LA-UR -85-1194

RECEIVED BY OSTI MAY 0 6 1985

LA-UR--85-1194

DE85 010791

TITLE: MECA: A MULTIPROCESSOR CONCEPT SPECIALIZED TO MONTE CARLO

AUTHOR(S): Johndale C. Solem

SUBMITTED TO: Los Alamos/CEA Monte Carlo Conference, Cadarache, France
April 1985

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

FORM NO 836 R4
ST NO 2629 5/81

MECA: A MULTIPROCESSOR CONCEPT SPECIALIZED TO MONTE CARLO

Johndale C. Solem
Theoretical Division
Los Alamos National Laboratory
Los Alamos, NM 87545

I compare discrete-ordinates and Monte Carlo techniques for solving integro-differential equations and compare their relative adaptability to vector processors. I discuss the utility of multiprocessors for Monte Carlo calculations and describe a simple architecture (the monodirectional edge-coupled array or MECA) that seems ideally suited to Monte Carlo and overcomes many of the packaging problems associated with more general multiprocessors.

Integrodifferential Equations

The last century has witnessed a dramatic expansion in the role of integrodifferential equations in basic and applied physics. The applications of such equations range from the mundane to the sublime: from multidimensional integrals to quantum field theory. Perhaps the most important class of integrodifferential equations are transport equations. In mathematical physics, transport equations occupy a no-man's land: the middle of a triangle bordered by diffusion theory, hydrodynamics, and optics. In fact each of these disciplines is a special case of transport theory. Except for those describing unrealistically simple processes transport equations have no analytic solutions.

Computer science has developed two approaches to the solution of integrodifferential equations: Discrete Ordinates and Monte Carlo. The two techniques in many ways complement each other. Discrete-ordinate techniques give an exact answer to a poorly defined problem, while Monte Carlo techniques give a poorly defined answer to an exact problem.

To use Discrete Ordinates, we must cast in finite difference form all of those parameters over which the integration is to be performed. In the example of transport theory, we must divide space and time into discrete intervals. In addition, cross sections must be defined in discrete intervals as functions of energy, direction, and scattering process. We must average cross sections over groups of energy and scattering angle, and we assume similar incremental averages over time and space. The answer from discrete ordinates techniques, however, is exact to the accuracy of the computing machine. The inaccuracy arises from averaging over the intervals of the discrete ordinates.

When using Monte Carlo, we can define all parameters of the problem to any precision we desire, in principle, to the precision of the computing machine. Cross sections can be defined by tables with variable intervals and with interpolation or polynomial fits to provide as much precision as we desire. Although we can define the problem exactly for a Monte Carlo calculation, the answer is fuzzy. As we use the laws of probability to integrate the equations, it is necessary that the solutions have some statistical fluctuation. The precision of an answer increases as the square root of the computer time used to perform the calculation.

To improve the precision of the answer in discrete ordinates, we improve the precision with which we describe the problem, that is, we refine the multidimensional mesh over which we the integration is being performed. This also requires more computer time. I suggest that when a deeper understanding of these alternative techniques to the solution of integrodifferential equations is acquired, we will find that both discrete ordinates and Monte Carlo offer a precision that increases as the square root of the computer resources expended if each is applied to an arbitrarily general problem.

The most important difference between the two methods is the flexibility offered to the computational physicist, particularly in modifying the two transport techniques to accommodate phenomena that are unique to the problem he is trying to solve.

Using the discrete-ordinates techniques, we usually start by writing down the transport equation that describes the particular physical phenomena we are planning to investigate. We then find a clever way to cast that integrodifferential equation into a set of finite-difference equations. We then write the computer code for calculating these, average raw cross section data into energy and angle groups, and proceed to calculate a solution. The difficulty becomes apparent when we want to change the problem. To make the mesh finer, we must go back and group average the cross sections all over again. But even more difficult is the problem of trying to add some new physical phenomena.

For example, suppose we were trying to calculate the transport of charged particles through a completely ionized plasma. We might first write down the transport equation for Rutherford scattering from the ions and electrons. We would difference that equation, we would set up the group averages from our understanding of the classical and quantum mechanical processes of scattering in a completely ionized gas, and we would proceed to perform the integration numerically. But what happens when we want to change the physics involved? Finding ourselves dissatisfied with only calculating the electromagnetic interaction in the plasma, we might decide to treat the strong interactions with nuclei of the plasma as well. Certain simplifications could be made by realizing that the nuclear interaction and Coulomb interference terms that attend this type of scattering are only important for large scattering angles. And although we could apply certain perturbation techniques, we would likely find ourselves writing down the original transport equation, adding the strong nuclear interaction and Coulomb interference, and differencing the equation all over again. This is a long and tedious task.

The greatest virtue of Monte Carlo is its flexibility. Changes to physics, such as those described above, can be easily accommodated. We need only to include the partial cross section for this new process in the total cross section and, on the basis of the probability of scattering derived from this new term, take a conditional branch to a new section of computer code that represents the strong interaction process. The writing of a conditional branch is a simple process. In fact, one could write an entire Monte Carlo transport code and never have to write down a transport equation. So Monte Carlo is the ideal technique for undertaking heuristic studies of transport processes; we have the flexibility to alter the physics with minimal effort.

Vector Machines

Computational physicists have been studying the application of vector processing machines to transport theory for the last fifteen years. Efforts to shape code architecture in a form that would allow the solution of integrodifferential equations on vector machines started with the Illiac IV[1-3] in the late sixties, proceeded through the Control Data STAR 100[4] in the early seventies, has been applied for the Cray I[5-6] machine, and work is continuing for future machines. Vector processing machines are ideally suited for discrete-ordinates problems. With discrete ordinates, one is frequently operating on many mesh points at the same time, and the same operation is applied to each of them. Monte Carlo, on the other hand, is a much more

difficult problem for vector machines. The crux of the difficulty lies in the construction of conditional branches, and conditional branches are the essence of Monte Carlo calculations. Although we frequently want to disguise them in other forms (conditional branches are not in vogue), Monte Carlo is an intrinsically "iffy" technique. The usual vector techniques involve filling a set of vectors with the phase space coordinates of Monte Carlo particles. Then as conditional branches split these vectors into various subroutines of the codes, the once full vectors are converted to sparse vectors--vectors that contain empty spaces. As the complexity of the problem increases, the vectors become progressively sparser. So while highly efficient techniques can be found for very simple transport processes, those that are more complex become progressively less efficient.

There is another problem with vector machines when applied to Monte Carlo. Upon adding more complexity to the transport equation, we can frequently find other methods for vectorizing the problem to take advantage of the speed of a vector machine. In other words, there are always clever techniques to circumvent the problem of sparsing vectors. But we must go back to the original transport problem, reconsider the whole picture, and then recast the problem back into a vector mode. Does this sound familiar? It is the same problem that we have with discrete-ordinate techniques. Every time we want to make a change to the physics of the problem, we must go back and consider it from the beginning and probably recode it entirely. Thus for vector machine, one might as well be using a discrete ordinates technique.

ENTER THE MULTIPROCESSOR

With the advance of very-large-scale integration, it has become possible to produce an entire computer on a single chip. Today's technology permits mass manufacture of chips with a half million or so transistor-equivalent devices. Numerous technologies including electron-beam and x-ray lithography portend enormous increases in device packing density and wafer-scale integration is near to reality. All of these technologies are relatively young and far from realizing their full potential. The number of devices on a chip has doubled every year or two for the last two decades and it is not unreasonable to forecast 10^8 per chip by the turn of the millenium.[7] Therefore computer architects have considered constructing an array of microprocessors with interconnecting links and exterior links to an array of memories in such a way as to have this microprocessor team "gang tackle" a numerical problem. For Monte Carlo this seems natural. If we are running a scalar algorithm, the precision of our Monte Carlo answer increases the square root of the computational time. Similarly if all the microprocessors are running without interference the precision of the answer should increase as the square root of the number of microprocessors. But they are all sharing memories, and there is the rub. Memory bank conflicts are the principle problem with multiprocessors.

Many researchers regard the multiprocessor as an array of micro-processing units (MPU) and memories linked together by a switching network. The processing units and/or memories are called nodes. The switches may be dedicated to a particular node, in which case the network is called static, or available to several nodes, in which case it is called dynamic. Computer

scientists have suggested an enormous variety of network topologies. Figure 1 is a representative sampling.

A good example of a dynamic network is the S-1 computer,[8] which has N memories and N processors linked by a crossbar switch (Fig. 1a). This architecture is limited because as the cost of memory plus microprocessor increases linearly with N and the cost of the crossbar switch goes as the square of N. Present estimates indicate that the cost of the switch will exceed the cost of the microprocessors and memory for $N \geq 1000$, although some reduction might be realized by replacing the $N \times N$ switch with two $N/2 \times N/2$ switches and two N-input exchange switches.[9] An alternative is to have an $N \times M$ crossbar switch, where the product of N and M remains constant, yet N can be changed by the programmer. This kind of architecture when applied to the Monte Carlo problem suggests that we might have considerable fewer memories than processors, because all of the processors use the same algorithm and the same data base. If each microprocessor were completely free to read from the common memory at its convenience, then only a small local memory would be necessary for each processor plus a common memory in which the results of all of the processors would be talleyed.

Further reduction of the number of switches can be obtained by using 2×2 crossover switches rather than the on-off switches of the crossbar. An example of this is the binary Benes[9] network (Fig. 1b). If it is acceptable to have delays in information transmission owing to blocking of signals competing for the same switch, then the number of processors can be reduced to N log N. Examples of such blocking networks are the baseline[10] and the omega[11] (Fig. 1c).

Static networks present an enormous range of choices. Those that have attracted most attention achieve symmetry at all levels of modularity by using hypergeometric topologies: the node positions become the minimal faces of regular polytopes. In Euclidean space of dimension $n \geq 5$, there are only three regular polytopes: n-simplex (Fig. 1d), n-octahedron, and n-cube (Fig. 1e). Both n-simplex[12] and n-cube[13-15] machines have been studied extensively, but the n-octahedron has been ignored, perhaps because when realized in ordinary space it is nearly the same as the n-simplex. Table I compares the three networks on the basis of usual parameters that characterize speed, cost, and utility. The hypercube enjoys an advantage in number of communication channels and switch positions, it pays for it with increased communication delay time. Unlike the simplex and octahedron, however, the topology of the hypercube is easily adaptable to real space. Hypercubes have been built at Cal. Tech. and elsewhere and the first commercial machine is soon to be offered by Intel. Two-dimensional arrays such as the systolic[16] and cartesian[17] (Fig. 1f) are particularly good at two-dimensional problems such as obtaining relaxation solutions to Poisson's equation. A number of demonstration machines of this sort have also been built.

For ideal scalar Monte Carlo, all of the processors will be reading the same algorithm and data base. They will also be talleying in the same memory arrays. For a very large number of processors, one copy of the algorithm will not be sufficient. If dynamic blocking networks are used, there will be delays getting to the memory and bank conflicts once inside memory. It is clear that several copies of the algorithm and data base will be needed as well as several copies of the talley memory. The fewer copies the more networking is required and the more bank conflict will occur. What is the

ideal ratio of memories processors? The thesis of this paper is that, given present technological trends, the ideal ratio is one.

THE TREK TO MECA

The Monodirectional Edge-Coupled Array (MECA) is the simplest possible architecture for a multiprocessor computer. It is a linear network of microprocessor units (MPUs) random-access memories (RAMs) and read-only memories (ROMs) as illustrated in Figure 2. Figure 2 shows a hypothetical tie-line connecting all of the microprocessor units. When this tie-line is pulled high, all of the microprocessors read and write the RAMs on their right as shown in Fig. 2a. When the line is pulled low, all of the microprocessors read and write to the RAMs on their left as shown in Fig. 1b. All of the microprocessors read and write RAMs only one at a time, and all of the microprocessors switch directions simultaneously.

This architecture eliminates two problems of Monte Carlo multiprocessors: (1) The price of the network does not increase with the number of MPUs but in fact disappears entirely; and (2) The time required to read an algorithm and data base into each of the RAMs is only slightly longer than the time to read the algorithm into a single memory.

To demonstrate this fact let us look at a specific case. Consider a MECA system consisting of N microprocessors and a Monte Carlo algorithm consisting of W words. The process of loading the memories proceeds as follows: The control processor, which is on one end of MECA loads the first word of the algorithm into the first RAM while holding the tie-line high. The control processor then pulls the tie-line low so the first RAM is read by the first MPU. The first MPU reads the first word out of RAM into one of its accumulators. The tie-line is then pulled high and the first MPU stores the first word into the second RAM while the control processor is reading the second word into the first RAM the tie-line is pulled low and the process is repeated. The instructions for this daisy-chain operation are contained in the ROM that accompanies each MPU.

If all of the RAMs were being loaded with the algorithm serially the time required would be given by

$$T_{serial} = NW\Delta t \quad , \tag{1}$$

where Δt is the time required for one word to be loaded. If, on the other hand, the algorithm could be fed to all the RAMs simultaneously the time required would be

$$T_{parallel} = T_{serial}/N = W\Delta t \quad . \tag{2}$$

With the daisy-chain operation described here, the total time to load the algorithm and all the RAMs is given by

$$T_{load} = (N + W)\Delta t \quad . \tag{3}$$

If we consider the loading of all the RAMs simultaneously to be the maximum possible efficiency then the efficiency of this daisy-chain technique is given by

$$E = T_{parallel}/T_{load} = W/(W + N) \quad . \tag{4}$$

For any realistic system this efficiency is very high compared to the serial efficiency. If, the algorithm were 64K words long, and there were 64K microprocessors, the efficiency would only be half the optimum or parallel efficiency. This is excellent when compared to other techniques.

At the end of any Monte Carlo operation, for example, calculating the heating in a reactor, it would be necessary to sum up the results from all the microprocessors into an array of numbers representing the heating in the various geometric zones under consideration. Such a process is completely analogous to the loading of the algorithm described above. Thus the efficiency is still given by Eq. (4).

PACKAGING

The most conspicuous advantage of the MECA computer is its adaptability to highly efficient packaging. In state-of-art LSI-based supercomputers, 30% of the time involved in a computational cycle is owing to delays in information transmission among the individual microcircuits. Factors of ten speed-up in the microcircuits will accrue little improvement unless the packaging is accompanied by a similar improvement in efficiency. Furthermore, as chips become more complex, the difficulty of making connections from the microcircuit to pins becomes intolerable. Present estimates[17] suggest that chip fanout will be limited to 256 pins.

Because it is monodirectional and edge-coupled, the MECA computer lends itself to a particular compact packaging technique. The chips or wafers can be stacked, eliminating the fanout problem entirely. The natural arrangement is to have alternating wafers of MPU + ROM and RAM. It may be necessary to have several RAM wafers if large memory is required. Because of the cleavage planes of silicon and silicon nitride it is natural, although not mandatory, to cut wafers into rectangles. For the MECA machine they must actually be squares. The squares of the edge-coupled array would then be stacked vertically with a 90° rotation at each chip. Figure 2a illustrates the physical connection of the edge-coupled array. It is a single surface folded together in a stack. Figure 2b is an exploded view of the stack with lines representing the electrical connections at the edges of the chips. Thermomigration techniques might be used to produce feedthroughs and microspring interconnects between the surfaces of adjacent wafers.[18] Because of the up-reading and down-reading components of each chip are spatially separated, the conductors that pass information from one chip to the other can conduct completely through the chip, greatly relaxing engineering requirements for their fabrication. In fact, this geometric separation allows the possibility of using capacitive coupling from one chip to another. Thus perfect contact between chips is not necessary in the MECA machine; although capacitive coupling may cause delay. The packaging of the MECA computer therefore approaches optimum efficiency. The efficiency is near optimum if alternating chips have to be MPU plus ROM and RAM and is identically optimum if each chip contains an MPU, ROM and RAM.

Cooling of the chip package is easily accomplished by putting a heat exchanging fluid around the package and through optimally placed holes that will line up when the chips are stacked. This can be done with minimal perturbation of the circuit layout. For fifth generation components, cryogenic packaging is likely, and the heat exchanging fluid may be liquid nitrogen. The cooling problem may be considerably mitigated when gallium-arsinide technology is further developed, particularly if transistors can be made small enough to operate in the ballistic regime.

CONCLUSIONS

I have described an extremely simple multiprocessor connection system that is ideally suited to Monte Carlo problems. The time lost in loading the algorithm and in gathering and summing information from all the multiprocessors is only a factor of two or so greater than optimum. The delays owing to pa configuration are minimized. The whole system is as well adapted to Monte Carlo as a single-scaler processor.

REFERENCES

1 D. Slotnick, AFIPS Conf. Proc. 30 477-81 (1967).

2 G. Feierbach and D. Stevenson, 77-92, Infotech State of the Art Report: Super Computers, Vol. 2, C. Jesshope and R. Hookey editors, Infotech Intl. Ltd. (1979).

3 D. Slotnick, Sci. Am. 224 (2), 76-87 (1971).

4 R. Hintz and D. Tate, COMPCON '72 Digest, 1-4 (1972).

5 R. Russel, Common. Assoc. Comput. Mech; 21, 63-72 (1978).

6 M. Dungworth, 51-76, Infotech State of the Art Report: Super Computers, Vol. 2, C. Jesshope and R. Hookey editors, Infotech Intl. Ltd. (1979).

7 L. Uhr, University of Wisconsin Technical Report #518 (1983).

8 L. Widdoes and S. Correll, Tutorial on Parallel Processing, 136-145, R. Kuhn and D. Padna, editors, IEEE Los Angeles (1981).

9 V. Benes, Mathematicsl Theory of Connecting Networks and Telephone Traffic, Academic Press, New York (1965).

10 C. Wu and T. Feng, Proc. Compcon Fall 1980, 599-605 (1981).

11 D. Lawrie, IEEE Trans. Computers, C-24, 1145-1155 (1975).

12 E. Aupperle in Computer Networks, 49-63, R. Rustin Editor, Prentice-Hall, Englewood Cliffs, NJ (1972).

13 H. Sullivan, T. Bashkow, and K. Klappholz, Proc. Fouth Symp. Computer Architecture, 105-125 (1977).

14 M. Pease, IEEE Trans. Comput, C-26, 458-473 (1977).

15 E. Bustos, J. Lavers, and K. Smith, COMPCON '79 Digest, 380-389 (1980).

16 H. Kung in Advances in Computers, 19, M. Yovitis editor, Academic Press, NY (1980).

17 G. Barnes, IEEE Tans, C-17, 746-757 (1968).

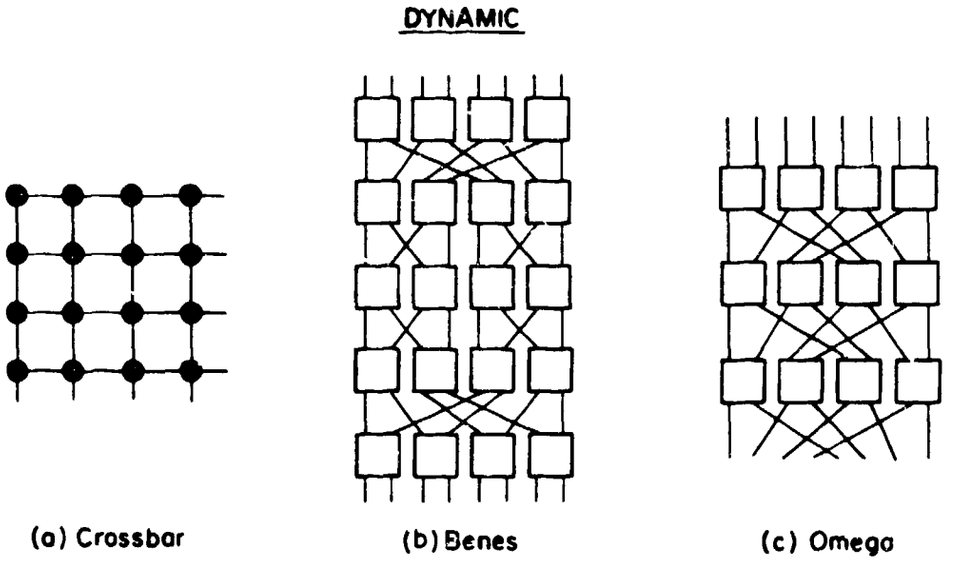
18 R. Etchells, J. Grinberg, and G. Nudd, Proc. Soc. Photo-Optical Instrumentation Engineers (1981).

TABLE I

Comparison of Hypergeometric Networks with N Processing Units

	simplex	octahedron	cube
Channels	$\frac{1}{2}N(N-1)$	$\frac{1}{2}N(N-2)$	$\frac{1}{2}N \log_2 N$
Switch Positions	$N-1$	$N-2$	$\log_2 N$
Average Delay	$\frac{N-1}{N}$	1	$\frac{1}{2}\log_2 N$

Figure 1. The crossbar switch (a) permits any node on the right to communicate with any node on the bottom without blocking. The Benes network (b) does the same, but requires 2 × 2 crossover switches rather than on-off switches. The omega network (c) reduces the number of switches to N log N but introduces blocking. The hypergeometric simplex (d) connects every node to every other node, in this case n = 7. The hypercube (e) requires fewer connections, here n = 4. The simplest 2d network is a cartesian grid (f).



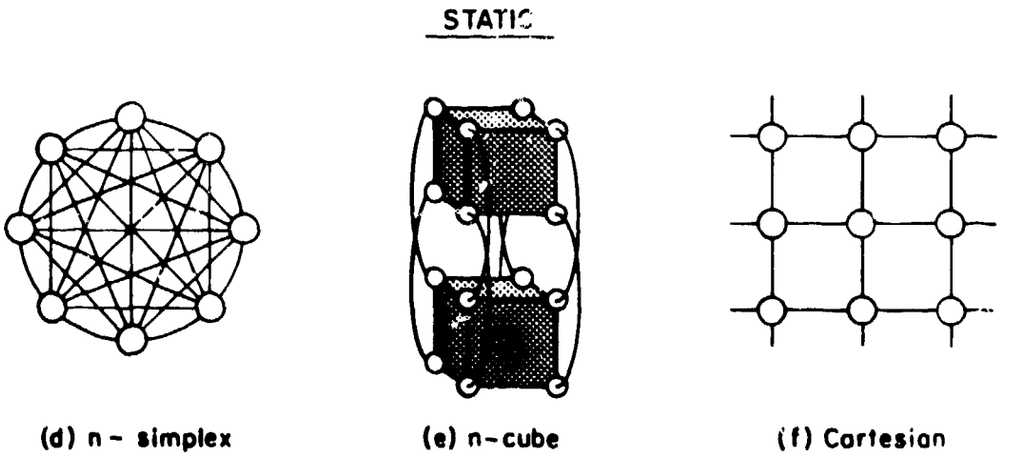
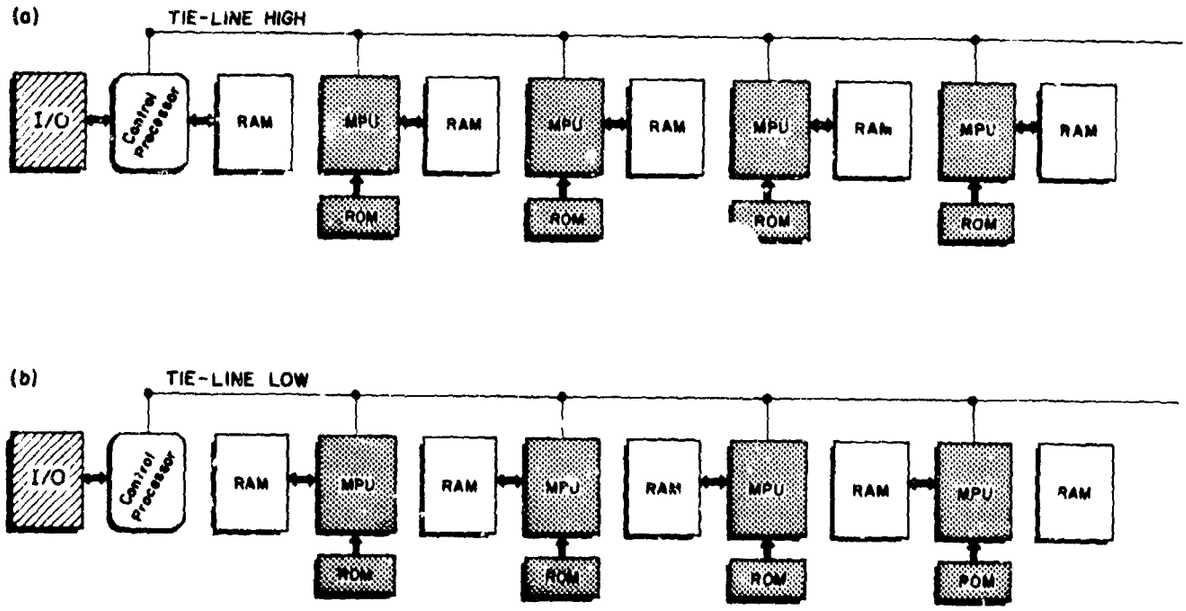
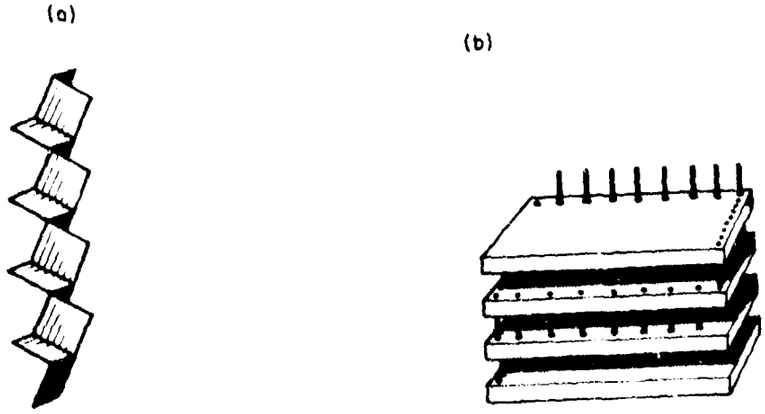


Figure 2. Schematic diagram of an edge-coupled array: (a) control processor pulls tie-line high and MPUs access the RAM to their right; (b) control processor pulls tie-line low and MPUs access the RAM to their left.



MONODIRECTIONAL EDGE-COUPLED ARRAY

Figure 3. Chip stacking: (a) edge coupled array is in fact a single surface; (b) exploded stacking with interconnections.



CHIP STACKING