DAP—A DISTRIBUTED ARRAY PROCESSOR

Dr. S. F. Reddaway
Language and Processor Department
Research and Advanced Development Centre
International Computers Limited

ABSTRACT

An array of very simple processing elements is described each with a local semiconductor store. The array may also be used as main storage.

Bit-organisation gives great flexibility, including the minimisation of word length. Use of MSI and LSI is helped by the simplicity of the serial design. Using 15-bit fixed point, the theoretical performance of a 72 x 128 array is about 10^8 multiplications or 10^9 additions per second. Comparisons are made with other architectures.

Meteorology is considered as an application. It is attractive to have the whole problem in the array storage.

1. INTRODUCTION

This paper describes a design study of an array of elements that can be used either as a "Single-Instruction, Multiple-Data stream" (SIMD) processor or as a store. Architectural features of interest are: (a) the use of serial arithmetic to simplify processor logic and optimise store utilisation; (b) an attempt to avoid I/O bottlenecks by mapping complete problems into the array, without relying on overlay techniques; (c) provision for using all or part of the array as a store when not performing its specialised processing functions; (d) the close integration of storage and logic.

The main attractions of array-type SIMD structures are: (a) high absolute performance on certain problems of importance; (b) high performance/cost, partly resulting from using common control logic.

Several examples of this type of architecture have been proposed (1-8) and applications have been suggested in, for example, meteorology, plasma physics and linear programming. Most structures have a single control unit that broadcasts instructions to a regular array of processing elements (PEs) each with individual storage and an arithmetic unit (AU).

Flynn (2) points out four factors that degrade the performance from the theoretical figure given by "Number of PEs times PE performance": (a) Each PE has direct access only to a limited region of store, and excess time may be taken accessing other regions; (b) Mapping the problem onto the array may leave some PEs unused; (c) Owing to overheads in preparing instructions for the array, there may be times when the whole array is idle; (d) While dealing with singularities or boundary conditions the majority of PEs are idle.

These factors are acknowledged to reduce the applicability of such an array. In the present design attempts have been made to mitigate their effect, but the over-riding consideration has been to simplify the PE design; this has been done to the extent that the theoretical performance is very high, in spite of the AU cost being small compared with that of the storage. In effect, therefore, the store is being adapted to an array processing function. This may be contrasted with attempts to adapt the processor to array operations (e.g. CDC STAR).

A dispersed system, i.e. one with many PEs each with local memory, has potential cost and speed advantages deriving from: (a) reduced "cable" delays; (b) reduced address transforming and checking; (c) faster actual access; (d) simplified data routing and priority logic.

A number of potential PE designs of varying parallelism have been considered for building arrays of the same theoretical performance, with the following general results.

The gate count varies with the degree of internal PE parallelism. A purely serial PE has considerable advantages particularly for low precision work.

Serial PEs have fewer connections at all packaging levels.

The extreme simplicity of serial PEs permits the very effective use of batch fabrication and testing techniques and keeps hardware development rapid and cheap. The small number of circuit and board types helps development, production, spares holding and maintenance.

Serial designs have exceptional functional flexibility; very few decisions are built into the hardware. However, fully indexed addressing is expensive.

The design is somewhat similar to SOLOMON 1 (8); the main differences stem from the exploitation of modern technology.

2. THE ARRAY

2.1 CONFIGURATION

FIGURE 1. M.C.U. DIAGRAM

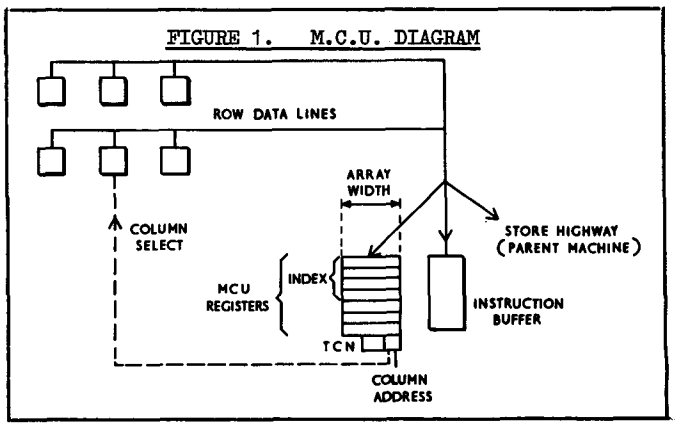


Figure 1 is an overall configuration diagram. The rectangular array has an essentially two dimensional nearest neighbour connectivity, and has one dimension matched to the store highway of a conventional computer (the "parent" machine). This connection provides the route for loading both data and array instructions into the array storage for array processing; it also permits the parent machine to use the array storage as its own main storage. Input/output is done by the parent machine.

The Main Control Unit (MCU) has: (a) a conventional instruction fetching arrangement; (b) an instruction buffer whose purpose will be described later; and (c) a set of registers, many of which can be matched to the array by row or column for a variety of purposes, one of which is indexing. For sizable arrays the MCU is a very small fraction of the total hardware.

After loading, the bits of a word are spread along a column of PEs, and this method of holding data is termed Main Store mode. Another method, termed Array mode, stores all the bits of a word in a single PE. This is more attractive for processing large arrays, but requires initial and final transformation of the data from and to Main Store mode; this is done inside the array.

2.2 THE PE

FIGURE 2. PROCESSING ELEMENT

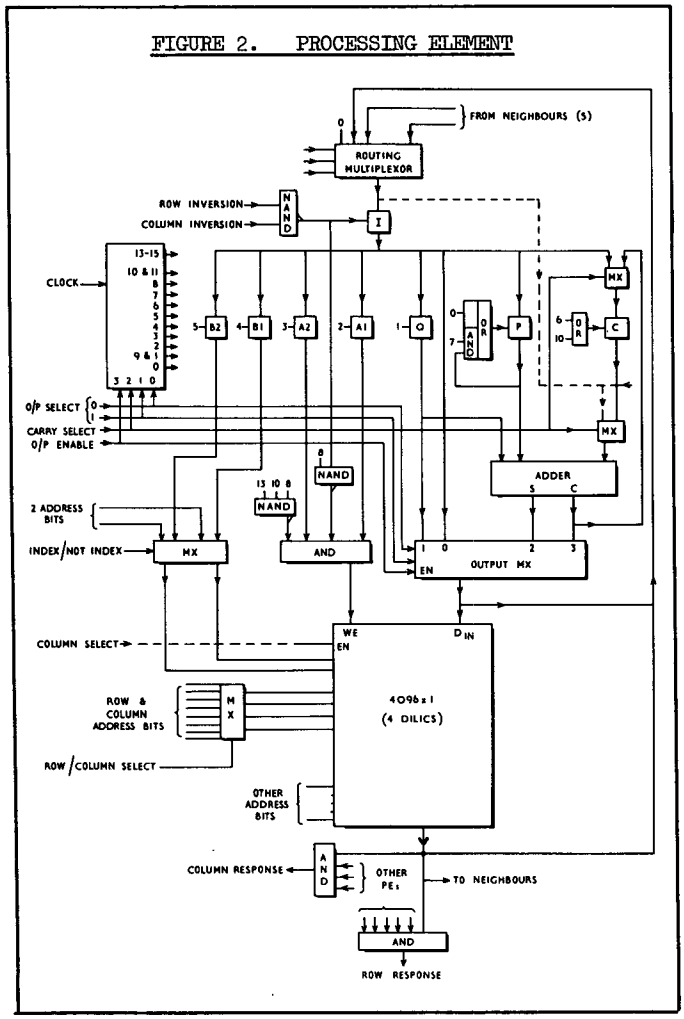


Figure 2 is a PE diagram. The registers are all one-bit; P and Q are for operands, C is the carry register, A1 and A2 are activity bits that can prevent writing to store, and B1 and B2 can supply 2 address bits. The routing multiplexor can select a bit from the PE's own store, or from a neighbour's store, for writing to a register; selecting zero and controlling its inversion permits data input from outside the array (for example, an MCU register). The sum, carry, data input or contents of Q can be output from the logic, usually to the store. The store contents can be output externally (to, for example, an MCU register) via the gates at the bottom of Figure 2; the bits output can be either from a selected column of PEs, or the logical AND of rows (or columns) of PEs. One use for the latter is for a test over all PEs.

The fifth "neighbour" connection is to the PE half a row away in the same row; this permits both faster mass movement of data around the array, and a "$2\frac{1}{2}$D" PE geometry. Bit patterns in one or two MCU registers can be applied to the "inversion" inputs to produce a veto selective by rows and/or columns on writing to PE stores. Figure 2 shows 4 address bits capable of being selected by row or column; what indexing facilities should be provided is still an area of debate.

Some differences from the PE in (7) are: (a) more row/column symmetry; (b) a latch feature (shown on the P register) for associative comparisons; (c) data can be shifted directly between PEs without using the store; (d) input data can be loaded directly into store; (e) there is a ripple carry path between PEs for Main Store mode arithmetic; (f) the bipolar store is now 4K instead of 2K.

It is intended to package 2 PEs minus their stores and routing multiplexors in one 24 pin integrated circuit.

2.3 EDGE CONNECTIONS

For instructions that involve neighbours, it is the array geometry that determines what happens at the array edges. Rows or columns may be: (a) cyclic, with their ends connected together; (b) linear, with a continuation onto a neighbouring line; (c) as (b) but with the extreme ends connected; or (d) plane, with external data applied at the relevant edge. In addition, a row may be considered in two halves ($2\frac{1}{2}$D geometry). There are thus 32 geometries, and they are set by program.

2.4 CONSTRUCTION

A board would contain a 6 x 4 PE section with 4K bits/PE; there would be 137 external connections and 173 ICs, 96 of them for storage. The array can be viewed as doing processing in the store, and costs only about 25% more than ordinary storage made out of the same technology. A platter would contain a 36 x 16 PE section; the number 36, and multiples of it, match standard store highways. "Folding" of the array makes connections between the extreme edges short.

The economy obtained by the dense packing of the integrated circuits is the result of the favourable marriage of space-limited (or power-dissipation limited) storage and pin-limited logic.

2.5 TIMING

Because most micro-instructions do not involve a response from the array, the equalisation, rather than minimisation, of delays is important. Even with a comparatively slow logic technology, the micro-instruction rate should be about 5-6 MHz; the storage

element delays are the biggest factor, and this illustrates how the array can exploit bipolar store speeds, unlike a large conventional machine.

2.6 FUNCTIONS

In (7) the basis of the micro-programming notation is given and it is shown how Array mode fixed and floating point instructions are built-up. Bit organisation means that only necessary work need be done; for example, multiplication only needs to calculate a single length result.

Code for execution must be compiled down to the one-bit micro-instructions, except that for working regularly along the bits of words a short loop can be constructed. This loop is held in the instruction buffer, so that no further instruction fetching from the array storage is needed during execution of the loop. This feature reduces the instruction fetching overhead from 100% to about 20%. Subroutine construction will be possible.

2.7 PERFORMANCE

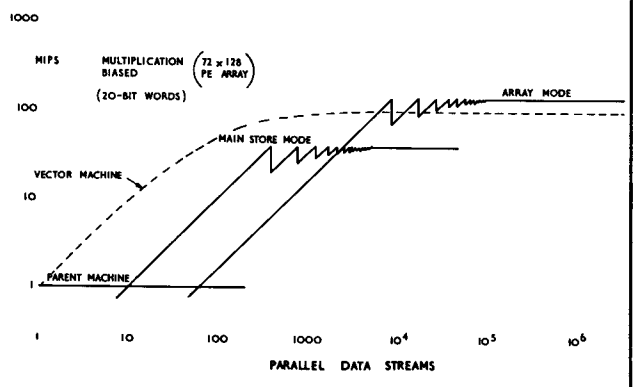
For array mode, fractional fixed point multiplication takes about

$$\frac{n\,(3n + 13)}{2}$$

micro-instructions where n is the word length; fixed point addition takes little more than 3n micro-instructions. Floating point takes a little longer for multiplication, and considerably longer for addition (see (7)). 20-bit multiplication takes about 730 micro-instructions plus about 160 cycles for micro-instruction fetching, and at $5\frac{1}{2}$ MHz would take about 160 μsec; 20-bit addition takes about 12 μsec. Multiplication of an array by a common number can be about four times faster.

Main store mode arithmetic is faster than Array mode for smaller arrays. In terms of absolute speed, addition is about 11 times faster and multiplication, using a carry save technique ending with a ripple carry, is about six times faster for 20 bit precision (the latter factor increases with the precision).

FIGURE 3. D.A.P. PERFORMANCE



The user has three modes of working at his disposal: the parent machine for scalar working, Main Store mode for small arrays and Array mode for large arrays. Figure 3 shows roughly what is possible in the three modes; the useful processing rate in Million Instructions (or, more accurately, results) Per Second (MIPS) is plotted against the number of parallel data streams for the type of computing indicated and a 9200 PE array. Only the top ends of the sloping lines depend on array size. The dashed line shows the similar graph for a powerful vector machine (there are many other differences between the two types of machine).

The overall performance depends on the application and programmer skill.

2.8 A COMPARISON

ILLIAC IV is a well known machine, so a brief comparison is attempted with Array mode, assuming the problem parallelism is sufficient to occupy either machine. Many differences are not easily quantifiable, but as a starting point the main assumptions for a numerical comparison are given in Figure 4. The first four lines give the instruction mix; B is the number of bits precision for the serial design, which has no separate store acesses because all functions are store-to-store. P is the clock period (180 nsec). 20% is subtracted from the ILLIAC IV totals to allow for instruction overlap.

FIGURE 4. DESIGN COMPARISON
ILLIAC IV ASSUMPTIONS

INSTRUCTION MIX AND TIMINGS:

	SERIAL DESIGN	ILLIAC IV			
		SINGLE PRECISION	DOUBLE PRECISION	TRIPLE PRECISION	
1 ADD/SUBTRACT	$(2+3B)P$	0·125	0·25	0·5 ?	μsec
1 MULTIPLY	$(4B+1·5B^2)P$	0·25	0·5	2·0?	μsec
2 STORE ACCESSESS	0	0·325	0·65	1·0?	μsec
1 MODE SETTING (Etc.)	4P	0·05	0·05	0·05	μsec
TOTAL	$(6+7B+1·5B^2)P$	0·75	1·45	3·55?	μsec
TOTAL -20%		0·6	1·16	2·8?	μsec
MANTISSA		25	49	73	BITS
EXPONENT		7	15	(23)	BITS
"USEFUL" EXPONENT		4	6	8	BITS

LOGIC/PE.
ILLIAC IV ~12000 FAST ECL GATES
SERIAL DESIGN ~60 TTL GATES
1 FAST ECL = 2TTL GATES
RATIO = 200 x 2 = 400

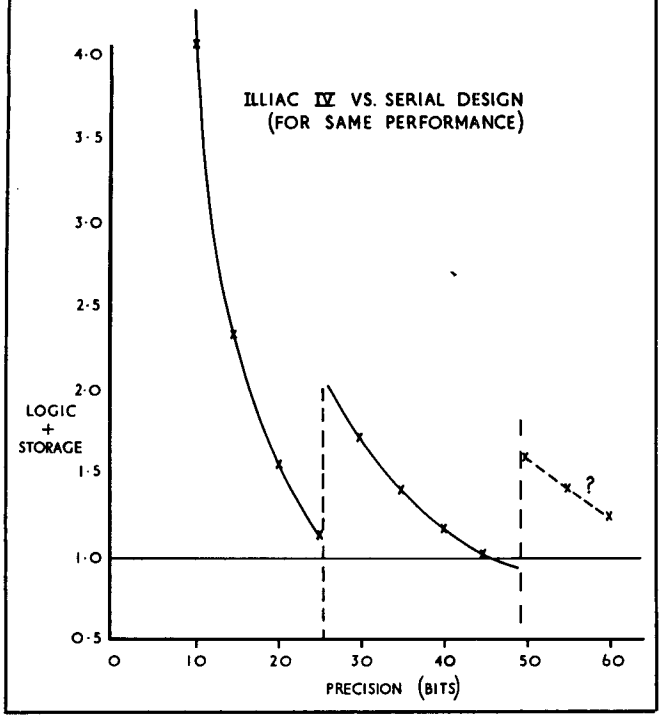
Figure 5 compares the hardware required to build an array of given performance for words of a particular precision. Logic and storage have equal weight; Figure 4 gives the gates/PE ratio and the storage comparison involves an estimate of the unnecessary bits in the ILLIAC IV word. The graph would favour ILLIAC IV only for working exclusively with 46-49 bit precision. At low precisions serial PEs have a very big advantage.

Such numerical comparisons are of only limited value. For example, the vertical scale of Figure 5 would be multiplied by about 4 if integrated circuit count were used as a hardware measure. Other factors such as hardware simplicity and repetition, pin counts and functional flexibility are equally important.

2.9 EXAMPLE OF STORAGE ECONOMY

For problems with large amounts of data, storage economy is important, particularly if it permits storing the complete problem in the array. The user can apply various tricks. As an example, consider three dimensional field problems. In order to prevent physical "truncation" errors, programs are designed so

FIGURE 5. COMPARISON WITH ILLIAC IV



that differences between neighbouring variables require fewer significant bits than the variables themselves. If variables have to be held simultaneously for two time steps, then, for example, they can be grouped into sets of 16 nearest neighbours in space and time (2 x 2 x 2 x 2), and held as follows: (a) a short floating point number close to the maximum of the group (maybe a 4-bit mantissa and 3-bit exponent); and (b) 16 differences in block floating point (maybe 12-bit mantissas and a common 2-bit block exponent). This results in 12.6 bits/variable and is roughly equivalent to floating point with a 15-bit mantissa and 3-bit exponent, i.e. a gain of nearly 50%; other machines require floating point variables to occupy up to 64 bits, i.e. up to 5 times more.

3. METEOROLOGY AS AN APPLICATION

This is considered more fully in (7). Meteorology includes both simulation experiments and forecasting, and as simulation programs are central to both, attention will be confined to them. (Forecasting also uses analysis and initialisation programs to assimilate the "real" data). For simulation programs, the frequency of add/subtract and multiply instructions is roughly equal, and divide is much less frequent. For DAP, multiplication takes much longer than addition, so the number of multiplications and their timing give a first approximation to the speed of a program.

The table gives a rough guide to parameters in use today and those that should be aimed at.

Using the 18 bit (fixed point) precision suggested in Section 3.3, each PE can perform a multiplication in about 140 μsec. Section 3.2 discusses the efficiency of PE usage; 50% might be a reasonable figure. Thus about 8000 PEs are adequate to perform the 2.5×10^7 multiplications per second indicated above.

TABLE

	Present		Next stage
	Forecast Programs	Global Research Programs	
Number of Vertical Columns of Grid Points	3000	10 000	x 4
Number of vertical levels	10	5	x 2
Total number of variables	2×10^5	2.1×10^5	x8 (1.6×10^6)
Time step	2 min.	5 min.	÷ 2
Number of time steps	1000	10 000	x 3
Multiplications per column per time step	1000	500	x 2.5
Multiplications/ sec.	1.2×10^6	1.2×10^6	x20 (2.5×10^7)
Speed-up over real time	50-100	50-100	50-100

3.1 STORAGE

It may be tempting to use a backing store for big problems; however, the smaller the array storage the larger is the channel capacity required. In (7) an example was studied of a problem using explicit integration which had 1.5×10^6 variables of average length 20 bits, and was processed on an 8200 PE array with an I/O channel of 10^7 bits/sec. Three formulations of the problem had the following trade-offs: (a) 1850 bits/PE and speed degraded by a factor of 2.5, (b) 2800 bits/PE and speed degraded by 1.3, and (c) 4600 bits/PE, the complete problem in the array and no degradation. A similar problem using implicit methods would have its speed degraded by an order of magnitude if a backing store was used.

This sort of problem needs about $5\text{-}10 \times 10^7$ bits of storage. The falling cost of semi-conductor storage makes this amount of array storage feasible, and the simplicity and reliability of a unified semi-conductor system makes it attractive. Partly for these reasons, the array has more resources devoted to storage than to logic.

3.2 PARALLELISM

Efficiency, defined as the fraction of time a PE is active, depends on programmer skill as well as the problem. Numerical procedures used at present have usually been devised with serial machines in mind, and sometimes a slightly different procedure may be much more efficient.

Explicit methods for the "basic" meteorological equations are efficient. Boundaries do not have much effect because it is usually a case of omitting things. "Secondary" effects may cause efficiency to drop. The computation is different if the air is saturated.

Convection may require the checking of neighbouring vertical layers for stability, followed by a relaxation process. Study indicates that these effects need not have a major effect on the overall efficiency.

Once various conditions have been established "branching" by means of activity bits is very rapid, and can be done frequently in order to improve parallelism. (A conditional branch in a conventional program loop, or selection in a vector machine, are slow by comparison).

Implicit methods involve either ADI (alternating direction implicit) or relaxation methods; the former are not particularly efficient but the latter are.

There seem to be 4 types of grid in use: (a) rectangular for fairly local forecasts; (b) octagonal in overall shape (rectangular neighbour connection) for the northern hemisphere; (c) cylindrical on a global latitude-longitude basis; (d) as (c) except that the number of points on a line of latitude is reduced as the poles are approached. (a) and (c) can fit a rectangular PE array. (b) and (d) would waste some of the PEs. (c) has reduced efficiency because a smoothing process is applied more times near the poles; this can be viewed as a trade-off for the wasted PEs of (d).

3.3 PRECISION AND NUMBER REPRESENTATION

Precision costs time and storage space, so that big problems should use only the minimum consistent with accumulated round-off error being small compared with other errors. Different variables can use different number representations and precisions. Knowledge of requirements is only patchy, but should improve; the pay-off, compared with fairly cautious starting schemes, might be a factor of about 1.5 in storage and 2 in speed.

Meteorology is largely concerned with absolute rather than relative accuracy, and the maximum possible values of variables are well understood; this points to either fractional fixed point or a simple floating point. Block-floating of arrays (9) can also be implemented efficiently.

An example of possible economy in space and speed occurs in explicit integration schemes; the increments to variables require considerably less precision than the full variables.

Careful choice of rounding method in order to avoid bias can also lead to economy (7).

A reasonable estimate of the average precision required for fractional fixed point variables might be 18 bits and rather less for the mantissa of floating point variables.

4. OTHER APPLICATIONS

An algorithm to solve the two dimensional Poisson's equation was studied. It used a Fast Fourier Transform technique, but the extensive data shuffling that this involved occupied only 20-25% of the time. There was also reduced parallelism in places, and a typical PE was idle about 50% of the time. .. On a 72 x 64 PE array, a 256 x 256 mesh was estimated to take 50 msec for 20-bit numbers; this compares very favourably with conventional machines. An interesting aspect is that the main array is held in Array mode and certain row and column features are dealt with in Main Store mode; Main Store mode vectors are combined with the array elements in single arithmetic operations.

For the array to be useful, problems must fulfil three conditions: (a) Processing, as opposed to I/O, must be important; (b) Much of the problem must be programmed with parallel and identical operations (these may, however, be selective); (c) Excessive time should not be spent shuffling data round the array. (In some cases this means the data should be fairly regular).

These requirements are not very severe, and the biggest barrier to widespread use is likely to be in devising an acceptable programming language. (In spite of many problems being naturally parallel, many users are indoctrinated by sequential thinking).

Some applications for array processors are discussed in (5). Further applications are suggested by the fact that the array can be used as an "associative processor"; examples might be air traffic control, graphics processing and symbol processing. Associative information retrieval can look attractive over quite a wide range of parameters; with the associative latch, each PE can scan 1 bit every micro-instruction, and so 10 000 PEs can scan 5×10^{10} bits/second.

The user has the freedom to optimise and experiment from the bit level upwards; this may help him understand his real computing requirements. The array is not arithmetic biased, and the functional flexibility permits functions to be tailored for all sorts of purposes. The hardware simplicity permits parameters such as the number of bits/PE and the type of storage to be varied easily; for example, a slower, cheaper MOS version would extend the range of applications considerably. The array modularity (almost like storage modularity) means that sizes from 500 to 30 000 PEs are reasonable.

ACKNOWLEDGEMENTS

The author would like to thank the Directors of ICL for permission to publish and J.K. Iliffe for his support and for originating many of the ideas. The contribution of A.W. Walton is also gratefully acknowledged.

REFERENCES

1. Barnes, G.H., Brown, R.M., Kato, M., Kuck, D.J., Slotnick, D.L., and Stokes, R.A. "The ILLIAC IV Computer", IEEE Transaction on Computers, C-17, p. 746 (1968).
2. Flynn, M.J., "Some Computer Organisations and their Effectiveness", IEEE Transactions on Computers, C-21, p. 948 (1972).
3. Goodyear Aerospace "STARAN - A New Way of Thinking". A Goodyear Aerospace brochure, Akron, Ohio (1971).
4. Huttenhoff, J.H., and Shively, R.R. "Arithmetic Unit of a Computing Element in a Global, Highly Parallel Computer", IEEE Transactions on Computers, C-18, p. 695 (1969).
5. Kuck, D.J. "ILLIAC IV Software and Application Programming", IEEE Transactions on Computers, C-17, p. 758 (1968).
6. Murtha, J.C., "Highly Parallel Information Processing Systems" in "Advances in Computers". Vol.7, (1966).
7. Reddaway, S.F., "An Elementary Array with Processing and Storage Capabilities", International Workshop on Computer Architecture, Grenoble, June 1973.
8. Slotnick, D.L., Borck, W.C., and McReynolds, R.C., "The Solomon Computer", Fall Joint Computer Conference 1962, p. 97.
9. Wilkinson, J.H., "Rounding Errors in Algebraic Processes", H.M.S.O. London (1963).