

PARALLELISM, PIPELINING AND OTHER FORMS  
OF SYNCHRONOUS MULTIPROCESSING

T. C. Chen

Rapporteurs : T. Anderson  
K. Heron

General Multiprocessing

The term "multiprocessing" is often used loosely, and can be taken to mean simply that many units are operating at the same time.

The simplest class of multiprocessing designs is that involving the overlapped operation of many units (which together form a system, for example a computer system). Since all of the units can be activated and available it seems reasonable to use them concurrently rather than one at a time, so as to eliminate unnecessary queueing and promote overall throughput - all at low extra cost.

Some simple examples are the overlapping of I/O and CPU activity, the overlapping of decode and execution cycles within a CPU, and the simultaneous use of many I/O units such as terminals and sensors.

If multiprocessing involves an overhead but is nevertheless designed into the system, this may be referred to as deliberate multiprocessing design. There can be many reasons for adopting this approach:

To gain speed without changing the technology - more significant since there is already some indication that the trend towards faster circuitry is slowing down. Speed of computation is essential for the solution of certain large problems, such as weather modelling and (of slightly less practical importance) the detection of the onset of turbulence when cream is stirred into coffee.

To lower the cost per module - by spreading the cost of design over high production volumes of components of the same simple design.

To achieve a satisfactory yield of acceptable components and minimise testing. This can be based on a learning curve which in turn can be obtained only when components have a long product life.

To reduce part numbers, and hence costs.

To decrease the number of component interconnections, or at least make them more systematic (interconnection cost being a major factor in LSI design).

To permit incremented upgrading or downgrading by adding or

removing modules. By covering a range in the performance spectrum this can reduce the number of distinct designs needed.

To provide facilities for internal (and external) communication.

To maintain security (by securing parts of the entire system).

To look forward in hope and expectation for the fulfillment of the promise of reliability, availability, and serviceability - not yet achieved in current multiprocessing design practice.

Some examples of deliberate multiprocessing designs are given, using a classification which is not particularly systematic.

- (i) Synchro-parallelism. Many identical modules performing the same task ("SIMD"), for example, Solomon, Illiac IV, DAP, Staran (even the Intel 8080 at the bit level).
- (ii) Pipelining. Many different modules working on the same job stream, for example, Texas Instruments ASC, Control Data Corporation STAR.
- (iii) Computer Networks. Many (possibly different) machines sharing communication facilities.
- (iv) Polymorphic systems. Many CPUs sharing many memory boxes through some sort of crossbar switch, for example, C.mmp.
- (v) Tree structured systems. Hierarchical organisation of many computers under the supervision of the same overall monitor.
- (vi) Loosely-coupled systems. Many and various computing resources with limited communications facilities.

Corresponding to the multiprocessed modules of a system are components of the system which are shared (if nothing is shared then concurrent execution is not termed multiprocessing).

Systems	Multiprocessed	Shared
SIMD	Processors	Decode control
Pipeline	Functional units	Precedence control
Network	Machines	Communication facilities
Polymorphic	CPUs, Memory boxes	Crossbar linkage
Tree	Machines	Monitor
Loosely-coupled	Resources	Communication control (occasional sharing)

### Synchro-parallel (SIMD) machines

The principal characteristic of a SIMD machine is its operation as  $N$  processors which themselves operate in unison on different data. Consider the following trivial example.

$$\begin{array}{l}
 \text{N sources of} \\
 \text{information}
 \end{array}
 \left[ \begin{array}{cccc}
 L & * & + & ST \\
 L & * & + & ST \\
 & & & \cdot \\
 & & & \cdot \\
 & & & \cdot \\
 & & & \cdot \\
 L & * & + & ST
 \end{array} \right]
 \begin{array}{l}
 \text{N sinks}
 \end{array}$$

The instruction sequences are all exactly the same, each being obeyed by a different processor. Only 4 time steps are needed, with N operation being performed at each of the steps. (Little or no interconnections are needed, although the classic example of a SIMD machine, Illiac IV, does have this capability.)

Certain computational problems do appear to be "naturally parallel" in this way, principally those involving the solution of partial differential equations, and those directly based on matrix computations. Other highly parallel programs, such as banking systems and payroll programs, usually include many exceptional cases which limit their exploitation on SIMD machines.

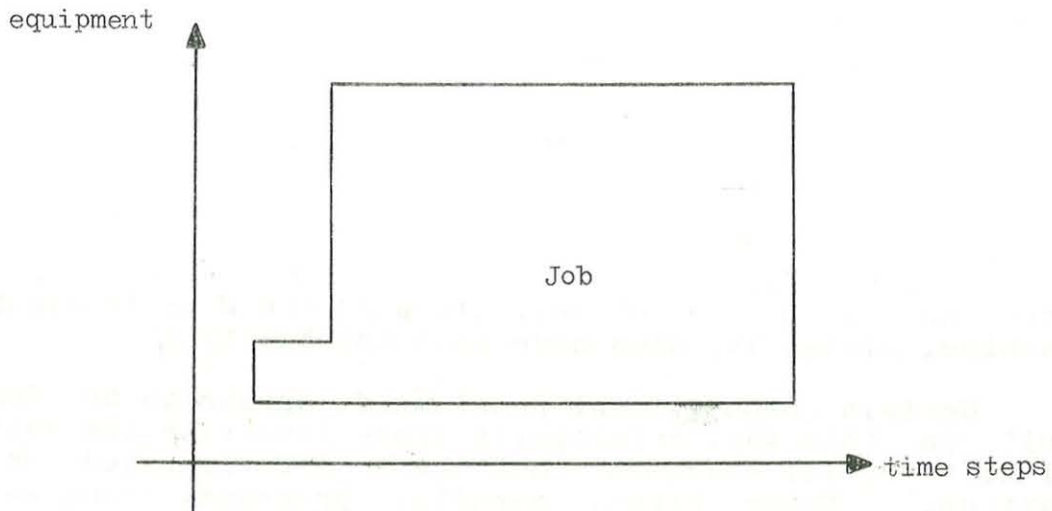
Even for the problems mentioned above there are a number of possible snags. The job width may be too small for the machine, or the job may include a very high overhead. The algorithm employed by the job may be inappropriate. (For instance, when solving the Navier-Stokes equation for the Earth's atmosphere a subdivision into concentric shells is inappropriate. For a machine like Illiac IV a subdivision into a grid of columns of air yields an appropriate algorithm.) The computer execution may have to be non-synchronous because of exception handling. Available language tools may be insufficient. (Certainly fortran is not the right tool. APL is much better but necessitates a great deal of structure redefinition, possibly because of the emphasis on array structures.) Programming costs may be excessive. The UNIVAC report asserted that the ratio of the cost of writing a line of code to that of executing it was 10. Problems can also arise in subdividing a task into parallel components, in communication requirements between these components, and in extra costs due to non parallel overheads before and after the parallel portion of a task.

A consequence of all these potential and actual limitations is that the successful application of synchro-parallel techniques is limited to very large, important jobs which must be performed repeatedly, and requires a careful choice of algorithm with great care being exercised in its programming and debugging.

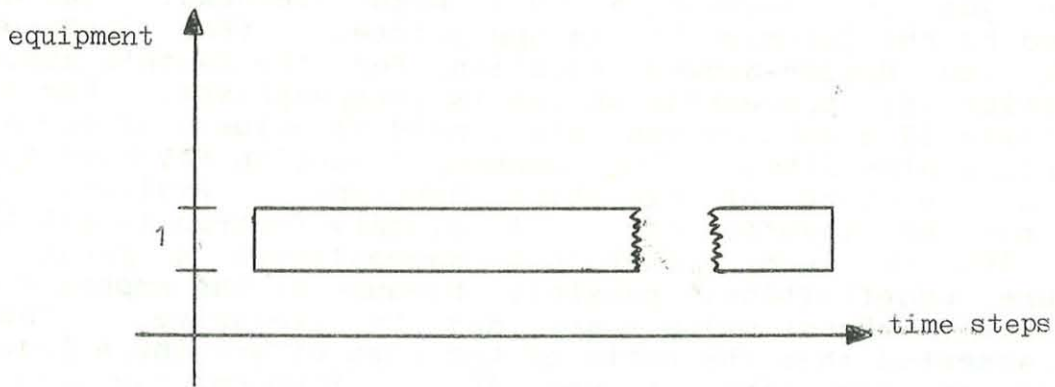
Fortunately there are (at least) two truly parallel and worthy problems, namely weather and wind tunnel computations.

#### A study of synchro-parallelism overhead

Consider the following diagram as indicating a job, which is not quite uniformly parallel, where the vertical axis indicates the degree of multiplicity available in the job while the horizontal axis indicates the number of time steps needed for each of the parallel elements of the job.

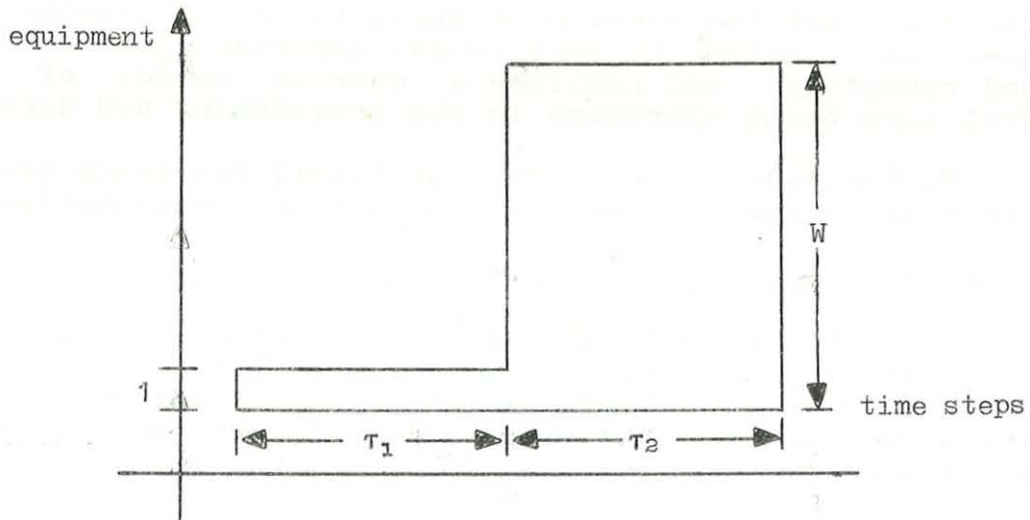


A Fortran programmer must transform the job to fit the uniprocessor provided by the Fortran compiler, thus:



The area of the job in both figures is the same.

To be more specific a parameterisation of the job can be adopted.



The repetition ratio  $\rho$  is defined as

$$\rho = \frac{W\tau_2}{\tau_1 + W\tau_2} = \frac{\text{area of rectangle } \square}{\text{area of whole figure } \sqsubset}$$

Clearly  $0 \leq \rho \leq 1$

If  $W=32$  and  $\tau_1 = \tau_2$  then  $\rho \div 97\%$ .

A number cruncher of multiplicity  $N$  ploughs through the job repeatedly until it has been completely covered. (Visual imagery may be enhanced here by the memory of a dragon with teeth spread  $N$  wide.) The number of sweeps required,  $n$ , will be  $\lceil W/N \rceil$  that is,  $W/N$  rounded to the nearest integer above. With  $W=32$  and  $N=20$ ,  $W = \lceil 1.6 \rceil = 2$  sweeps are needed.

This study will examine the effects of two mismatches - (i) the mismatch of the job to a perfect rectangle (ii) the mismatch of the width of the job to the capacity of the number cruncher.

Now, define performance  $P$  as  $\frac{\text{useful work}}{\text{time spent}} = \frac{\square}{\epsilon t}$

$$P = \frac{\tau_1 + W\tau_2}{\tau_1 + n\tau_2}$$

$$\text{So } \frac{1}{P} = \frac{\tau_1 + n\tau_2}{\tau_1 + W\tau_2}$$

$$= \frac{\tau_1}{\tau_1 + W\tau_2} + \frac{n}{W} \cdot \frac{W\tau_2}{\tau_1 + W\tau_2}$$

$$\text{Now } \rho = \frac{W\tau_2}{\tau_1 + W\tau_2} \text{ and } 1 - \rho = \frac{\tau_1}{\tau_1 + W\tau_2}$$

Hence we have

$$\frac{1}{P} = 1 - \rho + \frac{n}{W} \rho$$

$$P = 1 / (1 - \rho + \rho n / W)$$

The performance function is therefore defined as

$$\varphi(\rho, x) = 1 / (1 - \rho + \rho / x)$$

For  $0 \leq \rho \leq 1$  and  $1 \leq x \leq \infty$ ,  $\varphi$  increases with  $\rho$  and with  $x$

$$\begin{aligned} 1 &= \varphi(0, x) \leq \varphi(\rho, x) \leq \varphi(1, x) = x \\ 1 &= \varphi(\rho, 1) \leq \varphi(\rho, x) \leq \varphi(\rho, \infty) = \frac{1}{1-\rho} \end{aligned}$$

$$\frac{\partial \varphi}{\partial x} = \frac{\varphi^2 \rho}{x^2} = \frac{\rho}{[(1-\rho)x + \rho]^2}$$

$$0 \leq \frac{\partial \varphi}{\partial x} \leq \rho$$

The growth of  $\varphi$  with respect to  $x$  diminishes as  $1/x^2$ . Various upper bounds can now be placed on the performance  $P$ .

$$P = \varphi(\rho, W/n) = 1/(1 - \rho + \rho n/W)$$

$$(W/n \leq W) \quad P \leq \varphi(\rho, W) = 1/(1 - \rho + \rho/W)$$

$$(\rho \leq 1) \quad P \leq \varphi(1, W) = w$$

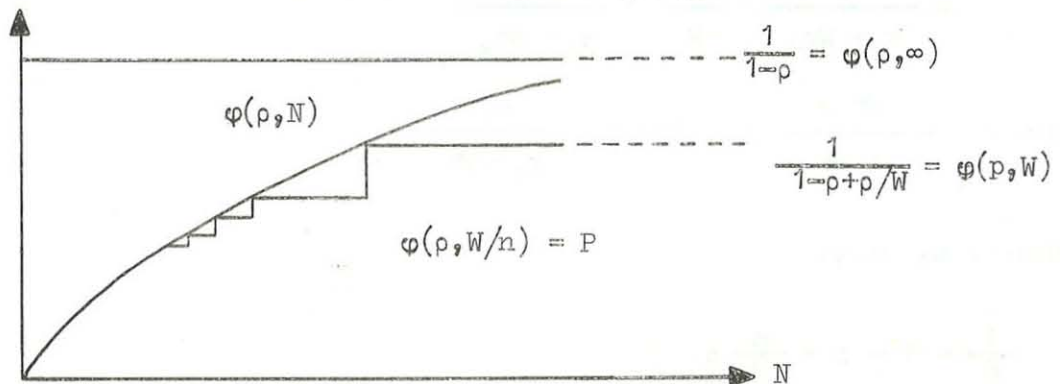
$$(W/n < \infty) \quad P \leq \varphi(\rho, \infty) = 1/(1 - \rho)$$

$$(W/n \leq N) \quad P \leq \varphi(\rho, N) = 1/(1 - \rho + \rho/N)$$

$$(\rho \leq 1) \quad P \leq \varphi(1, N) = N$$

It seems eminently reasonable that the performance can never exceed the job width  $W$  or the multiplicity  $N$  of the number cruncher. But in fact  $P$  is also limited by the nature of the job (regardless of  $N$ ) to be at most  $1/(1 - \rho + \rho/W)$  which in turn is bounded by  $1/(1 - \rho)$ .

These bounds are illustrated in the graph below.



As  $N$  is increased, the performance increases in a step-wise manner because of its dependence on  $n$  which depends on the divisibility of  $W$  by  $N$ . The maximum possible value of the performance is reached when  $N=W$  since then  $n=1$ . This maximum is the reachable bound  $\varphi(\rho, W)$ . A good continuous approximation to  $P$  is given by  $\varphi(\rho, N)$ .

While  $P$  can never reach the bound  $1/(1 - \rho)$  it may reach  $0.5/(1 - \rho)$ , when  $x = W/n = \rho/(1 - \rho)$  (for  $\rho \geq 0.5$ ).

$$\begin{aligned} \text{Then } \varphi(\rho, W/n) &= 1/(1 - \rho + \rho n/W) = 1/(1 - \rho + 1 - \rho) \\ &= 0.5/(1 - \rho) \end{aligned}$$

At this point, the rate of growth of  $\phi$  has the value

$$\frac{\partial \phi}{\partial x} = \frac{1}{4\rho} \leq \frac{1}{2}$$

The addition of an extra processor at this stage will at most give the benefit of only a half processor.

This little study is rather sobering. Continued investment in processing capacity for a particular task will not in the long run yield even a fractional return, since there is a strong unreachable bound which can be stated in terms of the characteristics of the job alone. Perhaps we should be satisfied with attaining 50% of that performance bound since that is usually possible.

Some partial solutions, of an engineering nature, are possible. The overhead involved in processing the non-parallel portion of the job ( $\tau_2$ ) may be overlapped with a separate processor. Incorporation of this feature in Illiac IV (finally achieved this year) produced a performance improvement factor of more than 2. Flexible modules could be designed to handle unusual events. As a further benefit, sufficient flexibility may allow modules to be slightly out of step - relaxing the requirement of strict synchronism. Centralised decoding can be a mixed blessing because of the multiple fan-out needed from the central resources (in the case of Illiac, 64 points must be reached simultaneously). Fan-out can be reduced by local decoding, and also by systematic time lag.

Jobs should be carefully selected to fit the machine. With  $\rho > .99$  and  $W > 100$  the performance bounds are less constraining since then  $1/(1 - \rho) > 100$  and  $1/(1 - \rho + \rho/W) > 50$ . Other guidelines are: to be content with achieving  $P = 0.5/(1 - \rho)$ , to choose  $N$  as a divisor of  $W$ , and to be prepared to fine tune a program, guarding against hidden overheads.

A brief summary of the Illiac IV machine may be of interest. It comprises 64 processing elements (PEs) in an 8x8 array. Parts of the machine can be disabled and withdrawn from a computation. There is a broadcast capability (for instructions and data) to all 64 PEs. Flexibility is provided to a PE by means of local memory with local indexing. Provision is made for communication traffic to neighbours in the 8x8 array (left, right, front, back). Floating point operations are available. The design follows the line of development of Coche and Kochen (1958) and Solomon (1962). Illiac IV is operational in Mountainview, California and is running better every day.

## Lecture 2 - Pipelining

Put succinctly, pipelining is time-synchronised division of labour along a processing path. To illustrate this consider figure 1 in which a processing segment  $S_i$  has data  $b_{i-1}$  entering and data  $b_i$  emerging. Data  $b_i$  emerges after a time  $\tau$ , and work  $W_i$  has been done in the segment. (Note that the term work is used loosely: it is not necessarily the same term used in physics). The graphic approach used in figure 1 illustrates the result of linking

segments together (assuming only a common cycle time  $\tau$  for the result to pass through a segment). A steady state is reached when all the segments are occupied. Then, the performance, defined as work per cycle is  $\Sigma w_k$ .

There are of course some limitations to be placed on pipelines and several of these are touched on, briefly, below.

Speed of Light For a pipeline of length  $D$ , the maximum speed at which data can travel from the input to the output is that of the velocity of light, and the time taken is minimally  $D/c$ . If the pipeline consists of  $M$  segments and cycle time is  $\tau$ , then it follows

$$M \geq D/c \text{ or } \geq D/Mc$$

Time can obviously be very small, but the important point to note is that it is the product  $M\tau$  which is limited, not  $\tau$  itself.

Quantum Theory The Heisenberg uncertainty principle imposes limitations on the precision with which a quantity can be measured at a given instant.

$$\Delta\tau \Delta E \geq \hbar (= 10^{-27} \text{ erg-secs.})$$

Arbitrarily small means arbitrarily small which in turn means arbitrarily large, and an arbitrarily large energy reservoir  $E$ . Speed equates to high energy.

Atomicity Limitation It is difficult to conceive a pipeline segment smaller than the diameter of a hydrogen atom. Hence for a segment length  $d$ ,

$$d \gg 10^{-8} \text{ cm.}$$

Again with the velocity of light as the limiting velocity at which data can pass through a segment, the cycle time limitation is

$$\tau \gg \frac{d}{c} \approx \frac{10^{-8}}{3 \times 10^{10}} \approx 0.3 \times 10^{-18} \text{ sec.}$$

There are still many orders of magnitude in hand over today's technology.

Engineering Limitations Within a segment, latches and logic elements are required to work on the data stream and to synchronise it to the next segment. These physical elements introduce time or "logic" delays. However, Hallin and Flynn (1972) have shown that for arithmetic functions using combinatorial circuits, and a latch devised by J. Earle, no additional delays are entailed if the cycle time is equal to four or more logical delays - the latching delay is overlapped fully.

### Pipeline Processing of Jobs

Figure 2 represents a sequence of similar jobs being fed through a four-segment pipeline. After four cycles job 1 emerges from the pipeline after work ( $W1+W2+W3+W4$ ) has been performed on the



data. Figures 3, 4 graphically represent the work done on the jobs, the latter figure representing the redistribution of work (represented by area) to facilitate calculation.

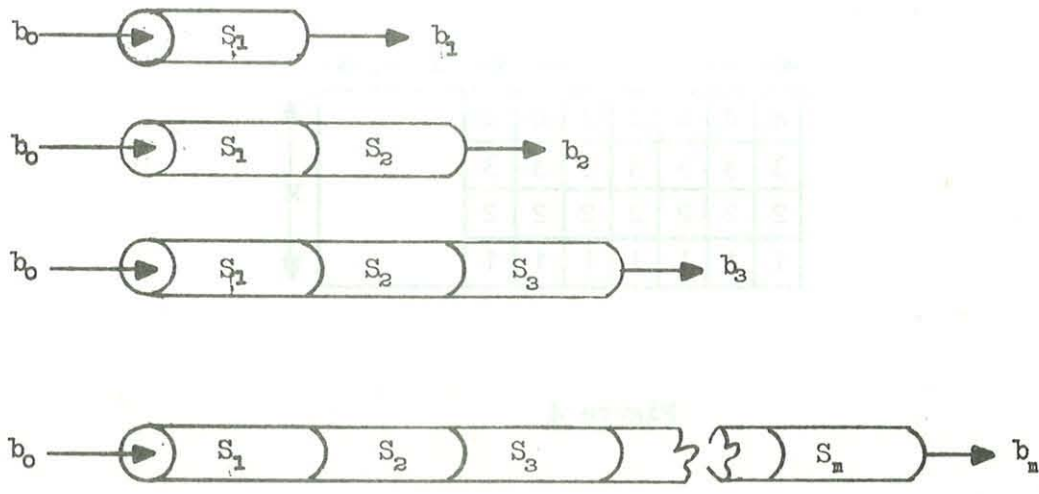


Figure 1

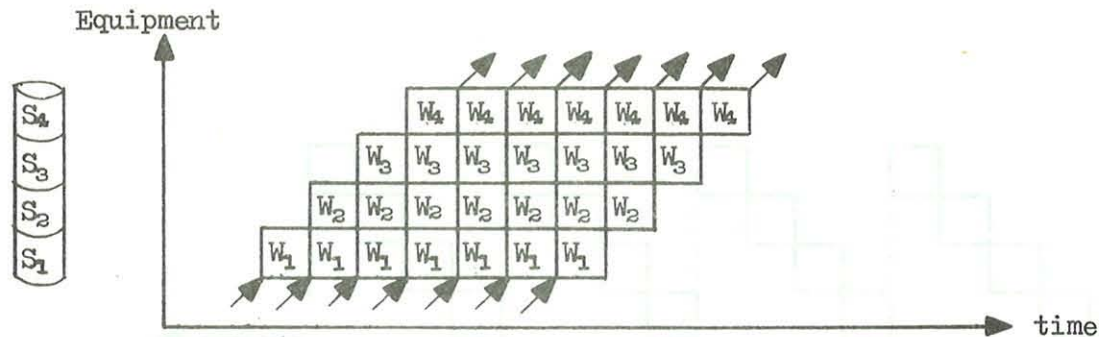


Figure 2

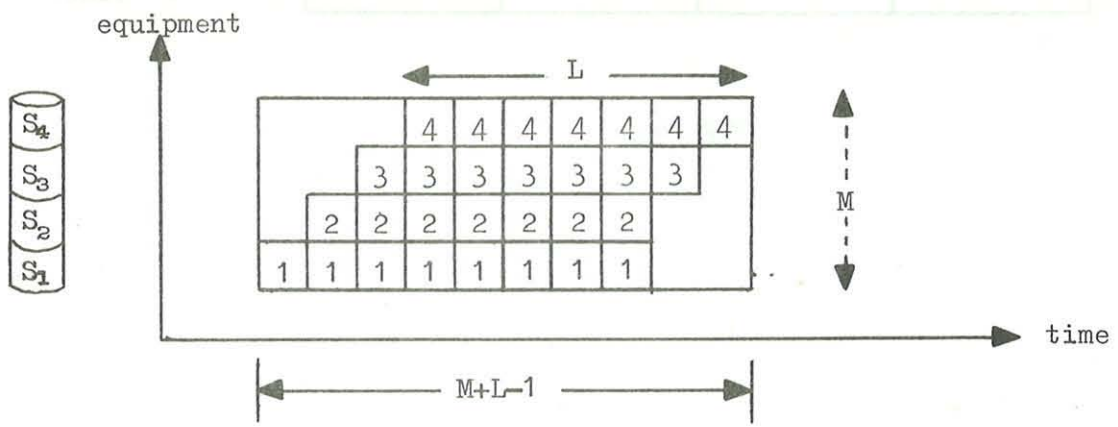


Figure 3

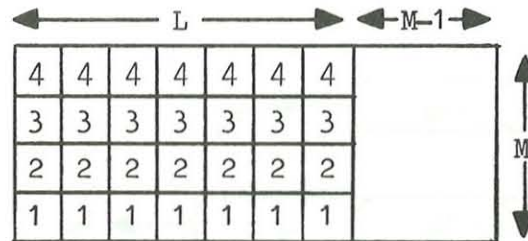
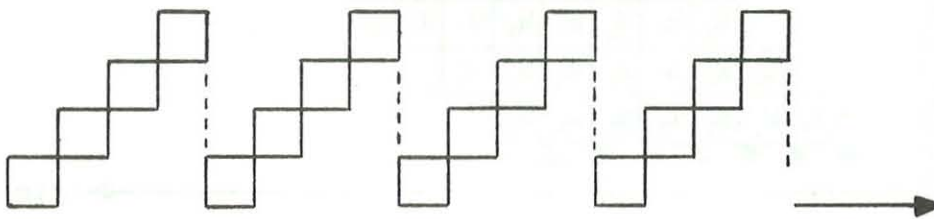


Figure 4

Equipment



same as



Figure 5

Then useful work = LM,  
 and time spent in doing it = L+M-1 cycles  
 Then defining performance  $P = \frac{\text{useful work}}{\text{time spent doing it}}$

$$P = \frac{LM}{L+M-1}$$

$$= M \left[ 1 - \frac{M-1}{L+M-1} \right]$$

The fractional loss of performance power

$$= \frac{M-1}{L+M-1}$$

Once the last job has entered the pipeline, it gradually empties with succeeding cycles - it drains. As the number of contiguous jobs in a batch submitted to the pipeline increases (represented by L) the fractional loss of performance, due to draining of the pipeline, grows smaller.

$$\text{Fractional loss} = \frac{(M-1)}{2} \quad \frac{2(M-1)}{3} \quad \frac{4(M-1)}{5} \quad \dots \quad \frac{k(M-1)}{k+1}$$

The moral is clear: try to extend L, that is, keep the pipeline full. In consequence, the pipeline should not be "normalised" unnecessarily. (Normalised used here to mean draining the pipeline completely before entering new data). The sort of error condition which might be met by normalising the pipeline is floating-point overflow, but even in this case it is possible to think of avoiding normalising the line by labelling the "over flow data" as being in error and processing later.

### Cost of Engineering the Pipeline

The cost of installing latches (simple forms of memory) and logic in pipelines was in earlier days, enormous, but the present day technology means that the cost can now be trivial. (Back in the days of STRETCH, the cost of the CPU was reckoned to be proportional to the number of register bits in the CPU.) There is, however, the hidden cost of finding the right algorithm to permit equal time division - really an art. Sometimes it is necessary to smuggle in something else, namely synchro-parallelism.

The pipeline must be well utilised. If the frequency of use is low, the pipeline is under-used. This is graphically illustrated in Figure 5. It is clear that with such a frequency of use, a simple non-pipelined processor is just as good. Recalling the definition of performance

$$P = \frac{LM}{M+L-1}$$

$$\text{If } L = 1/M, P = \frac{LM}{M+1/M-1} = \frac{M}{M^2-M+1}$$

In the case of the under-used four-segment pipeline of Figure 5,  $P = 4/13$  but in the case of the non-pipelined system,  $M=1$  and  $P=1$ .

Another concern in the cost of pipelines is the fact that pipeline segments are special purpose, and this may involve high development costs. Today one prefers systematic construction, each bit the same. The classical form of pipeline seems to violate this. Perhaps the non-classical form will do better.

### Synchro-Parallelism and Pipelining

Table 1 sets out to compare parallelism (exemplified by a Single Instruction Multiple Data stream mechanism, SIMD) and a pipeline. To permit closer comparison of "performance",  $p$  is defined by  $p = 1 - L$ . (This  $p$ , an effective repetitious ratio, was thought at first to be important, but later it was decided that it had little implication, except for comparing formulae). As can be seen from the table, more performance bounds can be placed on the SIMD mechanism than on the pipeline. Nevertheless they share certain attributes. Both are tightly-coupled multiprocessor systems and, as will be brought out later, they can complement one another. Both are now easier and cheaper to implement using recent developments in electronics - Large Scale Integration.

They both suffer from loss of performance worries when not kept fully busy, but the solution is to make the modules flexible - about this, more later.

### Some examples of how Synchro-Parallelism and Pipelines are compatible and indeed complementary

The first part of Figure 6 shows a pipeline segment with inputs  $a$  and  $b$ , and outputs  $c$  and  $d$ . The second part of the figure shows the crosslinking of the pipeline segments to give three horizontal pipelines operating in synchro-parallelism and four vertical pipelines also operating in strict synchro-parallelism. It is not clear whether this is a synchro-parallel design or a pipeline design, but the answer is that it does not really matter.

The next figure, 7, shows a solution to the problem of a pipeline segment which has a cycle time four times that of the other segments. Rather than slow down the cycle time for all segments to match that of the "long" segment, four such long segments are harnessed in parallel. Output from the preceding short segment is cyclically presented to each long segment input. The outputs are cycled again into the next short segment. Hence the first job to pass down the pipeline will take four cycles to appear at the output of the paralleled long segments, but subsegment jobs will appear one per cycle thus maintaining throughput rate of one per cycle.

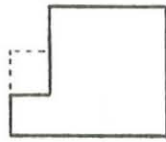
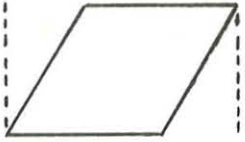
	 "SIMD"	 pipeline
Tightly coupled	YES	YES
Multiplicity	N	M
Modules	Processing Elements, no decode	Special purpose
Cost	Lowered by mass production	Latch insertion is cheap
Performance	$\frac{1}{1 - \rho + \frac{\rho n}{w}}$ <p>n is number of sweeps            ρ is            w is</p>	$\frac{ML}{M+L-1}$ <p>Defining <math>\rho = 1 - \frac{1}{L}</math></p> $P = \frac{1}{1 - \rho + \frac{\rho}{M}}$ <p>M is number of segments            L is number of jobs            through pipeline</p>
Performance Bounds	$\frac{W}{N}$ $\frac{w}{n}$ $\frac{1}{1 - \rho + \rho w}$ $\frac{1}{1 - \rho + \rho N}$ $\frac{1}{1 - \rho}$	$\frac{L}{M}$

Table 1 : Synchro-parallelism and Pipelining

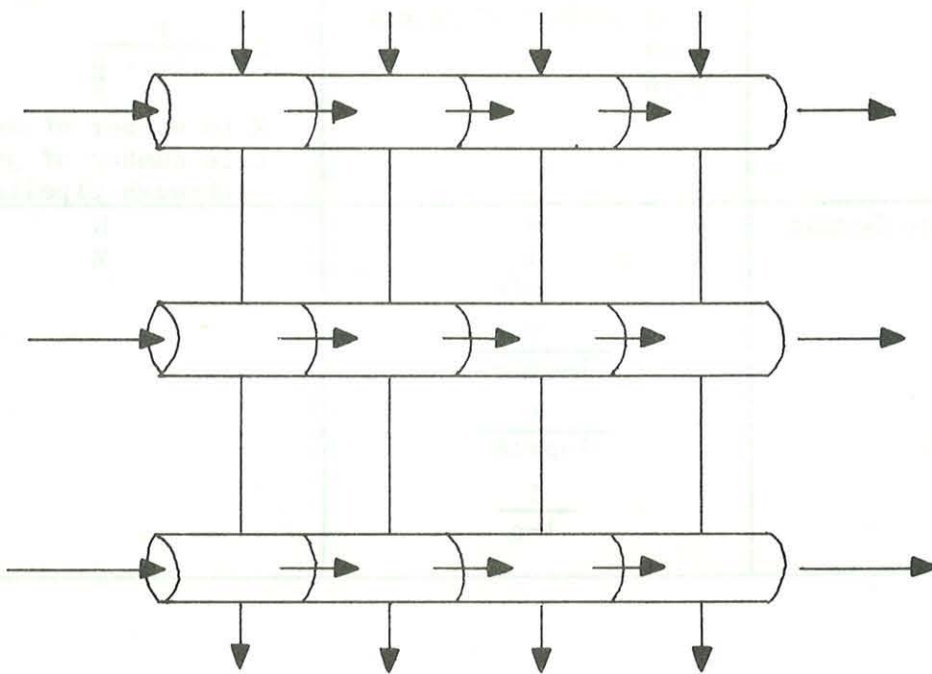
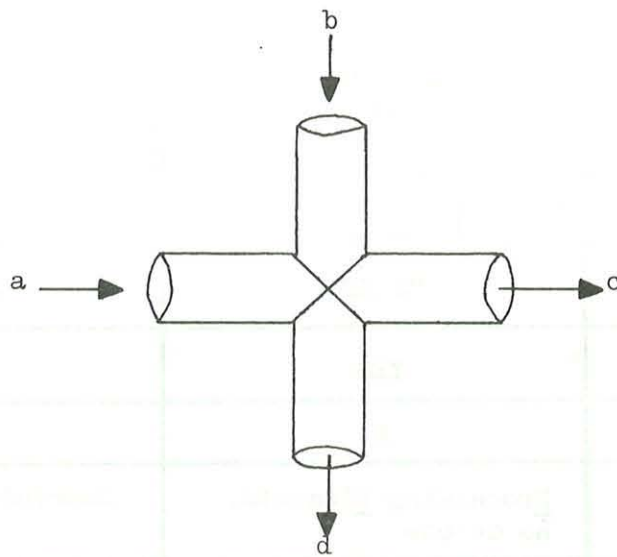
Cross Linking in Pipelines

Figure 6

The next figure shows a Wallace multiplier. Three operands through a carry save adder are squeezed into two. After the first stage of Figure 8, nine words become six words. The six outputs report to two elements in the second stage and are reduced to four words. A delay is introduced in the third stage to carry an operand to the final stage to combine with the other two operands. Thus a nine-number problem is reduced to a two-number problem. Each stage could be considered as a pipeline segment, or the whole unit could become a pipeline segment. In the latter case a huge multiplier tree could be built up, and in principle, the fastest-multiplier can be made in this way. The compression ratio for the Wallace multiplier is  $3/2$  or 1.5. One can progress by adding seven words together and compressing them into three (compression ratio better than 2). This is possible because adding the  $i$ th bits of each of the seven words results in a maximum sum of 7, which means that any such issue would be represented by at most three bits. Hence 7 words compress to three, and it follows that 15 words compress to 4 etc.

Figure 9 shows the VAMP design (VAMP for Vector Arithmetic Multi Processor) described by Senzig and Smith. The interleaved memory is under bus management and the "processors" are just registers plus a few bits, which can be masked "off". Arithmetic is performed in a common pipeline with staggered sequencing of jobs to fit the pipeline, and the jobs are returned to the originating "processors" or registers. Because of the sequential element in the data stream (the pipeline), this design should really be called a Serial Instruction Virtual Multi Data stream (SIVMD) rather than on SIMD. However, this approach may well present the best SIMD - pipeline compromise.

The section below the solid line in Figure 10 represents the virtual processors. The instruction enters the first "processor" A, and data is sent to the pipeline. The instruction then passes to B and B sends data to the pipeline while A's job passes into the second segment of the pipeline etc. It is worth noting that disabled "processing elements" do not withdraw performance from the arithmetic unit, because, unlike the Illiac IV scheme, the arithmetic unit is not masked off when the processing element is masked off.

#### Pipeline Summary and conclusions/proposals.

By judicious installation of latches (for time synchronisation) it is possible to achieve high throughput performance. The cost of these latches is relatively low but the segments are still special purpose. It is not always possible to install a pipeline in a given situation.

Pipelines, far from rivalling, complement and are complemented by synchro-parallelism. Certainly pipeline networks need more study. The biggest problem of pipelines, pipeline draining, can be remedied by multiprogramming in the small, having segments sufficiently flexible to avoid emergency drains and perhaps by allowing local autonomy under time pulse constraint.

One way of avoiding some of the development costs of special segments might be to use a microprocessor to permit

"personalisability".

Paralleling a long Segment in a pipeline

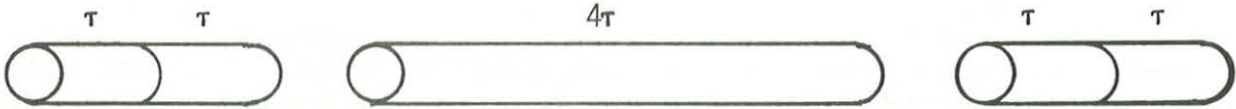
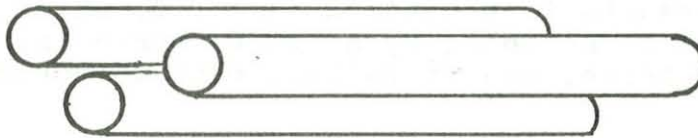


Figure 7



Throughput maintained at one job per cycle

The Wallace Multiplier

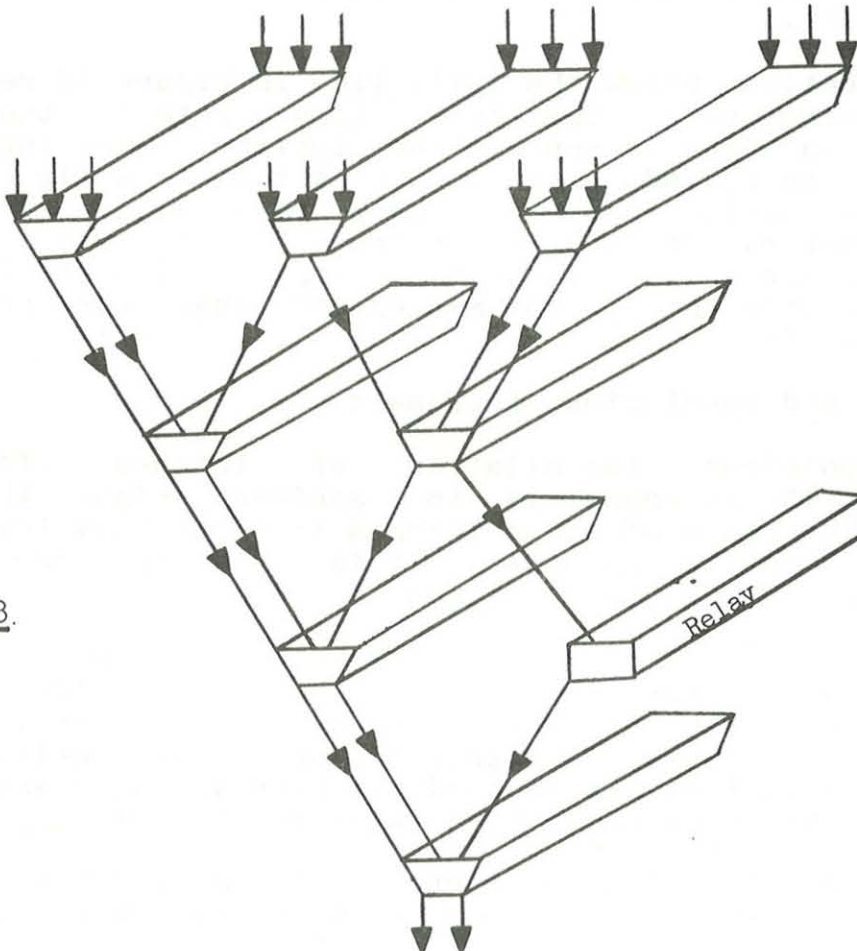


Figure 8.



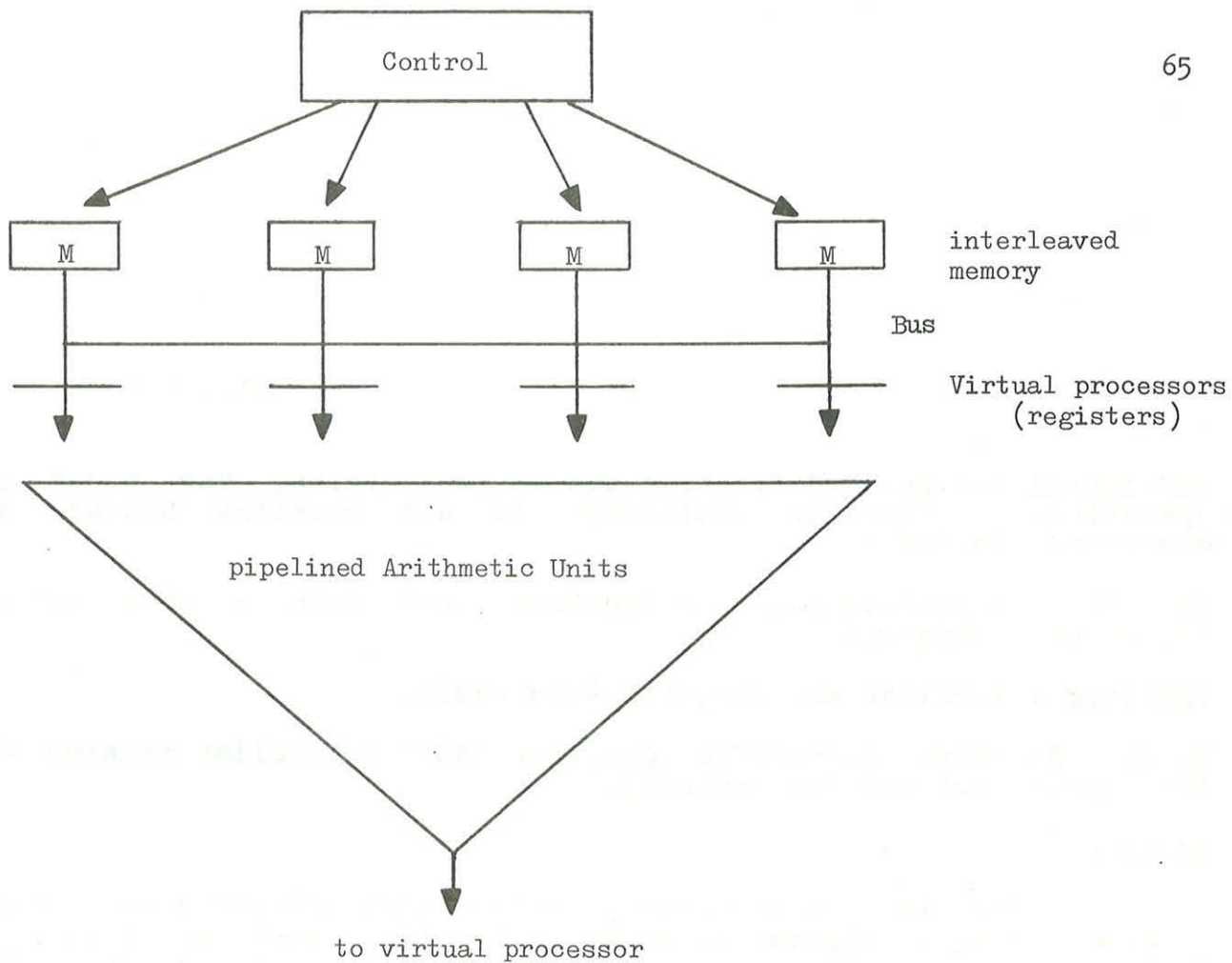


Figure 9

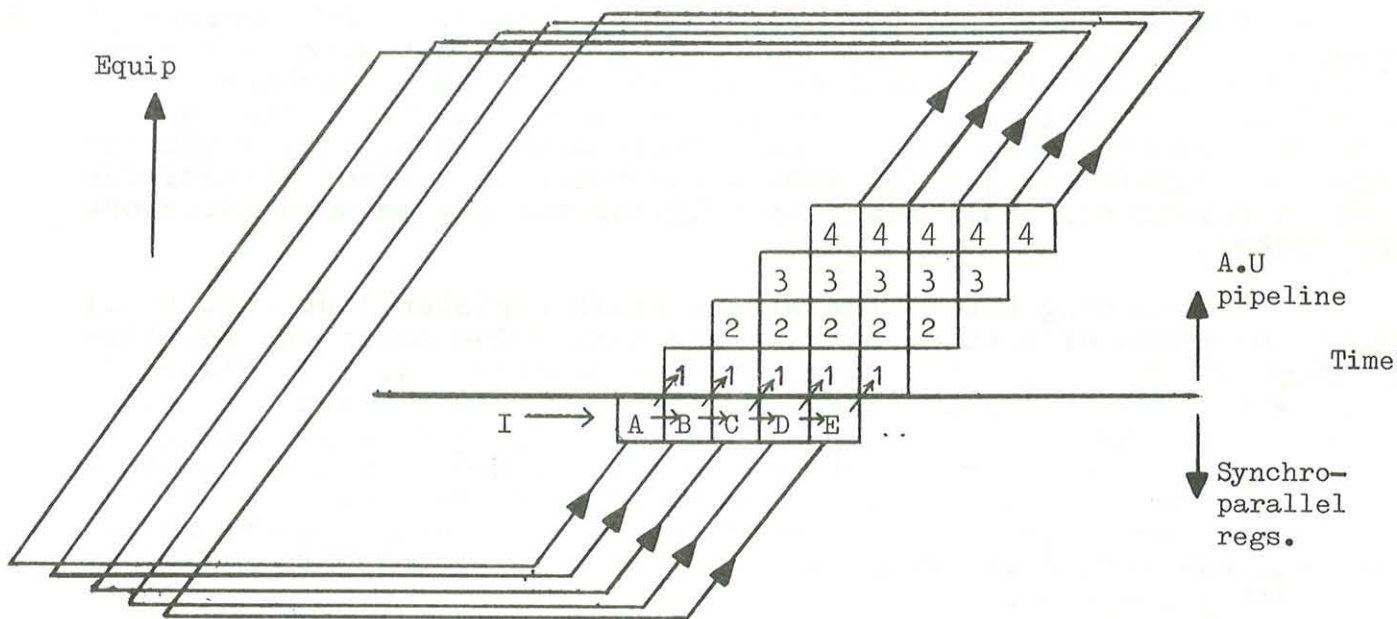


Figure 10

A few important machines which did or do incorporate pipelines are listed below. (The name pipeline probably arose in Project Stretch, IBM 195X)

IBM Stretch : It is not known as a pipeline machine but it had lookahead, a four-segment pipeline and a variable field length unit with pipelined data access.

IBM 7950 : Pipelined access, monitoring, processing but only one byte wide.

IBM 360/91 : Pipelined floating add in two cycles, one cycle per operation. Floating multiply is not pipeline because the occurrence is small.

CDC 7600 : Pipelined multiply duration,  $D=5$  with a rate of two cycles per operation.

CDC Star : Pipeline machine with 40ns cycle.

T. I. Advanced Scientific Computer (ASC) : Pipeline machine with 60ns cycle, but now discontinued.

### Lecture 3

The simplest pipeline possible can be pictured as a string of bits riding a railroad as shown in the first part of figure 1. At time  $t_0 + \tau$ , the bits have advanced one place to the right. This of course is the familiar shift register and is the form of the bubble devices and charge-coupled devices now becoming so important. If the shift register is fed back to itself it becomes a loop, a memory. Such closed-loop registers are reminiscent of disks and are sometimes called electronic disks. If one were to characterize a conventional disk briefly one could say that it was (i) cheap per bit, (ii) had a long access time, (iii) inflexible loop size and (iv) a large package (say 10 inch diameter disks). By comparison "electronic" disks are (i) costlier per bit, but with (ii) short access time, (iii) flexible loop size and (iv) small package size. Also magnetic-bubble devices are just as non-volatile. The cost of such devices is coming down - Texas Instruments is selling a 92k-bit magnetic bubble device for \$400 - and there is a place already for them in relatively small memories (filling the gap between mainstore and disks).

Returning now to the simple shift register loop - it still lacks the means of getting data in and out. One solution to this problem is the special switch, shown diagrammatically in figure 2. This switch has two inputs A and B, and two outputs C and D. (The switch was designed for bubble systems where tracks can cross.

The switch can assign inputs to outputs in two possible ways - shown in figure 2. A can be assigned to C and B to D. This is the "avoidance" switch condition or "off" condition. In the second switch condition, A is connected to D and B to C, the "crossover" or "on" switch condition.

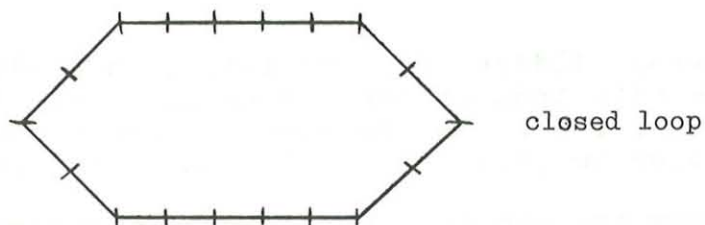
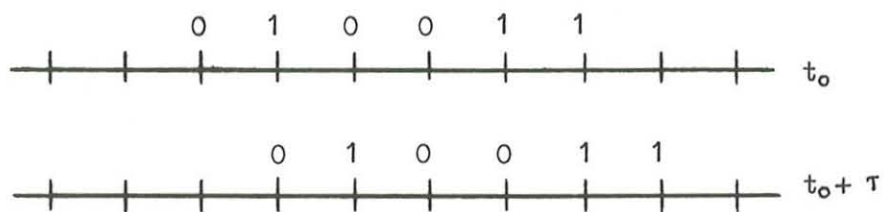
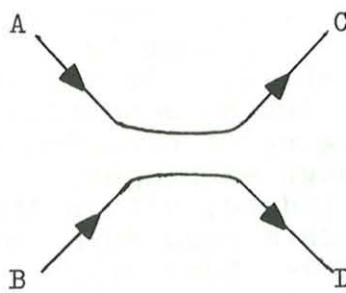
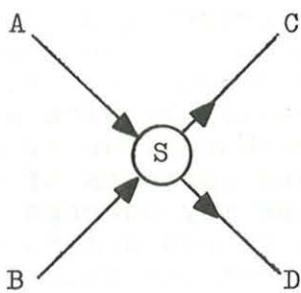
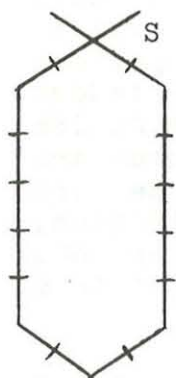
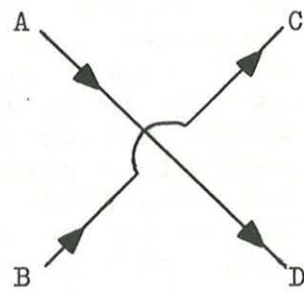


Figure 1

Figure 2



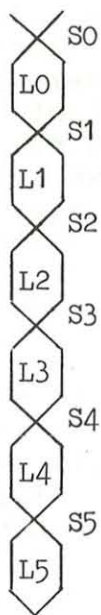
Avoidance condition (off)



Crossover condition (on)

Figure 3

The Ladder



These switches can be incorporated as linking mechanisms between shift register storage loops. Shown in figure 3 is a set of six loops connected by five switches S1 to S5, with a further switch S0 serving as an input/output switch. Such an arrangement has been entitled a "ladder".

It will be observed that to get information in or out of loop 5 takes the longest time, while for L0 it takes the least. (As an aside - there is a problem with real bubble devices, that to convert from bubble to electrical signal for the outside world requires a lot of chip area, so one cannot afford too many such I/O areas.)

The general ladder has N loops, each with 2M bits - a symmetric structure with arms of equal length. One may define a macro cycle for the ladder as the time to rotate one bit around a loop from point A back to point A. (Also call it a period.)

To load say the nth loop can take n macrocycles. First loop L0 is filled. Switch S1 is set to "crossover" or "on" (from default of "avoidance") and L0 and L1 are exchanged. L1 and L2 can be exchanged and the process may proceed to any depth in the ladder. In this way N records may be loaded into this pipeline-with-steering-switches. Each record for these purposes is of 2M bits and each occupies a loop. If all switches are set off, data will be trapped in the loops because data reaching a switch will be connected back to its own loop by the switch. The data pattern will repeat every period. If all switches are set off, the ladder is said to be in the IDLING condition. Obviously with a mechanism which allows exchanging of records, all permutations of records are possible - given enough exchanges. (Record contents of course are not shuffled and indeed, within the loops may undergo reflection, but no matter where data records have been stored and no matter how many reflections have been experienced when the data emerges from the I/O switch it is as it was when it went in).

One can consider the ladder system as a basis for a multilevel storage hierarchy. In figure 3, the loop on top, L0 enjoys the easiest access, while L5 has the worst. The principle of storage management says that if one wants to get something from the middle, say L4, one should get it and move everything else down one notch. This is the most-recently-used demotion in a two-level hierarchy. Dynamic storage management says that loop labels are really indicators of depth of loop. If the record from loop 4 is needed it is necessary to perform what is known as "topping" from loop 4 to the top. This is a permutation operation and, therefore must be capable of being done with the system described. Figure 4 demonstrates how a record "E" can be brought into the top loop (occupied initially by "A"). At time t, the switch between "E" and "D" is changed to "crossover" and E and D begin to exchange. One period later at t + T, the next switch up the ladder is changed to "crossover" and the first switch returned to "avoidance". At t + 2T the next switch pair changes, and so on. After four periods, the record E is trapped in the top loop, and all records formerly in loops above E have been demoted by one level. Topping from a depth k (to a depth 0) takes exactly k periods by exchange. However, by a single extension of the exchange idea it is possible to improve on k periods.

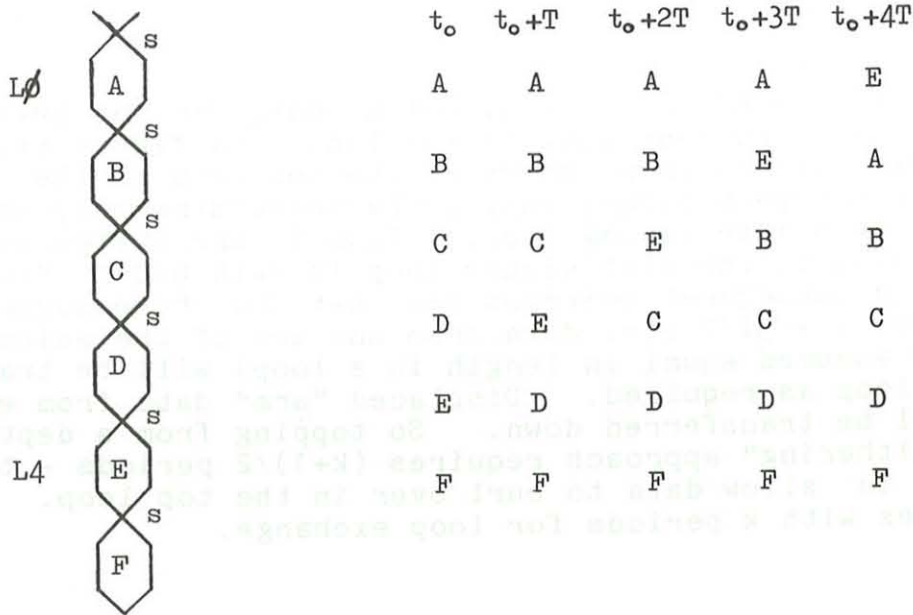


Figure 4 Topping from depth  $k$  to depth  $\phi$

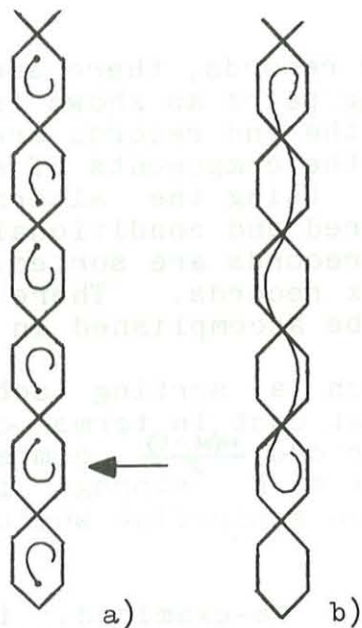


Figure 5 Slithering

### Slithering

It is reasonable to require that data in a buried loop should take the shortest path to the top. In figure 5b, data in one arm of the selected loop can reach the top loop in the shortest time by taking the path shown, that is by traversing only one of the two arms of each intervening loop. Thus if the switch connecting the selected loop to the next higher loop is switched to "crossover" at time  $t$ , and subsequent switches are set to "crossover" at  $t + T/2$ ,  $t + 2T/2$ ,  $t + 3T/2$  etc. data from one arm of the selected loop (the two arms assumed equal in length in a loop) will be transferred to the top loop as required. Displaced "arm" data from each loop traversed will be transferred down. So topping from a depth of  $k$  using the "slithering" approach requires  $(k+1)/2$  periods - the extra half period to allow data to curl over in the top loop. This of course compares with  $k$  periods for loop exchange.

### Sorting

Having got this far with storage management one is tempted to try to increase the operations which can be done. For example, why not attempt something else which is of a permutation nature, namely sorting. Sorting can consume 25% of a large machine's time - a quotation from Knuth. Classical sorting time is proportional to  $N \log N$  with  $N$  records and one machine. Now, with the possibility of multiprocessors giving simultaneous compare and moves, sorting time might be made more linear with  $N$ . As has already been shown, data movement can be achieved with the ladder by the appropriate setting of switches, at the correct times. But what sets the switches at the right times? The bubble memory described is not up to the job so it probably requires a microcomputer (packaged in close proximity to the information). However, before describing the details of such microcomputer control, it is appropriate to mention the Odd Even Transposition Sort (OETS).

Starting with a set of  $N$  records, there are two ways of grouping these records by neighbouring pairs as shown in figure 6. It may happen that one or both of the end records are not paired. Starting from either pairing scheme, the components of each pair are compared and conditionally permuted. Using the alternate pairing scheme, the pairs are again compared and conditionally permuted. This process is repeated until the  $N$  records are sorted. Figure 7 shows an example of the sorting of six records. There is a theorem which says that complete sorting can be accomplished in  $N$  stages.

It would appear that such a sorting scheme could be implemented as a pipeline, but at great cost in terms of hardware. Regardless of whether  $N$  were even or odd,  $\frac{N(N-1)}{2}$  comparators would be required by such a pipeline as well as  $N$  storage positions,  $N$  input ports and  $N$  output ports. Such a pipeline would be too good and just too expensive.

If the example of figure 7 is re-examined, it is clear that the sort has repeated sections. An even/odd sort pair is followed by an even/odd sort pair followed by an even/odd sort pair. This suggests the possibility of using one section three times (three for this specific example). This folding back would

increase the overall cycle time when compared with the full pipeline but it would reduce the hardware by roughly a factor of three. Figure 8 shows this pictorially. Figure 9 shows only the transpositions of figure 8 and it immediately suggests the further folding or telescoping of figure 10. In the latter, switch sets "a" and "b" are activated in alternate cycles. Hence an OET sort can be implemented by using a single set of switch elements N times, taking care only to activate alternate switch sets each cycle. Such a scheme closely resembles the ladder described above, with its storage loops (latches) connected by avoidance/crossover switches. With the ladder, however, data is loaded via the "top" storage loop instead of directly into the latches as envisioned above (figures 11 and 12). This introduces an overhead in time not present in the pipelined sorter outlined above. Nevertheless it is possible to compare directly the pipelined scheme and the ladder scheme (delaying for now the explanation of how the necessary comparisons and switch make/break decisions are made). Table 1 draws comparisons.

Note the great reduction in hardware in the case of the ladder even though a sort can take the same time in each case (with the pipeline, throughput is N times).

It is appropriate now to turn to the problem of a possible comparison and switch-setting mechanism. This mechanism might well be implemented with a microprocessor, packaged in close proximity to the ladder. Such a scheme is shown in figure 13. Data is loaded into the ladder and "keys", corresponding to the ingoing data records, are deposited with the microprocessor. The processor permutes the keys based on an odd/even transposition sort, and generates a number of inputs to a status register. The status register bits control the settings of the avoidance/crossover switches of the ladder. Thus in principle the records in the ladder can be sorted - and in linear time. Of course the processor needs to be faster than the record movement in the ladder, something like a factor of N faster. Although the "keys" should be as small as possible, the associated records can be as large as need be since there is nothing to prevent parts of records being held in parallel ladders, under common status-register control.

A somewhat embarrassing problem is raised by this ladder/CPU sorter and that is that the OET sort can be achieved in N periods using exchange (and  $\frac{N+1}{2}$  using slithering) but input takes N periods by exchange and output N periods by exchange. Hence the sorting, embarrassingly, is too fast taking less time than input/output. The answer may be that in the future, the data may already be in memory eliminating input/output, or, the sort may be hidden in the input/output time - a zero-sort-time situation.

Odd Even Transposition Sort (OETS)

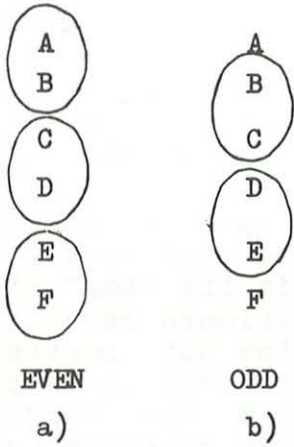


Figure 6

OETS Example

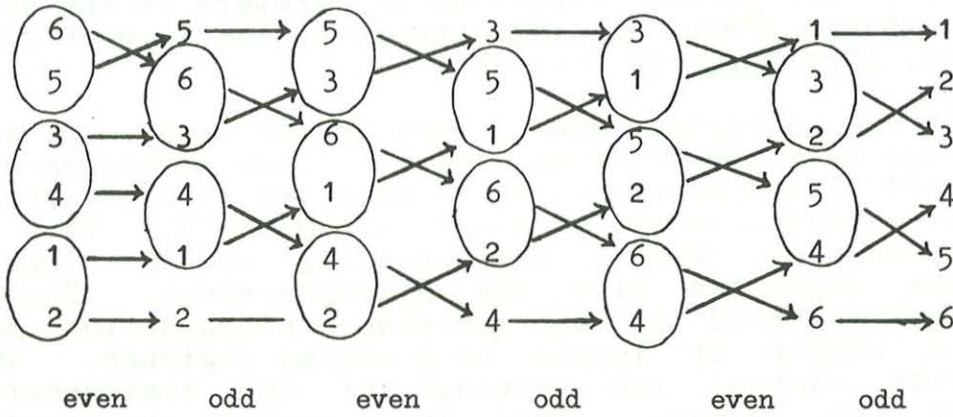


Figure 7

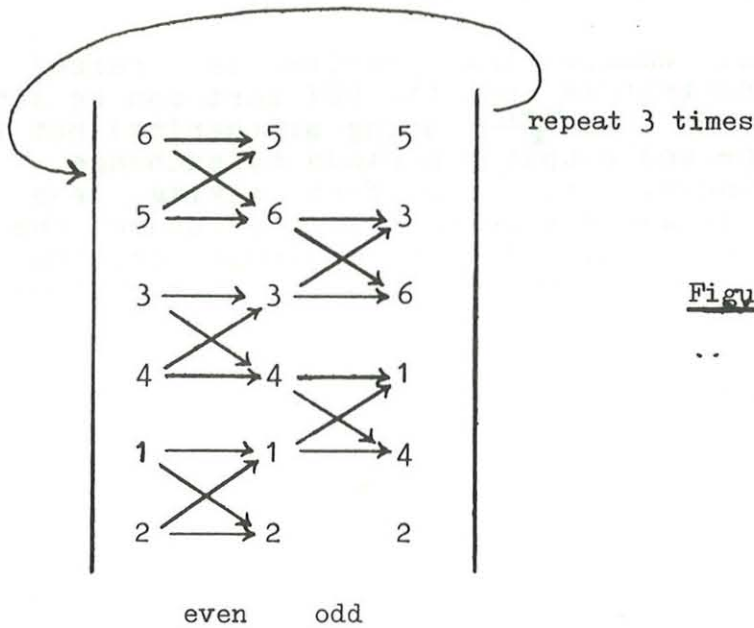


Figure 8



Figure 14 details the sorting of four numbers as they are passed in and out of a CPU-controlled ladder. The numbers enter in the order 3,4,1,2 and emerge in the order 1,2,3,4. Each snapshot shows the contents of the loops each half period (period is the exchange period). The first few snapshots show the numbers slithering into the ladder until "3" occupies the deepest loop completely. At this point the CPU control determines whether to set the switch at the junction of the deepest and next deepest loop to "crossover" or "avoidance". In this example this switch is set to "crossover" and in half a period "4" reaches the deepest part of the deepest loop and "3" appears at the switch between the highest loop and the middle loop. Now the CPU allows the switch into the deepest loop to remain at "crossover" and the other loop-connecting switch is set to "avoidance". Thus at the end of this half period, "4" is trapped in the deepest loop "3" in the middle loop and "1" in the highest loop. In the next half period "1" and "2" begin to exchange and "3" and "4" remain trapped. In the following half period, "1" emerges completely from the ladder and "2", "3" and "4" remain trapped in successively deeper loops. In successive half periods the other numbers follow "1", sorted in the order required. Note that the sorting takes place within the input/output time.

#### Concluding Remarks

This final section is an attempt to draw together some of the conclusions of previous sections and to suggest the general way forward. Comparison of pipelining and synchro-parallelism has shown that both systems are tightly coupled and can complement one another, but that both suffer wasted computing power, the result of lack of flexibility. Large Scale Integration (LSI) of logic elements can help by making tightly coupled systems more economical and can add decoding power to processing elements (for synchro-parallelism) and can permit the installation of tagging to postpone drastic actions in pipelines such as draining. Also LSI can provide general-purpose "personalisable" modules from which special pipeline stages can be made by changing read-only memory.

The future promise should be in a linked network of general-purpose micro-modules in a tree architecture with localised linkages to reduce the total amount of communication cost. The speaker, in 1972, gave the name "Polycentric Computing" to the relatively loosely-coupled system.

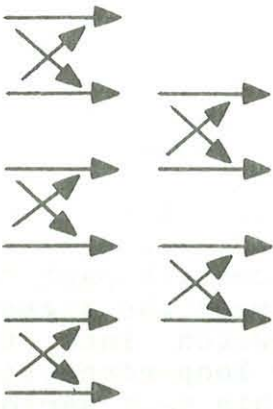


Figure 9



Figure 10



Figure 11

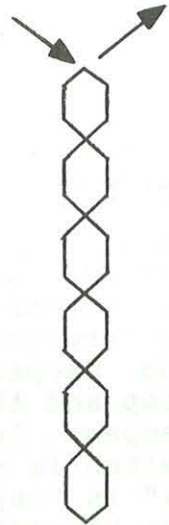


Figure 12

	Pipeline sorter	Ladder sorter
Comparators needed	$N(N-1)/2$	$N-1$
Storage (latches)	$N^2$	$N$
Inputs	$N$	1
Outputs	$N$	1
Repetitions	None	$N-1$
I/O time	None	yes
Duration	$N$ stages	$N$ stages

Table 1

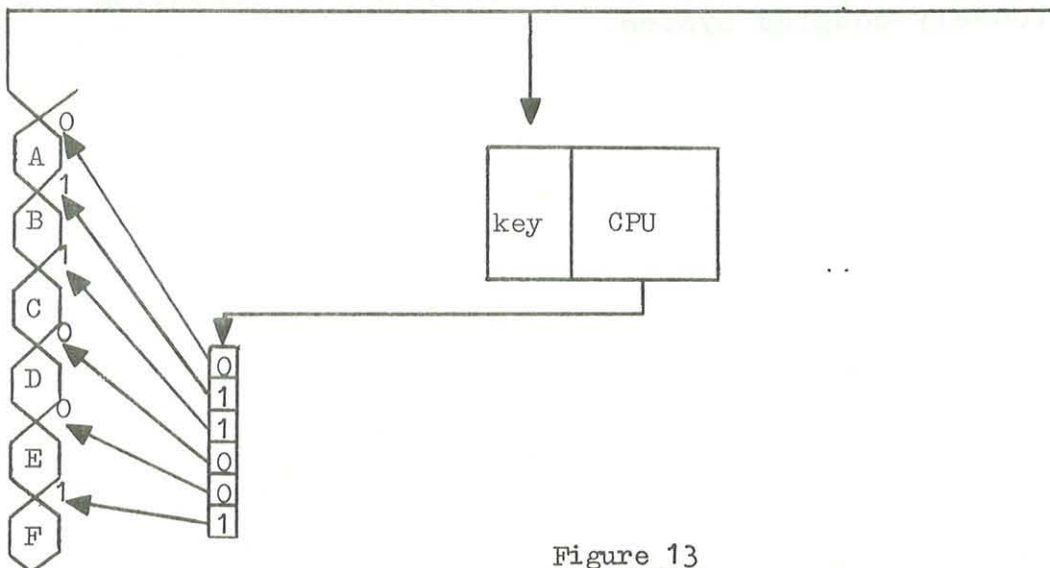
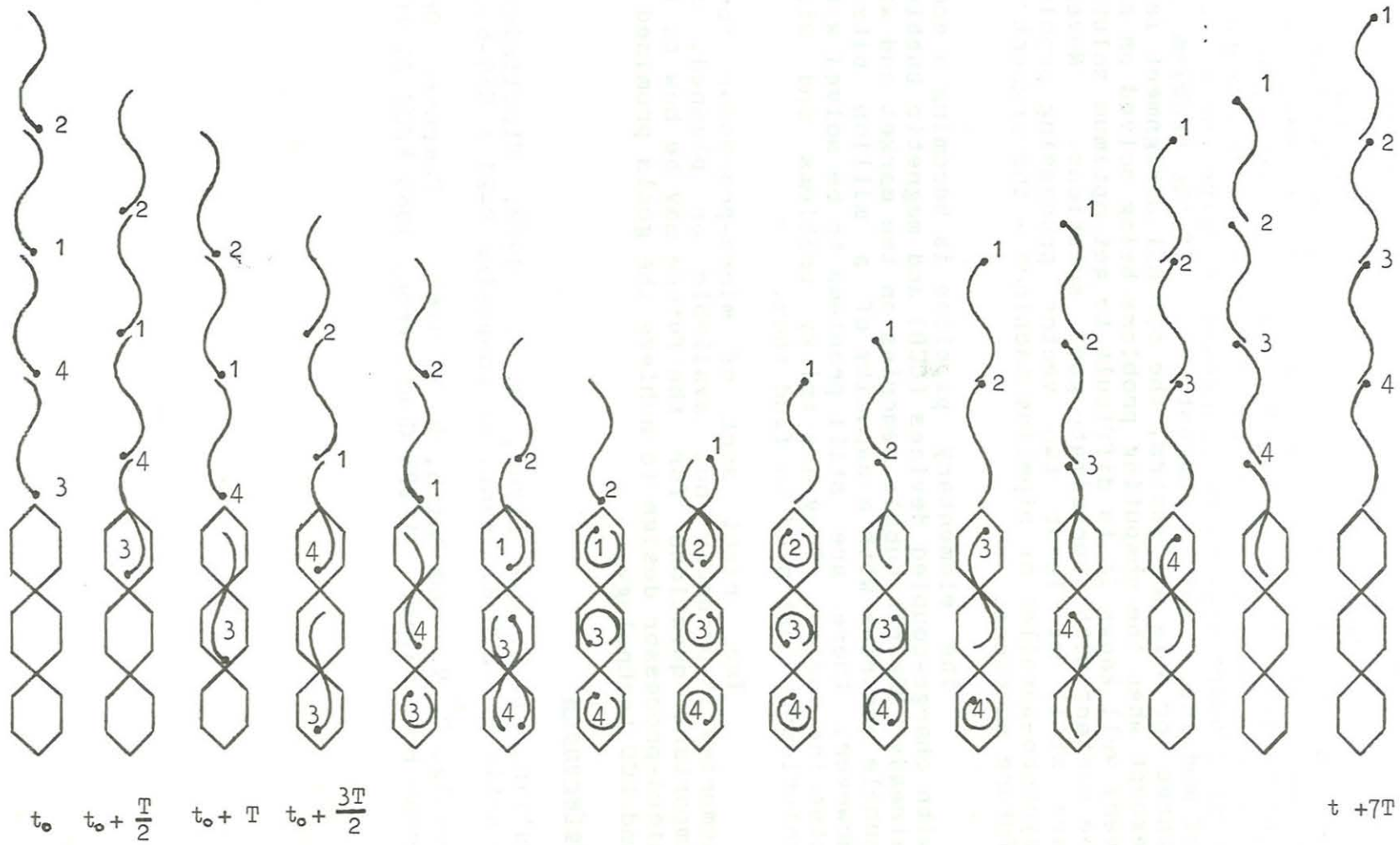


Figure 13

Figure 14



In the context of a system which is loosely coupled, H. Stone describes a problem. In it he has two processors and  $n$  program modules spread between the two. Run time on both processors is known. He proposes that the calling cost between modules in the same processor be zero, while there is a previously known cost when calling across the boundary between processors. The question is how to assign modules to processors. The answer is that the problem is the same as the multiflow graph problem, so that time proportional to  $n$  is needed to solve for an optimal assignment of modules. (Dinic suggest  $n$ ). If the problem is extended to three or more processors, the optimal assignment is unsolved. So, except when the computing problems being solved on such a system are very well known it is difficult to get optimum solutions. One must be content with approximate good solutions. Nevertheless, there are still at least two vector processing problems suitable for synchro-parallel or pipeline machines - the present Illiac IV and future machines.

The elementary pipeline is becoming a commercial reality with charge-coupled devices (CCD) and magnetic bubbles. There are already 92k-bit bubble memories on the market and within two years, bubble memories with a capacity of a million bits are expected. However, there are still problems to be solved with the necessary steering logic. They are tricky problems and will have simple solutions - just need to find them.

The right sort of micro-processor for these bubble memories is probably not available or planned, so one of the important questions for the future may be how to influence future micro-processor design to achieve the goals promised by the bubble and CCD technology.

#### References

- Hallin, T. G. and Flynn, M. J. 1972, Pipelining of Arithmetic Functions I.E.E.E. Trans. on computing C-21 : 880-86.
- Senzig, D. N. and Smith, R. V. 1965. Computer Organisation for Array Processing. AFIPS Conf. Proc. 1965 FJCC 27 (Part 11) 117-28.