REVUE FRANÇAISE D'AUTOMATIQUE, D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE. RECHERCHE OPÉRATIONNELLE

AMITAVA DUTTA
HOWARD J. SIEGEL
ANDREW B. WHINSTON

# On the application of parallel architectures to a class of operations research problems

*Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, tome 17, n° 4 (1983), p. 317-341.

<http://www.numdam.org/item?id=RO_1983__17_4_317_0>

R.A.I.R.O. Recherche opérationnelle/Operations Research
(vol. 17, n° 4, novembre 1983, p. 317 à 341)

# ON THE APPLICATION OF PARALLEL ARCHITECTURES TO A CLASS OF OPERATIONS RESEARCH PROBLEMS (*)

by Amitava DUTTA (1), Howard J. SIEGEL (2) and Andrew B. WHINSTON (3) (+)

Abstract. — *There has been considerable interest in developing efficient algorithms for Operations Research problems. While the efficiency of the algorithm is an important variable in the efficient use of the algorithm, another equally important variable, in our opinion, is the machine architecture on which the algorithm is implemented. Benefits of special architectures have already been recognized in other functional areas. More importantly, these architectures are now technologically feasible. In this paper, a special architecture is developed for a class of Operations Research problems. The objective is to demonstrate the feasibility and potential benefits of such an endeavor, rather than presentation of a fully detailed design for evaluation. To this end, the building blocks used to design the system are well established ones. Estimates of improvement in performance over traditional uniprocessor systems are presented. Close interaction between specialists in the area of Operations Research and computer architecture will be required for efficient implementation of such architectures.*

Keywords: Parallel Processing; Decomposition Algorithms; Interconnection Network; and SIMD Machine.

Résumé. — *On s'est toujours attaché à développer des algorithmes efficaces pour la Recherche Opérationnelle. L'efficacité d'un algorithme est, bien entendu, un critère important; il est un autre critère qui est, à notre avis, aussi important que le précédent : il s'agit de l'architecture de la machine sur laquelle l'algorithme est implanté. Les avantages dus à des architectures spéciales ont déjà été reconnus dans d'autres domaines. Plus important encore, ces architectures sont maintenant techniquement réalisables. Dans cet article, nous développons une architecture spéciale pour une classe de problèmes de Recherche Opérationnelle. Notre objectif est de démontrer la faisabilité et les avantages d'une telle entreprise plutôt que de présenter un projet détaillé en vue d'une évaluation. A cette fin, les blocs utilisés pour construire le système sont des blocs bien établis. Nous présentons des estimations de l'amélioration des performances par rapport à des monoprocesseurs traditionnels. L'implémentation efficace de ce type d'architecture requerra une étroite collaboration entre spécialistes de la Recherche Opérationnelle et spécialistes de l'architecture des ordinateurs.*

## 1. INTRODUCTION

The discipline of Operations Research has long been concerned with the development of various mathematical models and efficient algorithms for their satisfactory solution. A substantial number of these algorithms are computationally feasible today simply because of the availability of high speed digital computers. For example, a lot of algorithms that employ branch and bound techniques, or dynamic programming techniques would be practically unusable were it not for digital computers. An important aspect of the efficient implementation of such algorithms is the efficiency of the algorithm itself. This

(*) Received in May 1982.
(1) The Graduate School of Management, The University of Rochester, Rochester, N.Y. 14627.
(2) Electrical Engineering School, Purdue University, West Lafayette, IN 47907.
(3) Krannert Graduate School of Management, Purdue University, West Lafayette, IN 47907.
(+) This work was supported in part by the National. Science Foundation under Grant No. ECS-812089 *b*, Grant No. IST-8108519 and Grant No. ECS-8116135, and a gift from The IBM Corporation to The Management Information Research Center.

R.A.I.R.O. Recherche opérationnelle/Operations Research, 0399-0559/1983/317/$ 5.00

concern is amply evident in the continuous efforts of researchers to develop better algorithms. An equally important aspect, in our opinion, is the architecture of the machine on which the algorithm is implemented. While the general purpose machine (uniprocessor) may be capable of executing the algorithm, alternative architectures may offer substantial improvements in performance. This fact has already been recognized in certain application areas, as for example, in the area of image processing [16, 22] and missile tracking [6].

There appears to be little work done on special architectures suited to operations research problems. We hope to call attention to the interesting possibilities offered by special architectures by developing one for a class of operations research problems. Before proceeding with the development, a few words regarding the general objectives of the paper are in order.

Our immediate objective is to expose the benefits of a special architecture to a class of operations research problems. The long term objective of course is to stimulate the development of special architectures for other classes of problems in the field. No attempt has been made to "fine tune" the architecture to exploit intricate properties of problems in the class we have investigated. Such an effort needs considerable interaction between hardware specialists and experts in the operations research area. Thus, in reading the paper, it should be kept in mind that we are not presenting a fully developed system for evaluation. However, the potential for special architectures in this area should strike the reader. Refinements to the system will be the subject of further work.

## 2. THE LINEAR PROGRAMMING PROBLEM

We have chosen probably one of the simplest and most widely used models in operations research – namely linear programming (LP). It has been in use for a long time and there are well established techniques, most notably the Simplex method, for its solution. Since the LP problem is being used only as an example of a target applications area that could profit from special architectures, the discussion in this section is tutorial in nature. Any elaborate treatment of the LP problem here would probably be of interest only to linear programming specialists. The basic problem may be stated as:

$$\text{maximize } c_1 x_1 + c_2 x_2 + , \ldots , + c_n x_n ,$$

$$\begin{aligned} \text{subject to } & a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \leqq b_1 , \\ & a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n \leqq b_2 , \\ & \vdots \\ & a_{m1} x_1 + a_{m2} x_2 + , \ldots , + a_{mn} x_n \leqq b_m , \\ & x_1 , x_2 , \ldots , x_n \geqq 0 . \end{aligned}$$

Using the more compact vector and matrix notations, the problem may be written as:

$$\text{maximize } \mathbf{c}'\mathbf{x},$$
$$\text{subject to } \tilde{A}\mathbf{x} \leqq \mathbf{b},$$
$$\mathbf{x} \geqq \mathbf{0},$$

where:

$$\mathbf{c}' = \{c_1, c_2, \ldots, c_n\}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \tilde{A} = \begin{bmatrix} a_{11}\ a_{12} \ldots a_{1n} \\ \vdots \qquad\quad \vdots \\ a_{m1}\ a_{m2} \ldots a_{mn} \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$$

The simplex method for solving this problem is well known and will not be repeated here. For details *see* [8]. A program for executing the simplex method on a parallel processor will be developed later in the paper. The marked improvement in solution time will then be readily seen.

Consider a class of LP problems that exhibit a special structure.

The practical problems in this class are typically very large, i.e., a large number of variables, and a large number of constraints. The constraint matrix has a few, so called, coupling constraints. The rest of the matrix displays a block angular structure.

The problem assumes the following form:

maximize $c_1 x_1 + c_2 x_2 + , \ldots, + c_n x_n$,

subject to

$$\boxed{\begin{matrix} a_{11}x_1 + & & +a_{1n}x_n \\ \vdots & & \vdots \\ a_{p1}x_1 & & +a_{pn}x_n \end{matrix}} \leqq \left.\begin{bmatrix} b_1 \\ \vdots \\ b_p \end{bmatrix}\right\} \begin{matrix} \text{Few} \\ \text{coupling} \\ \text{constraints} \end{matrix}$$

$$\boxed{B_1 \hat{x}_1} \leqq \begin{bmatrix} b_{p+1} \\ \vdots \\ b_q \end{bmatrix}$$

$$\boxed{B_2 \hat{x}_2} \leqq \begin{bmatrix} b_{q+1} \\ \vdots \\ b_t \end{bmatrix}$$

$$\boxed{B_3 \hat{x}_3} \leqq \begin{bmatrix} b_{t+1} \\ \vdots \\ b_s \end{bmatrix}$$

$$\boxed{B_4 \hat{x}_4} \leqq \begin{bmatrix} b_{s+1} \\ \vdots \\ b_m \end{bmatrix}$$

Partitioning the vectors **c**, **x**, **b**, and the coupling constraints appropriately, the above problem can be rewritten in general, in matrix notation as follows:

$$\text{maximize } \hat{c}_1\hat{x}_1+\hat{c}_2\hat{x}_2+\hat{c}_3\hat{x}_3+\ \ldots\ +\hat{c}_p\hat{x}_p,$$

$$\begin{aligned}
\text{subject to } & A_1\hat{x}_1+A_2\hat{x}_2+A_3\hat{x}_3+\ \ldots\ +A_p\hat{x}_p \leqq \hat{b}_0,\\
& B_1\hat{x}_1 \leqq \hat{b}_1,\\
& B_2\hat{x}_2 \leqq \hat{b}_2,\\
& B_3\hat{x}_3 \leqq \hat{b}_3,\\
& \qquad\qquad \vdots\\
& B_p\hat{x}_p \leqq \hat{b}_p\\
& \hat{x}_1, \hat{x}_2, \hat{x}_3, \ldots, \hat{x}_p \geqq \mathbf{0},
\end{aligned}$$

$$\mathbf{X}=\{\hat{x}_1, \hat{x}_2, \hat{x}_3, \ldots, \hat{x}_p\}, \qquad \mathbf{b}=\begin{bmatrix}\hat{b}_0\\ \vdots\\ \hat{b}_p\end{bmatrix}$$

Matrices $A_1, A_2, A_3, A_4$ all have the same number of rows, this number being equal to the number of coupling constraints. The number of columns in the matrices will not necessarily be the same, depending on how the variables have been partitioned to match the block angular structure. Of course, this notation can easily be extended to the case with $p$ blocks in the block angular structure.

At first glance, this structure appears to be special enough not to appear frequently in practice. However, many transportation problems have this form [29]. More importantly, this form of the LP problem arises when several departments are independently trying to optimize that portion of the objective function relevant to itself while using resources that are also needed by other departments. A central authority would like to impute certain prices to these shared resources so that the departments use the shared resources in quantities that maximize the total objective of the firm as a whole [10, 13].

The coupling constraints represent the constraint relations pertaining to shared resources. The matrices $B_i$ and vectors $b_i$, $1 \leqq i \leqq p$, represent constraints that affect only one particular department. The decision variables for the $i$-th department are contained in the vector $\hat{x}_i$.

This LP problem in block angular form can be solved by the well known Dantzig-Wolfe decomposition algorithm [9], and the reader is referred to [11], pp. 148-152 for mathematical details. Here, we choose to describe informally, the nature of the algorithm, to see how it could benefit from parallelism. The algorithm is two level with a "master program" at the second level and

subproblems at the first. Each subproblem is of the form (*see* preceding formulation):

$$\text{Minimize } (\hat{c}_i - \pi_1 A_i)\,\hat{x}_i.$$
$$\text{Such that } B_i \hat{x}_i = \hat{b}_i,$$

where $\pi_1$ is a vector of imputed prices handed down to each subproblem by the master problem. Each subproblem arrives at optimal values for its decision variables $x_i(\pi_1)$ and objective function $z_i^0$, and returns these values to the master problem. At this stage, the master problem checks appropriate criteria (again, *see* [11]) to see if has an optimal solution. Otherwise a new vector of imputed prices is computed and returned to the subproblems. The process repeats itself, and, if the master program is not degenerate, the decomposition principle will find the optimum in a finite number of iterations.

The subproblems to be solved are independent in that they do not need any information from each other. Hence their solution could logically proceed in parallel. This parallelism in itself would cause an improvement in the solution time. Each subproblem is an LP. It was mentioned sometime earlier that the ordinary LP problem could be solved on a parallel processor with improvement in solution time. Thus, in addition to solving the subproblems in parallel, each subproblem is solved on a parallel processor with attendant time savings. Thus the basic strategy is to solve the subproblems in parallel and parallelize the solution of each subproblem as much as possible. In the next section, a brief introduction to a class of parallel processors is presented. An architecture of a parallel processor which facilitates solution of the problem follows. Estimates of improvements in solution time are presented.

## 3. PARALLEL PROCESSING

A class of parallel processing machines is the SIMD (Single Instruction Stream Multiple Data Stream) machines [2, 4]. A general SIMD architecture is shown in Fig. 3.1.

The PE's stand for processing elements. There are $P > 1$ processing elements. Each PE has arithmetic and logical capabilities, $\text{Proc}_i$, and has its own memory $\text{Mem}_i$ (Processing Element Memory). All PE's are identical. The control unit (**CU**) drives the PE's in a synchronous fashion by feeding all PE's the same instruction at one time. The enabled PE's apply that instruction to their individual data streams. There are masking schemes to disable particular processors as and when necessary. The interconnection network allows PE's to

transfer data among themselves. Each PE transfers data through its data transfer register (DTR). Various interconnections networks have been proposed. For detailed analysis of their relative capabilities, *see* [17, 18, 19].

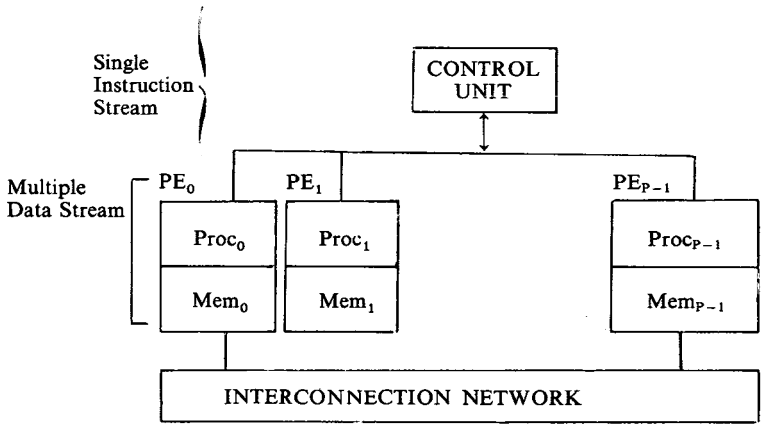


**Fig. 3.1. – Architecture of an SIMD machine**

The ILLIAC machine [2, 5] is an example of an SIMD machine. PEPE [7] is another. SIMD machines, in fact any parallel processor, involve considerable hardware and development costs. Adequate applications must be found to make these machines economically viable. PEPE is designed for missile tracking applications [6], and ILLIAC for weather forecasting. Architectures for image processing applications have been proposed in [16, 22].

### 3.1. Skewed storage

A technique for storing matrices in such SIMD machines that is of particular importance to us is the skewed storage technique. Consider an 8 by 8 matrix. In skewed storage format, the matrix would be stored as shown in the memory map of Fig. 3.2. In Fig. 3.2, each column is the memory associated with the PE marked above it. Each cell corresponds to a memory unit (word, byte, etc.).

| $PE_0$ | $PE_1$ | $PE_2$ | $PE_3$ | $PE_4$ | $PE_5$ | $PE_6$ | $PE_7$ | |
|---|---|---|---|---|---|---|---|---|
| 0,0 | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | 0,7 | $row_0$ |
| 1,7 | 1,0 | 1,1 | 1,2 | 1,3 | 1,4 | 1,5 | 1,6 | |
| 2,6 | 2,7 | 2,0 | 2,1 | 2,2 | 2,3 | 2,4 | 2,5 | |
| 3,5 | 3,6 | 3,7 | 3,0 | 3,1 | 3,2 | 3,3 | 3,4 | |
| 4,4 | 4,5 | 4,6 | 4,7 | 4,0 | 4,1 | 4,2 | 4,3 | |
| 5,3 | 5,4 | 5,5 | 5,6 | 5,7 | 5,0 | 5,1 | 5,2 | |
| 6,2 | 6,3 | 6,4 | 6,5 | 6,6 | 6,7 | 6,0 | 6,1 | |
| 7,1 | 7,2 | 7,3 | 7,4 | 7,5 | 7,6 | 7,7 | 7,0 | $row_7$ |

**Fig. 3.2. – An 8 by 8 Matrix in Skewed Storage.**

With this storage technique, any row and any column of the matrix can be loaded into the eight processors in parallel. Entry $(i, j)$ in a cell in Fig. 3.2 signifies that the value corresponding to row$_i$ and column$_j$ of the matrix is stored in that cell.

E. g., (i) To load row 3 (note that rows are numbered starting from zero), $PE_k$ loads cell 3 into its DTR (the memory cells of each PE are also numbered from zero to seven). Then each $PE_k$ transfers the contents of its DTR to $PE_{(k-3) \text{ Mod P}}$ where $P$ is the number of processors. This is a uniform inter PE shift and is discussed in more detail in section 6.4. In general, to load row $r$, each $PE_k$ loads its $r$-th cell and shifts its data to $PE_{(k-r) \text{ Mod P}}$.

(ii) To load column 4, $PE_k$ loads cell $(k+4)$ Mod P into its DTR. It then transfers the contents of its DTR to $PE_{(k-4) \text{ Mod } P}$. In general, to load column $c$, $PE_k$ loads cell $(k+c)$ Mod $P$ into its DTR and then transfers that data to $PE_{(k-c) \text{ Mod } P}$. For a general description *see* [26]. It is important to point out that the transfer from $PE_k$ to $PE_{(k-c) \text{ Mod } P}$ is done in parallel (simultaneously) for $k=0, 1, \ldots,$ P-1. This should become clear in section 6.4.

### 3.2. Recursive doubling

Another useful technique used often with SIMD machines is recursive doubling. Consider finding the minimum of $N$ numbers. On a uniprocessor, one would execute $N-1$ paired comparisons, retaining the smallest number at each stage and obtain the smallest of the $N$ numbers. Assuming each comparison

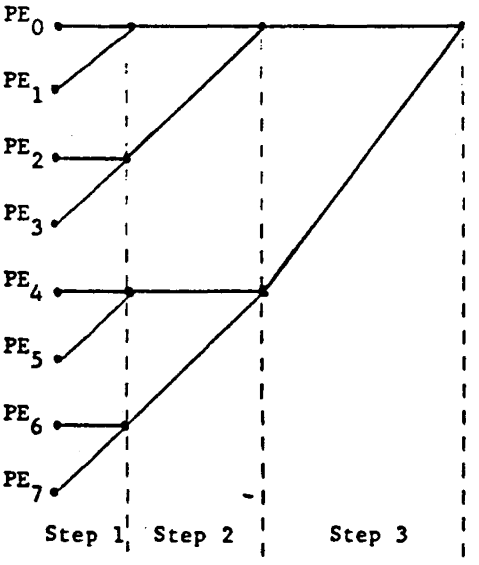


**Figure 3.3. – Recursive Doubling.**

takes one time unit, the sequential process takes $N-1$ time units. The recursive doubling can be best described by an example. Consider finding the minimum of 8 numbers. Figure 3.3 describes the process graphically.

In Step 1, the following pairs of PE's *simultaneously* make comparisons $-[\{0, 1\}, \{2, 3\}, \{4, 5\}, \{6, 7\}]$. PE's $\{0, 2, 4, 6\}$ contain the minimum values of the respective comparisons.

In Step 2 the following pairs make comparisons $[\{0, 2\}\ \{4, 6\}]$. Subsequently, at the end of step 3, $PE_0$ contains the minimum of the eight numbers. Thus, recursive doubling required three time units. In general, for $N$ numbers, $[\log_2 N]$ time units are required as compared to $N-1$ time units using the sequential technique. If $N$ is not a power of two, some dummy numbers can be 'padded up' to obtain a power of two as needed by recursive doubling. As $N$ increases to practical values (say 100) the savings in the number of steps is substantial. (For $N=100$, 7 steps are needed with recursive doubling as compared to 99 steps sequentially.) The recursive doubling technique also requires some data transfers which degrades performance somewhat. However, many existing and proposed interconnection networks can do the data transfers required at each step of the recursive doubling process in one move [3, 12, 15].

## 4. EXECUTING THE SIMPLEX METHOD ON AN SIMD MACHINE

A standard form of the simplex tableau is shown in Figure 4.1. There are several variants of this tableau. This method was chosen simply for expository purposes.

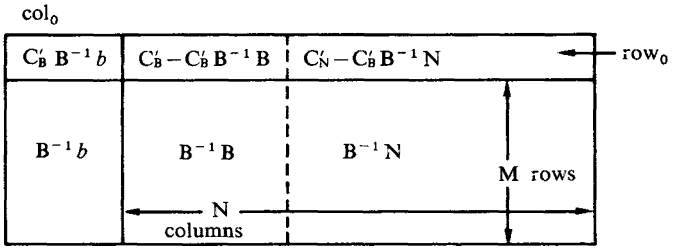


**Figure 4.1. — The Simplex Tableau.**

Starting with an initial feasible solution, the solution is obtained as a sequence of pivot operations. The tableau is assumed stored in the PE's of the SIMD machine in skewed storage format [26]. Skewed storage allows the retrieval of an entire row or an entire column of the tableau in parallel. The architecture of

Figure 3.1 is assumed. Assume for the moment that the number of colums in the tableau is less than or equal to the number of PE's. Techniques for wrapping around' the tableau do exist for the not unlikely case of the number of columns being greater than the number of PE's.

In Figure 4.1, $\text{row}_0$ of the tableau contains the current value of the objective function and the reduced costs. $\text{Column}_0$ includes the objective function value $C'_B B^{-1} b$ and the values of basic variables ($B^{-1} b$). A flow chart of the simplex procedure is given for convenience in Figure 4.2.

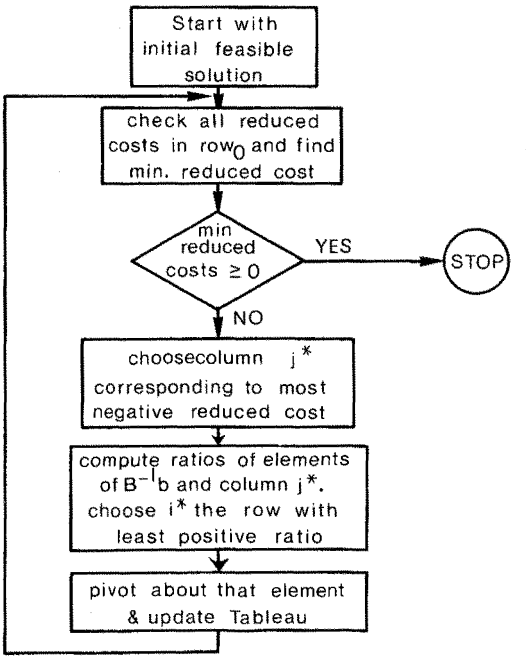


**Fig. 4.2. – Flowhart of the Simplex Method.**

In Figure 4.2, the operation of pivoting brings column $j^*$ into the basis.

A high level "program" for running this on an SIMD machine is given in the next section.

### 4.1. A Parallel program for the simplex method

An explanation of the symbols used in the program follows. The symbols have been divided into three groups. The first group refers to hardware components of the parallel processor. The second group describes variables and the third group explains the operations. It must be emphasised that we are not attempting to list

all hardware components or the full repertoire of instructions of a typical parallel processor. The descriptions are intended only as an aid to following the subsequent procedure.

(i) Hardware components:

**PE** : A processing element, consisting of a processor and its associated memory. Each PE has, among other components, the following three components of interest for our purposes.

**DTR** : Data transfer register. This register is the only one linked to the interconnection network. To transfer data to/from a PE, the data item must be in the DTR of the PE.

**A** : A privileged register. Most arithmetic operations incolve this register. Further, any data loaded from a PE's memory must be loaded first into A.

**B** : A general purpose register.

(ii) Variables :

Variables in an SIMD machine can be of two types. One is the PE variable. Its value is local to every PE. The other is a **CU** (Control Unit) variable. This variable resides in the CU and may be broadcast to all PE's in parallel. This is the usual method of transmitting constant values to PE's.

**PEN** : PE number. This is the address of a PE. If there are $P$ processors, they are numbered 0 to $P-1$.

DIV, JSTAR, ISTAR SCALE, MIN :

These are CU variables. DIV holds the value of the pivot element; JSTAR, ISTAR are the column and row number, respectively, of the pivot element. MIN holds the minimum value of some comparisons.

SCALE holds a common multiplier used in updating the coefficient matrix using the pivot element.

(iii) Operations:

$\leftarrow$ : Assignment. The quantity on the right is assigned to the variable to the left of the arrow. Recall that every operation is done in parallel by each PE unless otherwise modified by a mask as described below.

[Mask] : Normally, all PE's execute any instruction handed down by the CU. The masking operation selects some PE's to perform an operation. Such PE's are said to be active. Theoretically, every operation could be accompanied by a mask. There are various representations of masks [17]. When an operation is accompanied by a mask, only PE's whose address (PEN) matches the mask will execute that instruction.

$x$ **Mod P** : $x$ modulo $P = x - P^* \left[ \frac{x}{P} \right]$.

**Shift**$_x$ : This is an inter PE data transfer between the $A$ registers via the DTR's and interconnection network. This operation causes $PE_i$ to transfer its data to $PE_{(i+x) \text{ Mod } P}$ where $P$ is the number of processors. It must be emphasized that all PE's can do this at the same time due to the nature of the interconnexion network. This should be evident in section 6.

WHERE (Logical expression) DO;
:
END;

: This is a data conditional operation. Initially, all active PE's evaluate the logical expression using their individual data streams. Those PE's among this active set, where the expression evaluates to false, are deactivated. Only the remaining active PE's execute the subsequent DO; ... END; block. This is a standard construction in languages for parallel processors [1, 25].

R.D. Comp $i$ to $j$ : PE's $i$ to $j$ perform recursive doubling as described in section 3.2. The numbers to be compared, reside in the $A$ registers of PE's $i$ to $j$. The minimum value appearing as a result of the recursive doubling process is contained in the DTR of $PE_i$. It is assumed that $ij$ and that $j-i+1$ is a power of two or the data is padded up.

IF (Logical expression) THEN...;:

The control unit evaluates the expression. If it evaluates to true, the THEN block is broadcast to the PE's. This is a standard IF-THEN statement.

/*Comment */ : Explanatory comments explaining particular steps.

Referring to Figure 4.1, there are $N+1$ columns and $M+1$ rows in the tableau. Assume $N+1 > M+1$ and further that $N+1$ is a power of two. Should $N+1$ not be a power of two, assume that there are $2^r$ PE's where $2^{r-1} < N+1 < 2^r$. These assumptions are made in the interest of simplicity. Section 4.2 discusses the realistic possibility of having too big a tableau. The parallel program now follows. Since by assumption, $P$ (the number of processors) $= N+1$, they are numbered 0 to $N$.

/* *Step* 1. Find the minimum reduced cost */.
$A \leftarrow \text{row}_0$ [PE's 0 to $N$]; /* Each PE loads its element of $\text{row}_0$ into its register $A$ */
R.D. Comp [1 to $N$]; /* Find the minimum Value */.
$B \leftarrow$ DTR [$PE_1$]; /* Save the minimum Value */
IF ($B \geqq 0$) [$PE_1$] THEN STOP;
/* All reduced costs are non negative. So stop */
MIN $\leftarrow$ DTR [$PE_1$];

```
WHERE (A=MIN) DO [PE 1 to N]; /* find the PE in which */
  JSTAR ← PEN;                /* the lowest number */
END;                          /* resides */

/* Step 2. Compute ratios of x_i/a_{i,JSTAR}
A ← col_0 [PE's 0 to M]; /* All PE's load column zero */
B ← A [PE's 0 to M],     /* Save column 0 */
A ← col_JSTAR [PE's JSTAR to (JSTAR+M+1) Mod P]; /* P=N+1 */
Shift_-JSTAR [PE's JSTAR to (JSTAR+M+1) Mod P];
A ← B/A [PE's 1 to M]; /* Each PE divides the contents of its B and A register
                          and stores result back in its A register */

WHERE (A<0) DO [PE's 1 to M]; /* A setup for following */
  A ← INF;                    /* recursive doubling */
                              /* operation, INF is an */
END;                          /* arbitrarily high positive number */
R.D. Comp [1 to M]; /* find the minimum quotient */
MIN ← DTR [PE_1];
WHERE (A=MIN) DO [PE's 1 to M]; /* find the row */
  ISTAR ← PEN;                  /* with the lowest */
END;                            /* ratio */

/* Step 3. Now perform the pivot operation */

(a)  DIV ← A [PE_(ISTAR+JSTAR) mod P];
     A ← row_ISTAR [PE's 0 to N];
     A ← A/DIV [PE's 0 to N];
     B ← A [PE's 0 to N];
     Store A [PE's 0 to N];

                  /* Skewed storage */
For j=0 to M DO; /* Update all other rows */
  IF (j≠ISTAR) THEN DO;

(b)  A ← B [PE's 0 to N];
     Shift_-(j-ISTAR) [PE's 0 to N];
     A ← row_j [PE's 0 to N];
     SCALE ← A [PE_(j+JSTAR)];
     A ← A-B* SCALE [PE's 0 to N];
     Store A [PE's 0 to N];

  END;
END;
Go to Step 1.
```

The purpose of this procedure is to show how the simplex method can be executed on an SIMD machine. Estimates of improvement in solution time over the uniprocessor case follows in section 4.3. Note that the test WHERE (A = MIN)... may result in nonunique PE numbers. In such cases, any one of the PEN's holding the minimum value could be arbitrarily assigned to JSTAR (or ISTAR). This does not affect the algorithm.

## 4.2. Wraparound

It could very well happen that the number of columns/rows in the tableau is greater than the number of PE's. In such case, the strategy of wrapping the matrix around the PE's is followed. One or more PE's will then have more than one row/column. Operation on these rows/columns must proceed sequentially and this degrades performance. As an example of wrap around, consider an 8 by 12 matrix and only 8 PE's. The matrix stored in wraparound format will look as shown in Figure 4.3.

| $PE_0$ | $PE_1$ | $PE_2$ | $PE_3$ | $PE_4$ | $PE_5$ | $PE_6$ | $PE_7$ |
|---|---|---|---|---|---|---|---|
| 0,0 | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | 0,7 |
| 1,7 | 1,0 | 1,1 | 1,2 | 1,3 | 1,4 | 1,5 | 1,6 |
| 2,6 | 2,7 | 2,0 | 2,1 | 2,2 | 2,3 | 2,4 | 2,5 |
| 3,5 | 3,6 | 3,7 | 3,0 | 3,1 | 3,2 | 3,3 | 3,4 |
| 4,4 | 4,5 | 4,6 | 4,7 | 4,0 | 4,1 | 4,2 | 4,3 |
| 5,3 | 5,4 | 5,5 | 5,6 | 5,7 | 5,0 | 5,1 | 5,2 |
| 6,2 | 6,3 | 6,4 | 6,5 | 6,6 | 6,7 | 6,0 | 6,1 |
| 7,1 | 7,2 | 7,3 | 7,4 | 7,5 | 7,6 | 7,7 | 7,0 |
| 0,8 | 0,9 | 0,10 | 0,11 | – | – | – | – |
| – | 1,8 | 1,9 | 1,10 | 1,11 | – | – | – |
| – | – | 2,8 | 2,9 | 2,10 | 2,11 | – | – |
| – | – | – | 3,8 | etc. | – | – | – |
| – | – | – | – | – | – | – | – |

**Figure 4.3. – Example of 'Wraparound'.**

The result of this wraparound is that we will have part parallel and part sequential operation. The degree of sequential operations depends on how many wraps we have. For example, with the present set up, any operation on the first eight columns could be done in parallel. Operations involving columns 1 to 10 say will take two steps since $PE_0$, $PE_1$ and $PE_2$ will have to operate sequentially. In the degenerate case of only one PE, all cells of the array are stored continuously and purely sequential operation results.

*Example :* If there was only one PE, the matrix would be stored as follows:

| $PE_0$ |
|---|
| 0,0 |
| 0,1 |
| 0,2 |
| 0,3 |
| 0,4 |
| 0,5 |
| 0,6 |
| 0,7 |
| 1,0 |
| 1,1 |
| 1,2 |
| ⋮ |

The same storage scheme as in a uniprocessor machine

Let $w$ represent the number of wraps for columns and $w'$ the number of wraps for rows. These two numbers need not be the same when the matrix is non-square and skewed storage format is followed. Therefore:

$$w^2 = \left\lceil \frac{N+1}{P} \right\rceil \quad \text{and} \quad w' = \left\lceil \frac{M+1}{P} \right\rceil.$$

### 4.3. Estimates of improvement in solution time

The simplex tableau on which the estimates are based is shown again for convenience below in Figure 4.4.

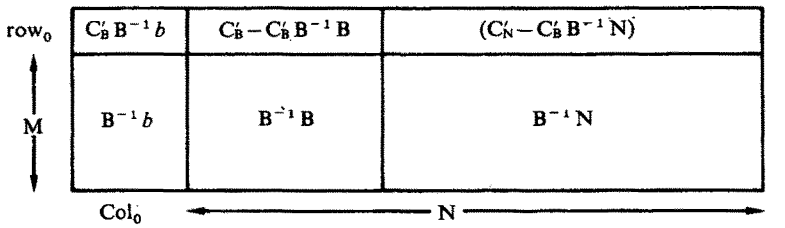


**Figure 4.4. The Simplex Tableau.**

$\text{Row}_0$ contains reduced costs, and column zero the solution vector. $C'_B B^{-1} b$ is the value of the objective function.

If the simplex algorithm of Figure 4.2 is executed on a uniprocessor, the analysis of the number of steps is as follows:

*Step* 1: Find the lowest reduced cost. There are $N$ values for reduced costs that must be examined for the minimum. This will take $N-1$ comparisons.

*Step* 2: Is the minimum reduced cost $\geqq 0$? This is one compare operation.

*Step* 3: Find the row with minimum ratio of $x_i/a_{ij}{}^*$ ($j^*$ is the column corresponding to the minimum reduced cost). This requires $M$ division operations and $M-1$ comparisons (to find the minimum).

*Step* 4: Pivot operations assume $a_{i^*j^*}$ is the pivot. $i^*$ is the row with the lowest ratio in step 3. $\text{Row}_{i^*}$ will require $N+1$ divisions. Every other row will require $N+1$ multiplications and $N+1$ subtractions.

E. g.,:

$\text{row}_{i^*}$: $a_{i^*0}$, $a_{i^*1}$, $\boxed{a_{i^*j^*}}$ (pivot), $a_{i^*N}$,

divide by $a_{i^*j^*}$ : $\frac{a_{i^*0}}{a_{i^*j^*}}, \frac{a_{i^*1}}{a_{i^*j^*}}, \quad , 1, \quad , \frac{a_{i^*N}}{a_{i^*j^*}},$

$\text{row}_r$ : $a_{r0}, a_{r1}, \ldots, a_{rj^*}, \ldots, a_{rN}$,

operation on row $r$: $a_{r0} - a_{rj^*}\left[\frac{a_{i^*0}}{a_{i^*j^*}}\right], \ldots, 0, \quad a_{rN} - a_{rj^*}\left[\frac{a_{i^*N}}{a_{i^*j^*}}\right]$

(1 mult. / 1 subtraction)

Hence for a total of $M+1$ rows we need:
$N+1$ divisions for row $i^*$
$(N+1)M$ Multiplications
$(N+1)M$ subtractions
} for the other $M$ rows.
Hence one iteration of simplex requires $(N+M-1)$ comparisons, $(M+N+1)$ divisions, $(N+1)M$ multiplications, and $(N+1)M$ subtractions.
Now if the same iteration were carried out on an SIMD machine, the analysis of the number of steps would run as follows. Assume $w$ wraps for columns and $w'$ wraps for rows as before.

*Step* 1 : The recursive doubling takes $\log_2 N$ compares and $\log_2 N$ shifts. Such shifts can be done in one move (*see* Section 6.4). In general, at most $\underline{w'+\log_2 N}$ compares and $\underline{\log_2 N}$ shifts are needed. The step to test if the minimum reduced cost is nonnegative also requires one comparison, as in the uniprocessor. (Note $w'$ compares are done within each Pε.)

*Step* 2 : Finding the row with the minimum ratio of $x_i/a_{ij^*}$.

This requires one shift ($\text{Shift}_{-\text{JSTAR}}$) operation and one divide operation. The recursive doubling to find the minimum element needs $\log_2 M$ compares and $\log_2 M$ shifts. In general, at most $(w+\log_2 M)$ shifts and $w$ divides and $w+\log_2 M$ compares are needed.

*Step* 3 : Pivot Operation.

For $\text{row}_{\text{ISTAR}}$, one division is needed, or in general, at most $w'$ divisions. Now, each iteration involves at most,

$2w'$ shifts, $w'$ multiplies, and $w'$ subtractions.

Since we go through the loop $M$ times, we have a total of $(M\,w')$ shifts, $w'$ divisions, $M\,w'$ multiplies, and $M\,w'$ subtractions.

So in terms of shifts and arithmetic operations, one iteration of the Simplex Method on the SIMD machine takes, at most, $(w'+\log_2 N+w+\log_2 M)$ compares, $(\log_2 N+w+\log_2 M+w'\,M)$ shifts, $(w+w')$ divides, $M\,w'$ multiplies, $M\,w'$ subtractions.

To get an estimate for comparative time performance overall, let us make the following realistic assumptions about time for operations:

subtraction : 1 time unit;

multiply : 5 time units;

divide : 8 time units;

comparisons : 3 time units;

$\text{shift}_j$ : 4 time units. (Recall from Section 4.1, that this is an inter PE data transfer.)

(The entries in the following table are time units per iteration of the simplex procedure.)

| Machine | Uniprocessor | SIMD $w=w'=1$ | SIMD $w=w'=2$ | SIMD $w=w'=3$ |
|---|---|---|---|---|
| $N=30$, $M=20$. . . . . | 4 275 ($P=1$) | 290 ($P=32$) | 516 ($P=16$) | 742 ($P=8$) |
| $N=1\,000$, $M=500$. . . . | 3,019,505 ($P=1$) | 5 153 ($P=1\,024$) | 10 179 ($P=512$) | 15 205 ($P=256$) |

For the SIMD machine, the critical factor is the inter PE data transfer time (shift-time) since there are so many of them. A slight change in data transfer time can substantially improve the SIMD performance. It is assumed that a uniprocessor and a processor in an SIMD machine performs an arithmetic/logic operation in the same time.

## 5. AN ARCHITECTURE FOR PARALLEL IMPLEMENTAL OF THE DANTZIG-WOLFE ALGORITHM

The previous sections exposed the parallelism that could be exploited when solving a regular LP problem using the simplex method. This section discusses the architecture of a parallel processing machine that is suited for the class of LP problems that have the block angular structure as described before.

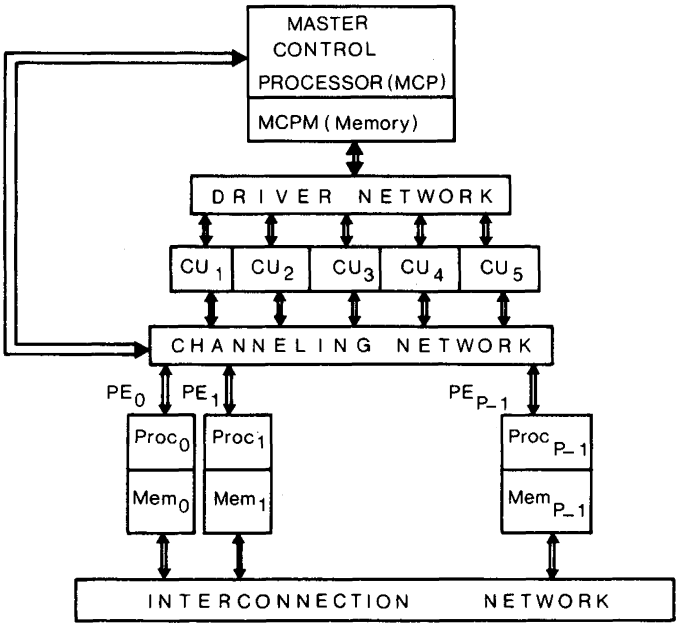


**Fig. 5.1. – Parallel Architecture for D – W.**

The architecture presented in Figure 5.1 is different from the regular SIMD architecture presented in Figure 3.1 in two major ways. First, there are two levels of control. As before, there are $P$ PE's and they perform the bulk of the computation. Also as in Figure 3.1, there is an interconnection network for the PE's to communicate with each other. However, in Figure 5.1, the interconnection network is also partitionable in the sense to be described presently. Our machine has several CU's as opposed to one in the general SIMD machine. (There are five CU's in the diagram only as an example.) The number of CU's is fixed at the time of design. One CU together with a set of PE's constitutes an SIMD machine. Figure 5.1 in effect represents an architecture for several cooperating SIMD machines. The idea of having several SIMD machines (a multiple SIMD or MSIMD system) was also envisioned in the design of the

ILLIAC [2]. Other systems using a similar concept are described in [14, 21]. There is a master control processor (MCP) that is a small processor in its own right and co-ordinates the CU's. MCP communicates with the CU's through the driver network and the CU's communicate with the PE's through the channeling network that also happens to be partitionable.

Execution of the Dantzig-Wolfe ($D-W$) procedure occurs in the following manner. The MCP will be in charge of handling the master problem of ($D-W$), and co-ordinating the CU's. Each CU handles one or more subproblems. This is where the ability to partition the machine structure becomes crucial. The sizes of subproblems (i.e., number of constraints and/or variables) will vary from problem to problem. Recall from earlier sections that each subproblem in $D-W$ is an LP problem and that an LP problem could be executed efficiently using the parallelism inherent in an SIMD machine. As such, the architecture must permit the flexibility of assigning a variable number of PE's to each CU, depending on the size of the subproblems. This will make maximum use of the computing resources. A collection of PE's together with the CU to which they are assigned, solves one subproblem. Such a CU-PE collection unit functions as an SIMD machine. It is thus apparent why partitioning capability in the interconnection network and in the channeling network is needed. Each partition of these two networks will serve a group of PE's that are working on the same subproblem. The next section discusses details of the various components of the machine.

Most parallel processing machines function as special purpose processors in conjunction with some larger general purpose computer. The ILLIAC has a B 6500 as a host for example [5]. The function of the general purpose machine is to perform I/0 functions and set up data in the respective PE's, handle interrupts, etc. The reader may also refer to [28], for descriptions of such total systems.

## 6. SYSTEM DETAILS

We now elaborate on the different structural components in the architecture of the parallel processor presented in Figure 5.1. Since implementation issues are beyond the scope of this paper, unnecessary detail has been omitted. However, it should be clear that the structural components are relatively simple.

### 6.1. The processing element

The internal structure of a processing element is shown in Figure 6.1.

The A register is the accumulator; registers $B$ and $C$ hold operands for binary operations. Any data transfers to and from other PE's is done via the DTR.

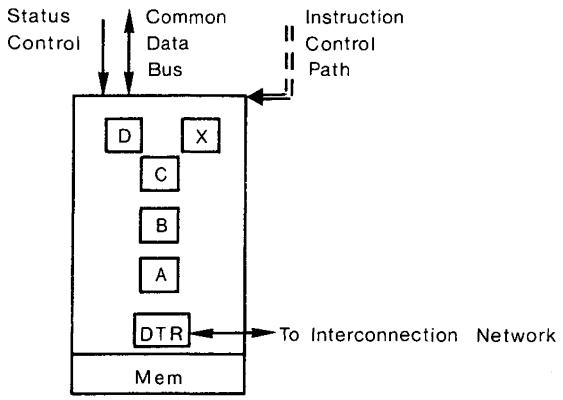


**Fig. 6.1. – The Processing Element.**

The *X* register is an index register and *D* a status register. Each PE has lines to its control unit for transmission of status and for reception of common data and for instruction execution control.

## 6.2. The control units (CU)

The CU's drive the PE's. Figure 6.2 contains the details of the components making up a typical CU. The basic operation performed by any CU is instruction decoding and generation of the associated proper sequence of

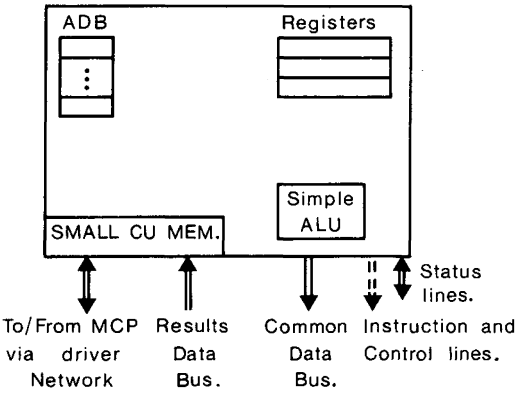


**Fig. 6.2. – Control Unit Details.**

operations to drive the PE's. Any data that is common to all PE's, say a common multiplier, is transmitted to all PE's in parallel through the Common Data Bus. Recall from the program in section 4, that particular PE's had to be deactivated

during certain operation. Thus, while the same instruction is broadcast to all PE's, only the active ones execute it. The status lines are used to transmit this information to PE's.

The accumulators and ALU are the registers and arithmetic/logic unit within the control unit. They are necessary for instruction decoding and other simple arithmetic operations the CU may have to perform.

The CU memory would be typically much smaller that the PE memories. The program to be executed is stored in the CU memory. Reserved areas of this memory can be written to by PE's using the Results data bus and read by the MCP. As in the ILLIAC [5], the **ADB** (advanced Data Buffer) is just a high speed scrath pad memory. The internal structure of this CU reflects the needs of the application. For example, the Results data bus, and the bus leading to the MCP are necessary to receive and transmit subproblem results to the master control processor.

### 6.3. Channeling network

A multiplexor arrangement is proposed for the channeling network, with the requirement of partitionability in mind.

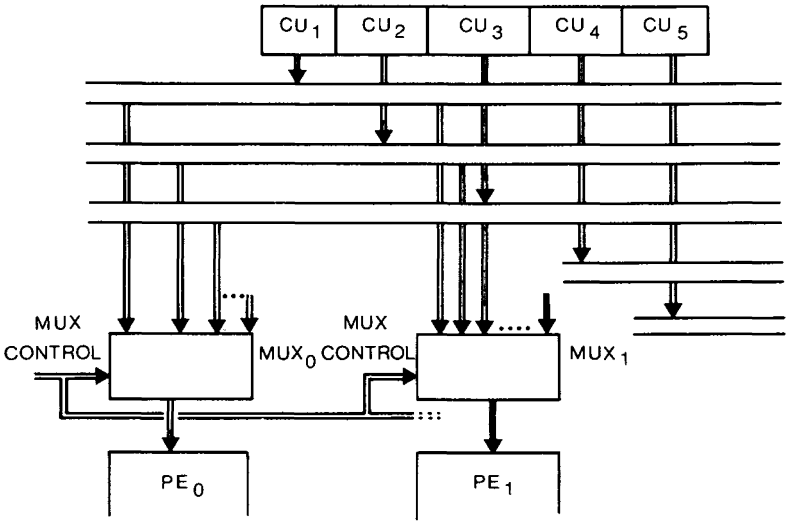


**Fig. 6.3. – The Channeling Network.**

Each PE gets its instruction stream from our particular CU, depending on its MUX CONTROL. The MUX CONTROL is set by the MCP for a particular

assignment of PE's to CU's. This multiplexor arrangement enables us to assign a variable number of processors to a CU depending on the size of subproblems. This is in effect a cross-bar switch as used, in the MAP, MSIMD system [14].

### 6.4. Interconnection network

The interconnection network allows PE's to transfer data among themselves. There are various propositions for interconnection networks each with its own advantages, as mentioned in section 3. The interconnection can be viewed as a blackbox as shown in Figure 6.4.

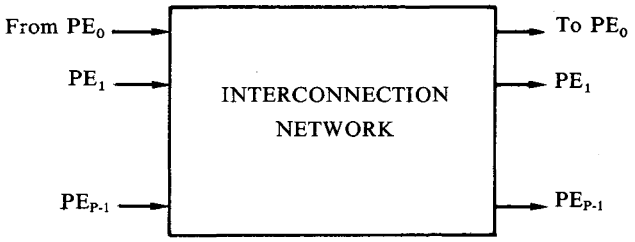


**Fig. 6.4. – Interconnection Network as a Blackbox.**

Data from each PE can be routed to other PE's through the network. The network we wish to use is called the multistage cube network. As mentioned in section 5 in connection with the analysis, inter PE data transfers are a major overhead with SIMD machine. Hence it would be to our advantage to design the network to reduce the number of transfers required for the class of applications envisioned. One of the major transfer operations performed for the $D-W$ algorithm is during loading of rows and columns from skewed storage. This has to be done very frequently. The other requirement that must be satisfied is that the network be partitionable; i.e., the various subnetworks arising out of the partition should have the same data transfer capabilities as the whole network. Recall that PE's are assigned to certain CU's. It is undesirable that PE's working on one subproblem transfer data to/from another set of PE's working on another subproblem.

The multistage cube network [18] succeeds on both counts. Its configuration for the case of eight processors is shown in Figure 6.5.

At every stage $i$, input lines that differ in the $i$-th bit of the binary representation of their number, are paired together. The boxes are known as interchange boxes and they can either pass their two inputs striaght through or switch them as shown by dotted lines in Figure 6.5.

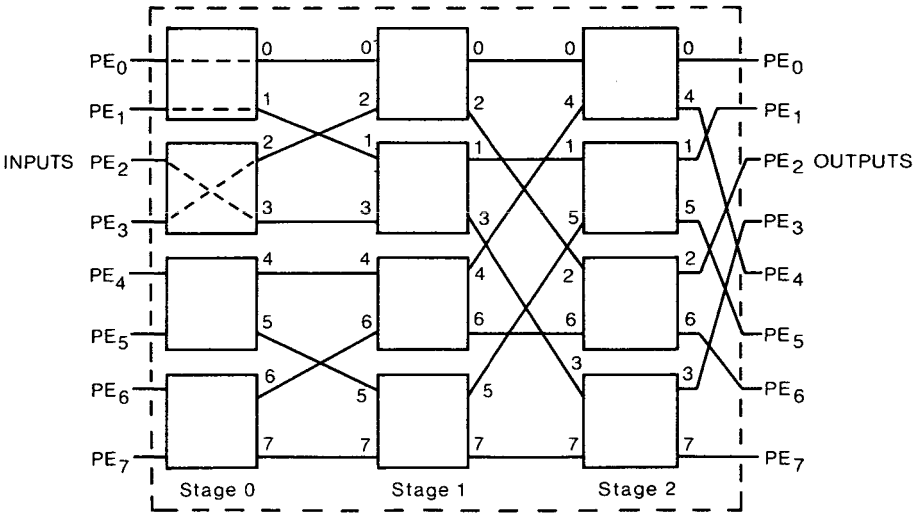


Fig. 6.5. – The Multistage Cube Network.

A uniform shift of $x$ occurs when ever PE[$i$] transfers data to PE[$(i+x)$ Mod $P$] where $P$ is the number of processors. The multistage cube network can perform uniform shifts in a single pass through the network. This has been proved [12]. Each interchange box is independently controllable.

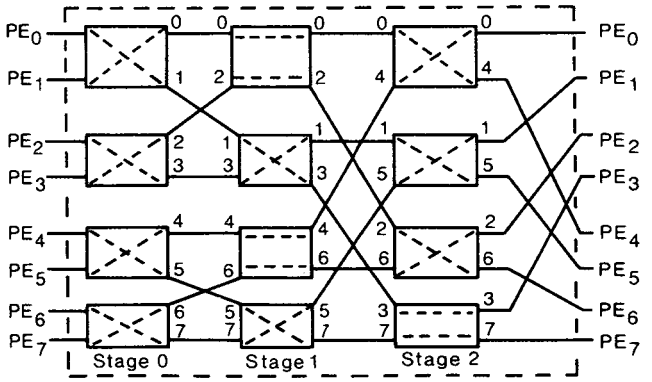


Fig. 6.6. – A Uniform Shift of Three.

One routing scheme employs destination tags. Each data item is tagged with the PEN of its destination PE. For example, for a uniform shift of 3, $PE_0$ would tag it data item with 011 (binary representation of 3, the destination PE), $PE_4$ would attach a tag of 111. At stage $i$, if the $i$-th bit of the sending PE is a Ø then

the upper box ouput is taken, if it is a 1 the lower box output is taken. Consider the uniform shift of 3. The switching pattern of the boxes is shown in Figure 6.6.

The multistage cube network can also be partitioned into smaller networks, each of which is in itself a multistage cube network. For example, setting all the boxes in stage 2 to "straight" produces two subnetworks. – $\{0, 1, 2, 3\}$ and $\{4, 5, 6, 7\}$. Each of these is in itself a multistage cube network. For details, *see* [20, 23]. The reader will have noticed that partitions are powers of two. For example, 1 024 processors, can be partitioned into subgroups of 64, 64, 128, 256, 512 or 128, 128, 256, 512, etc.

### 6.5. The master control processor and driver network

The Master Control Processor (MCP) is in charge of handling the master problem. It is also charged with setting the multiplexor in the channelling network; i. e., it controls the assignment of certain PE's to CU's. The CU's coordinate solutions to subproblems in turn. Steps 2 and 3 in $D-W$ (Section 2) are associated with the Master problem. The internal structure of MCP is much the same as that of a CU except that it has more sophisticated arithmetic/logic capabilities than the CU. This is necessary because in addition to instruction decoding, it has to perform the arithmetic/logic demanded by steps 2 and 3 of $D-W$.

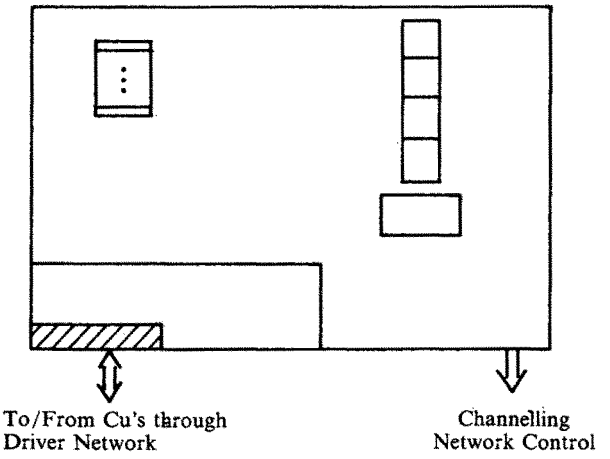


Fig. 6.7. – The Master Control Processor.

It is quite likely that the number of subproblems will be much smaller than the number of rows or columns of the constraint matrix. Hence, it seems practical to perform the summation of $\Sigma Z_i$ in Step 2 of $D-W$ (Section 2) sequentially. The

hatched portion of MCPM is a reserved area in which the CU's deposit their values of $Z_i$, and decision variables. Recall that simplex multipliers from the master problem must be transmitted to the CU's for use in subproblem solving. This is accomplished through the DTR (Data Transfer Register) in Figure 6.7. In other respects, the MCP is just like a control unit. It has an ALU and some general purpose registers.

## 7. SUMMARY

An architecture of a parallel processor that is specially suited for solving a class of operations research problems has been presented. We recognize the existence of many techniques for efficiently solving large scale systems optimization problems. In our opinion, parallel processors are an interesting class of machines which could be profitably used in the efficient solution of large scale optimization problems. With the emergence of powerful low cost microprocessors, the viability of having a very large number of processors in one machine is hardly in doubt. However, the successful application of such machines to the optimization area will require the close co-operation of both hardware specialists and operations research specialists. Such joint efforts are already underway in other areas of application [24, 27].

## REFERENCES

1. N. E. Abel *et al.*, *TRANQUIL: A Language for an Array Processing Computer*, AFIPS 1969 SJCC, May 1969.
2. G. Barnes *et al.*, *The Illiac IV Computer*, *I.E.E.E.* Trans. Comp., Vol. C-17, No. 8, August 1968.
3. K. E. Batcher, *The Flip Network in STARAN*, 1976 Int'l Conf. Parallel Processing, August 1976.
4. K. E. Batcher, *STARAN Parallel Processor System Hardware*, in 1974 Nat. Comput. Conf., A.F.I.P.S. Conf. Proc., Vol. 43, May 1974.
5. W. J. Bouknight *et al.*, *The ILLIAC IV System*, Proceedings of the I.E.E.E., Vol. 60, No. 4, April 1972.
6. J. A. Cornell, *Parallel Processing of Ballistic Missile Defense Radar Data with PEPE*, COMPCON 1972, September 1972.
7. B. A. Crane *et al.*, *PEPE Computer Architecture*, COMPCON 72, *I.E.E.E.* Comput. Soc. Conf., September 1972.
8. G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, N.J., 1963.
9. G. B. Dantzig and P. Wolfe, *The Decomposition Algorithm for Linear Programming*, Econometrica, Vol. 9, No. 4, 1961.
10. D. P. Dzielinski and R. E. Gomory, *Optimal Programming of Lot Sizes, Inventory and Labor Allocation*, Management Science, Vol. 11, No. 9, 1965.

11. L. S. LASDON, *Optimization Theory for Large Systems*, Macmillan Series in Operations Research, MacMillan Company, New York, 1970.
12. D. H. LAWRIE, *Access and Alignment of Data in Array Processor*, I.E.E.E. Trans. Comp., Vol. C-24, No. 12, December 1975.
13. A. S. MANNE, *Programming of Economic Lot Sizes*, Management Science, Vol. 14, 1958.
14. G. J. NUTT, *Microprocessor Implementation of a Parallel Processor*, 4th Symp. Comp. Arch., March 1977.
15. M. C. PEASE, *The Indirect Binary N-Cube Microprocessor Array*, I.E.E.E. Trans. Comp., Vol. C-26, May 1977.
16. D. ROHRBACKER and J. L. POTTER, *Image Processing with the STARAN Parallel Computer*, Computer, Vol. 10, August 1977.
17. H. J. SIEGEL, *Analysis Techniques for SIMD Machine Interconnection Networks and the Effects of Processor Address Masks*, I.E.E.E. Trans. Comput., Vol. C-26, February 1977.
18. H. J. SIEGEL, *Interconnection Networks for SIMD Machines*, Computer, Vol. 12, No. 6, June 1979.
19. H. J. SIEGEL, *A Model of SIMD Machines and a Comparison of Various Interconnection Networks*, I.E.E.E. Transactions on Computers, Vol. C-28, No. 12, December 1979.
20. H. J. SIEGEL and S. D. SMITH, *Study of Multistage SIMD Interconnection Networks*, 5th Annual Symp. Comp. Arch., April 1978.
21. H. J. SIEGEL, P. T. Jr. MUELLER and H. E. Jr. SMALLEY, *Control of a Partitionable Multimicroprocessor System*, 1978 Int'l. Conf. Parallel Processing, August 1978.
22. H. J. SIEGEL, L. J. SIEGEL, R. J. McMILLEN, P. T. Jr. MUELLER and S. D. SMITH, *An SIMD/MIMD Multi-Microprocessor System for Image Processing and Pattern Recognition*, 1979 I.E.E.E. Comp. Soc. Conf. Pattern Recognition and Image Processing, August 1979.
23. H. J. SIEGEL, *The Theory Underlying the Partitioning of Permutation Networks*, I.E.E.E. Trans. Compt., September 1980.
24. L. J. SIEGEL, H. J. SIEGEL, L. J. SAFRANEK and M. A. YODER, *SIMD Algorithms to Perform Linear Predictive Coding for Speech Processing Applications*, 1980 Int'l. Conf. Parallel Proc., August 1980.
25. K. G. Jr. STEVENS, *CFD-A FORTRAN-like Language for the ILLIAC IV*, Conf. Programming Languages and Compilers for Parallel and Vector Machines, ACM, March 1975.
26. H. S. STONE, *Parallel Computers*, in *Intro to Computer Architecture*; H. S. STONE, Ed., SRA, Inc., Chi., 1975, pp. 318-374.
27. P. H. SWAIN, H. J. SIEGEL and B. W. SMITH, *Contextual Classification of a Multispectral Remote Sensing Data Using a Multiprocessor System*, I.E.E.E. Trans. on Geoscience and Remote Sensing, Vol. GE-18, No. 2, April 1980.
28. K. J. THURBER, *Large Scale Computer Architecture: Parallel and Associative Processors*, Hayden Book Company Inc., Rochelle Park, N.J., 1976.
29. A. C. WILLIAMS, *A Treatment of Transportation Problems by Decomposition*, J. Soc. Ind. Appl. Math., Vol. 10, No. 1, 1962.