SHIFT}
for j := 0 to s − 2
do
        foreach p : 0 < p ≤ s − 1
        pardo
                foreach q : 0 ≤ q < N − 1
                pardo
                        v[q, p − 1] ← v[q, p]
                odpar
        odpar
od
{MAIN}
forever
loop
        for j := 0 to (logN − 1) step 1
        do
                foreach l : (0 ≤ l < N/2 − 1)
                pardo
                        oper(v[2l, 0], v[2l + 1, 0], l, j)
                odpar
                foreach l : (0 ≤ l < N − 1)
                pardo
                        v[l, 0]− > v[2lmod(N − 1), 0]
                odpar .
        od
pool
```

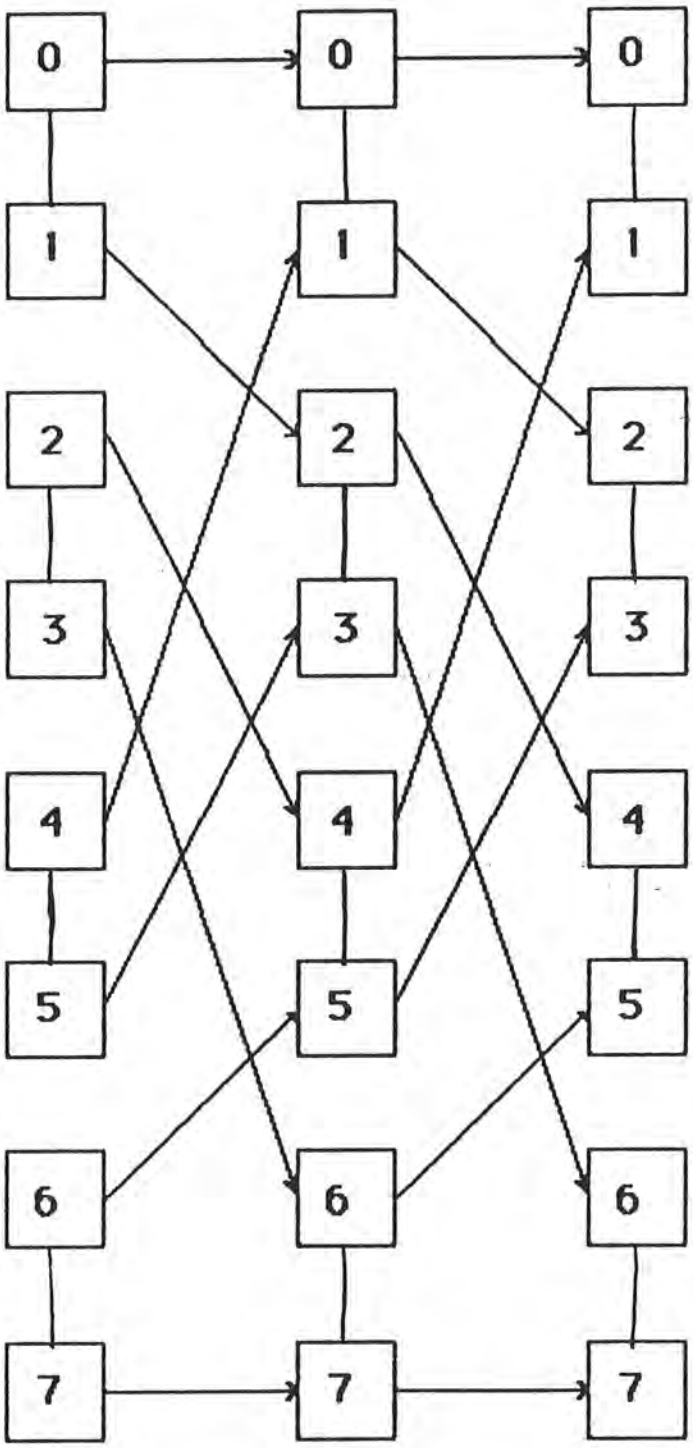
table 1

```
{MAIN}
forever
loop
        foreach p : 0 < p ≤ s − 1
        pardo
                foreach q : 0 ≤ q < N − 1
                pardo
                        v[q, p − 1] ← v[q, p]
                odpar
        odpar
        for j := 0 to (log N − 1) step 1
        do
                foreach l : (0 ≤ l < N/2 − 1)
```

```
            pardo
                    oper(v[2l, 0], v[2l + 1, 0], l, j)
            odpar
            foreach l : (0 ≤ l < N − 1)
            pardo
                    v[l, 0] ← v[2lmod(N − 1), 0]
            odpar
    od
pool
{TRANSFER EDGES COMPUTATION}
```

table 2

Fig. 1: the shuffle for $n = 8$



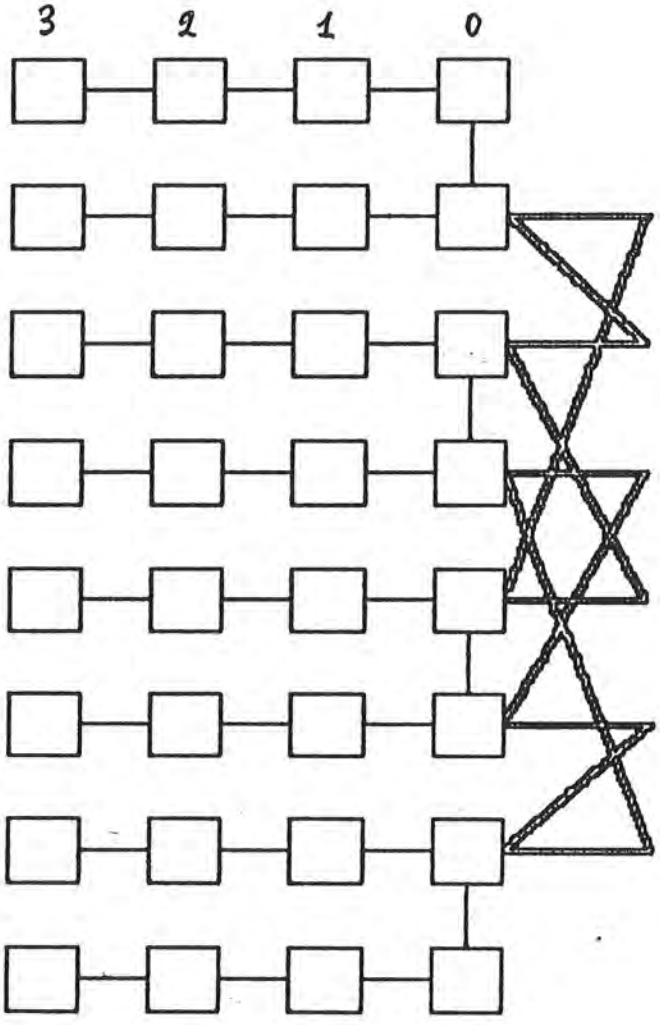


Fig. 1: The 3CA for $n = 3$ and $s = 4$.

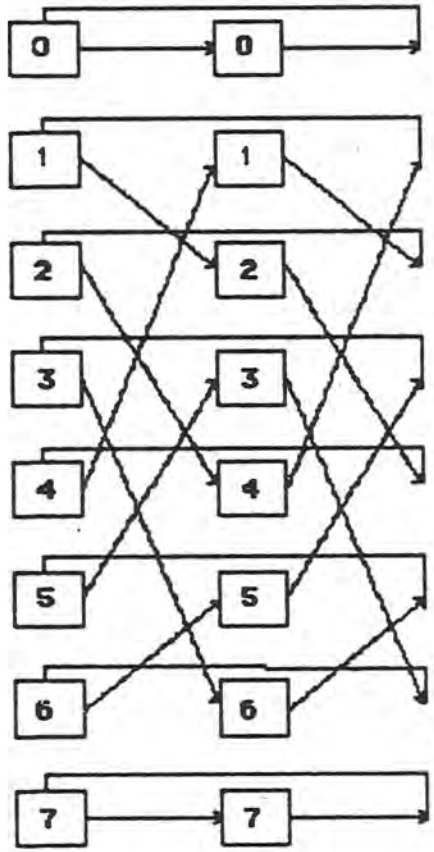


Fig 3: the PPS[z] with N = 8.

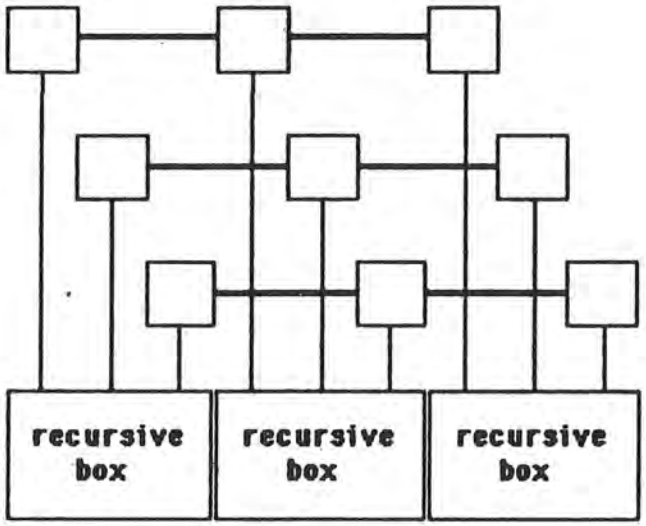


Fig. 4. Recursive Network for the GRC paradigm when $K = 3$

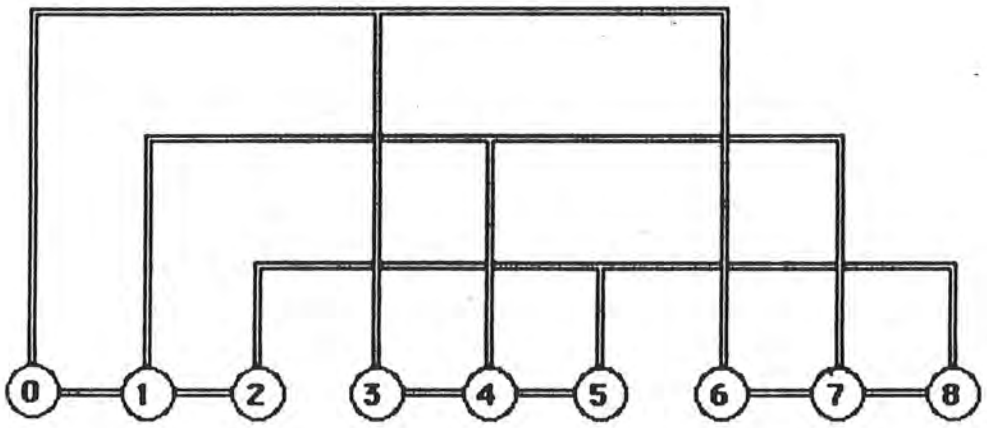


Fig. 5. the K-cube for $n = 9$ and $K = 3$

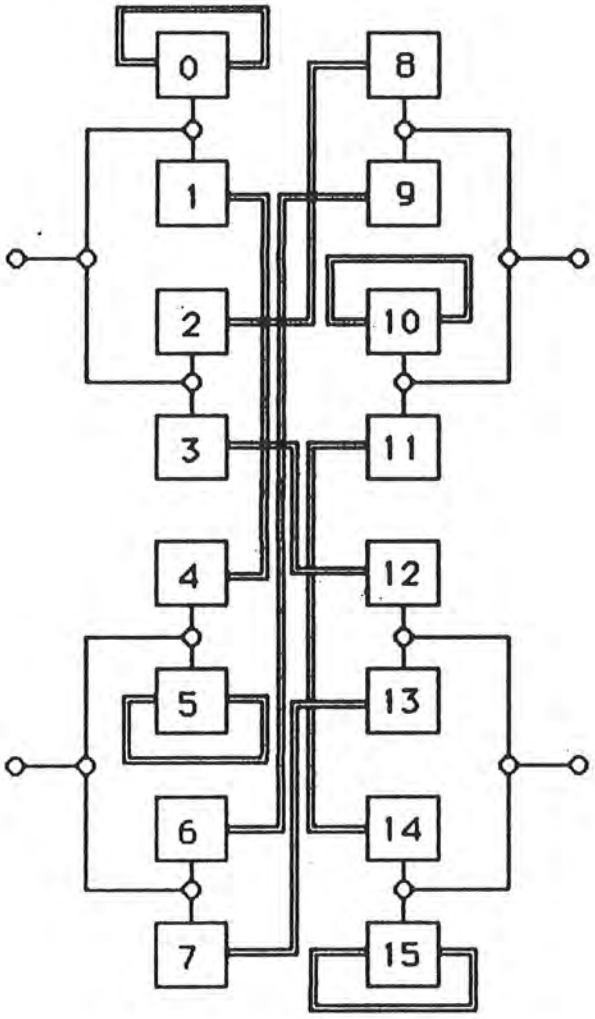
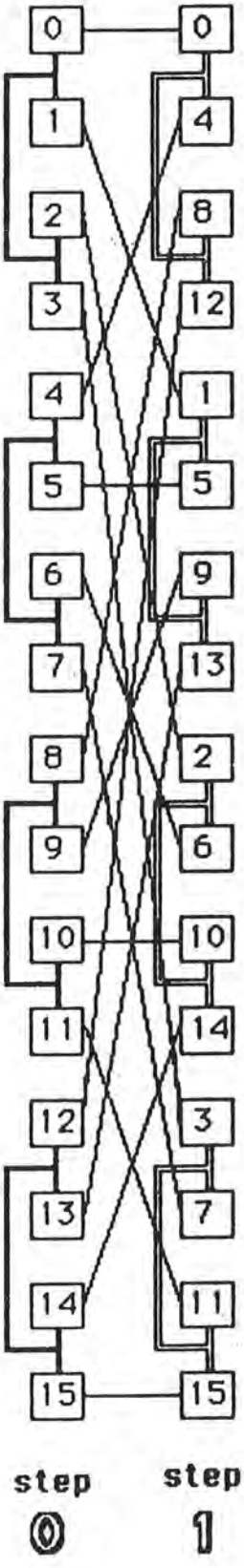


Fig. 6. the P_kS for $n = 16$ and $K = 4$.



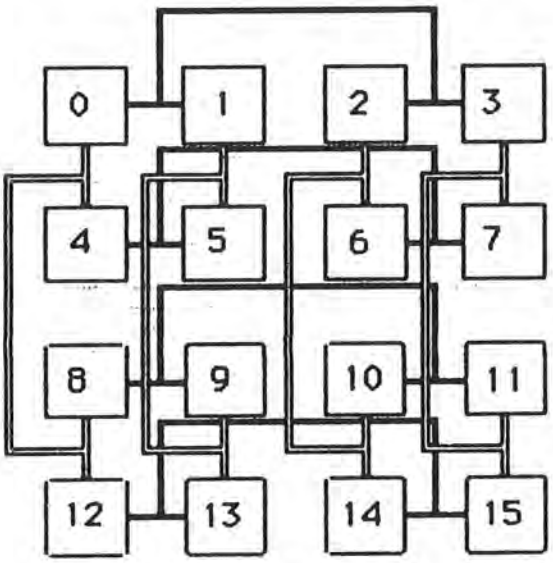


Fig. 7: the emulation of the MOT: at step 0 the processing on the rows is performed (bold lines), at step 1 the processing on the columns (double lines).

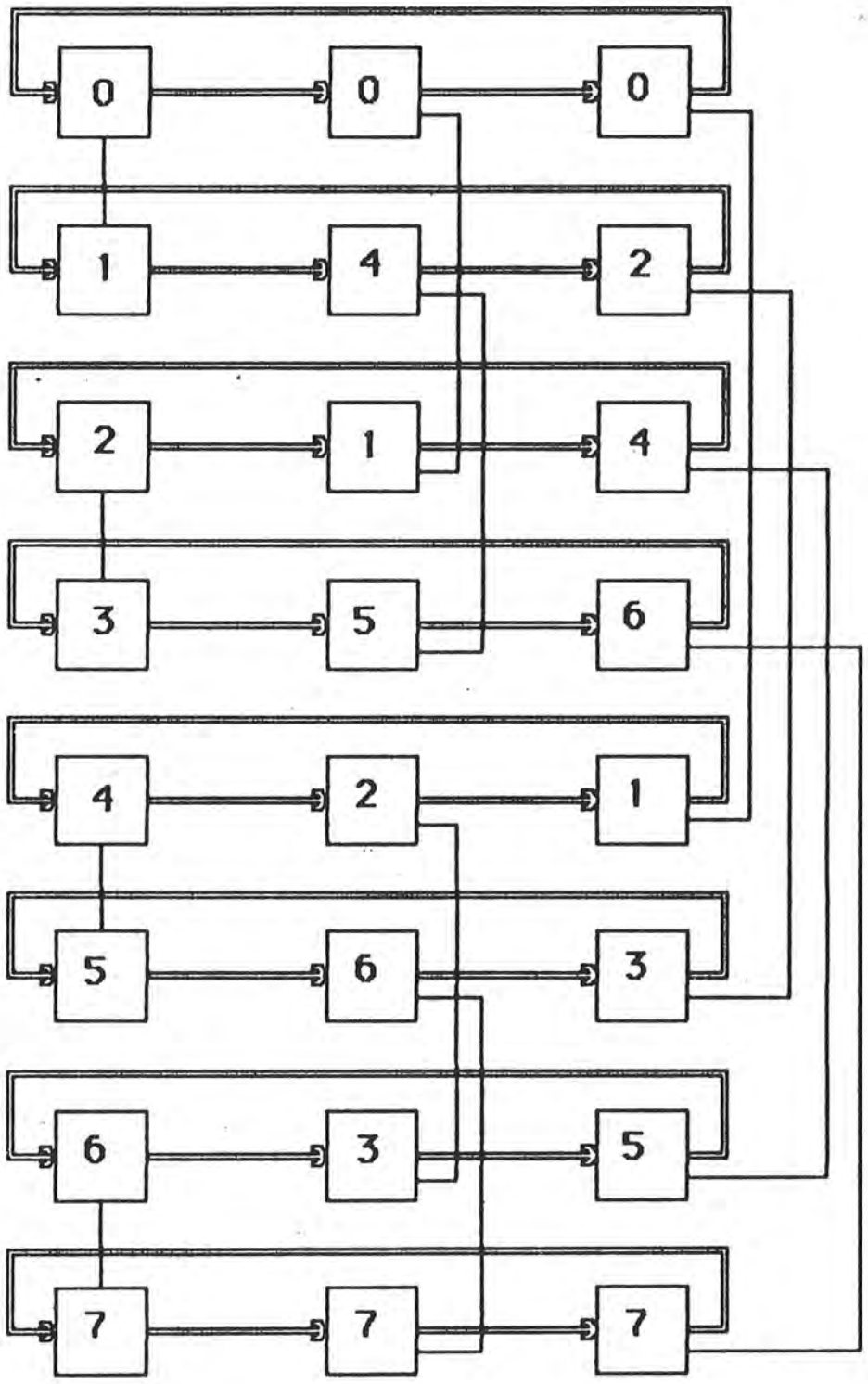


Fig. 8. A layout for the SCA.