COMPARING SERIAL COMPUTERS, ARRAYS AND NETWORKS, USING MEASURES OF "ACTIVE RESOURCES"

by

Leonard Uhr

Computer Sciences Technical Report #446

August 1981

COMPARING SERIAL COMPUTERS, ARRAYS AND NETWORKS, USING MEASURES OF "ACTIVE RESOURCES"*

Leonard Uhr

Computer Sciences Department
University of Wisconsin

Abstract

This paper explores possible measures of the efficiency and power of a computer that will allow us to make meaningful comparisons not only between different traditional serial computers, but also between serial computers and parallel array and network computers. It attempts to apply "active resources" (e.g., gates, chip area) counts to compare protototypical examples of each of these general types. Active resources measures suggest that larger numbers of simpler computers, each with a small local memory, give the greatest throughput. The large SIMD arrays appear to be today's best examples of such systems.

Introduction

New measures are needed to evaluate and compare traditional serial computers with parallel network and array computers. A number of researchers have built or are building large SIMD arrays, including CLIP4 (Duff, 1978), DAP (Reddaway, 1978), and the MPP (Batcher, 1980). Other researchers are designing smaller MIMD networks of more powerful computers, including MICRONET (Wittie, 1978), PM4 (Briggs et al., 1979) and PASM (Siegel et al., 1979). These arrays and networks appear to offer enormous potential increases in both power and speed, especially for image processing and other perceptual tasks where large pictures (e.g., 512 by 512 arrays of picture elements) must be analyzed (see Uhr, 1982). But it is difficult to evaluate their efficiency, or to compare them with traditional serial computers.

Today "percent CPU utilization," although people are aware this does not tell the whole story, is virtually the only measure used to evaluate how efficiently a computer operates. This percent is arrived at by accumulating the total time the CPU is busy executing program instructions. A system with less than 30% or 40% CPU utilization is considered poor; one with more than 60% or 70% CPU utilization is considered very good. (Sometimes the percent used for operating system overhead is included in this measure of productively used time, sometimes it is not.)

This measure reflects the fact that a computer's job is to execute users' programs (compute); the CPU does that computing; therefore the computer is busy if and only if the CPU is busy (computing on user programs).

When one views the problem of using a computer efficiently from this perspective, one concludes that the whole system should be designed to keep the CPU busy. This results in the addition of as much memory as is needed to ensure that the information (program statements, data) the CPU needs to continue working will always be immediately available. Now that memory has become very cheap, this rule of thumb appears to be even more reasonable. Since the processor is also idled when a program inputs or outputs data, multi-programming systems are used so the processor can always find a program to work on. This necessitates still more memory. Additional input-output processors, that can work in parallel with a CPU that is multi-programming, so that when one program must wait for I-O another can immediately be re-initiated, and fast cache memories and registers, are all indicated for this same basic reason - to keep the CPU busy.

Evaluating and Optimizing Traditional Serial Computers

The major way a traditional serial computer is improved in power and in speed is by beefing up the CPU, with faster hardware, and also with more hardware, in-

*This work was supported by NSF grant MCS-8109103 for research on multi-computer architecture.

cluding a number of special-purpose processors optimized to execute specific instructions, or that cast frequently used sequences of instructions into optimized hardware. Thus 32-bit parallel multiply, 8-bit character match, floating point accelerators, pipelines for vector arithmetic, and a variety of other pieces of hardware are put on the CPU's bus, with the controller assigning the appropriate processor to each instruction the CPU executes. Both this special-purpose hardware and increased speed from brute technology make the CPU gobble up data ever faster, and thus exacerbate the need to add memories, I-O processors and a multiprogramming operating system.

Parallel Array and Network Multi-Computers

Both arrays [e.g., ILLIAC IV (Barnes et al., 1968), CLIP4 (Duff, 1978), DAP (Reddaway, 1978), MPP (Batcher, 1980)] and networks [e.g., Cm* (Swan et al., 1977), PASM (Siegel et al., 1979)] have many processors. The obvious way to apply measures of CPU utilization to them is to average the percent CPU utilization for all processors. But there are a number of serious flaws in such a procedure, as we shall see.

Networks look very bad when such a measure is used. Each node of a multicomputer network is a traditional computer, with all the problems of utilizing the CPU. But in addition there are many new problems of contention and of communications, including the loading and transferring of programs, the sending and receiving of intermediate results, the requesting and sending of information not stored locally, and a variety of other "message-passing" tasks.

Arrays (here I am considering the SIMD tightly coupled multi-computers like ILLIAC IV, CLIP4, DAP, MPP), because every processor is executing the same instruction at any time, suffer when (as frequently happens) some or many of the processors must be "disabled" to do nothing, because the present instruction is not relevant to all parts of the problem.

Measures That Consider Utilization of All the System's Resources

Any computer, including the multicomputer networks and arrays as well as the traditional single-CPU serial computer, is built from processors linked via wires and switches to memories and to input-output devices. [This paper will ignore the major problem of supplying adequate input and output channels, except for the following comments: To the extent we succeed in increasing the throughput of any computer we thereby increase its processing bandwidth, and we must therefore increase its input and output bandwidths commensurately. If input-output delays are excessive, rather than rely entirely on multi-programming and on larger memory and input-output bandwidths, we can develop languages that expedite input-output in parallel with processing, and languages and operating systems that anticipate needed data.] Thus a computer is built from processors, memories, wires and switches. But all of these components are built from (wires connecting) logic gates, which in turn are built from transistors.

The Computer's Underlying Gates, Transistors and Chip Area

We therefore should seriously consider evaluating a computer's percent utilization, effective power, or efficiency in terms of the total hardware in that computer, using the number of gates, or the number of transistors, or, for a particular chip technology, the number of transistor-equivalent devices.

As we move from LSI chips, with a few thousand devices on each chip, to VLSI, VVLSI, ... chips, with hundreds of thousands or millions of devices on each chip, two additional measures become increasingly attractive:

Wires might be counted along with devices they connect. Wires (which are merely very thin etched lines on the chip) become an appreciable part of the system. It is important to have a modularized, highly packed design, one that puts devices that must be linked by wires as close together as possible.

Chip area used by a module of the total system, whether a specialized processor, or the total processor, or a processor plus its memories, will become an increasingly more appropriate measure. Chip area is a function of the interactions between the particular chip fabrication technology used, the design rules for building gates and larger modules using gates, and the ability of the designers and their computer-aided design tools to minimize chip area used (see Mead and Conway, 1980).

This then gives several alternative measuring units:

a) number of gates,
b) transistors,
c) transistor-equivalent devices,
d) devices-plus-wires,
e) chip area.

These are all similar in that they are far more micro-modular than the traditional alternative, CPU utilization. They may appear to be at too low a level. But they are of great potential interest, and I think validity, because they allow us to compare radically different computers, consider the separate parts of a large CPU separately, and consider processors and memory together.

Gate Counts in SIMD Arrays, MIMD Networks, Serial Computers

To give some examples (see Kuck, 1978, Uhr, 1982):

A traditional serial computer's CPU has typically used around 10,000 gates (a super-computer like the CRAY-1 uses roughly 100,000 gates).

A traditional high-speed memory typically has from 64,000 to 64,000,000 bits of memory, each bit stored by from 1 to 12 transistors.

Typical SIMD array computers use much simpler 1-bit processors, with only 100 to 600 gates per processor, and very small memories for each processor, of from 32 to 4,000 bits. Thus, e.g., a 100 by 100 array of 10,000 processors might have 5,000,000 to 50,000,000 gates for processors, and 320,000 to 40,000,000 bits of memory. (CLIP4 has 9,216 processors, totalling about 3,000,000 gates, and 320,000 bits of memory. DAP has 4,096 processors, with about 400,000 gates, and 16,000,000 bits of memory.)

Table 1. Gate Counts of Resources in Different Types of Computer. [Assume 1 bit/gate]

Computer	Proc(s)	Memory(s)
Serial:		
small (8bit)	3,000	64,000
medium	10,000	6,400,000
large	30,000	64,000,000
super	100,000	256,000,000
1-bit SIMD Arrays:		
small	50	32
medium	100	256
large	300	1,000
super	600	4,000
Examples:		
CLIP4	300	32
DAP	100	4,000
MPP (planned)	600	1,000

The contrasts are striking: a traditional serial computer has from 98% to 99.99% or more of its gates in memory; the SIMD arrays have 98% (DAP), 63% (MPP), and 9% (CLIP4) of their gates in memory.

Table 2. Total Gates in Arrays.

Computers		Proc-Gates	Mem-Bits
CLIP	10,000	3,000,000	320,000
DAP	4,000	400,000	16,000,000
MPP	16,000	9,600,000	16,000,000

Estimates of Resources "Active and Working"

When a traditional computer's CPU is being "fully utilized" it will be executing one of its instructions, using several registers and memory locations from which it fetches that instruction's operands and into which it stores the results. It will also be using registers and memory locations for the actual instructions, plus controller hardware to decode instructions, assign particular registers and processors, and handle a variety of other control functions. Many of the instructions will operate on 32-bit data. But some instructions will operate on 8-bit characters, some on small integers (e.g., 6, 11 or 19 bits), and some on 1-bit logic expressions.

When an SIMD system executes an instruction one controller will be similarly active. But now many processors will be active, and each will use essentially the same amount of register and high speed memory as that used by the single serial processor.

The hardware needed for the 32-bit processors will have many more gates than that used by the 1-bit processors in today's 1-bit machines (but it will be roughly comparable to the 10,000-gate ILLIAC IV processor). Because the 1-bit processor does not have special-purpose processors for specific operations, and because it is so simple, a relatively large percent of its gates will be used for each operation. In sharp contrast, a far smaller percent of the gates in a traditional 32-bit computer's CPU will be used for each particular instruction. Very roughly, more than 40%, and probably more than 70% of the 1-bit processor's gates will be active, whereas less than 20%, and probably far less than 10% of the 32-bit processor's gates will be used.

The situation for the controller is quite different. In both cases the controller and its gate utilization can be assumed to be the same. But the array uses (and hence amortizes the cost of) its one controller for thousands of processors, whereas a network of traditional computers uses a separate controller for each processor. Since controllers can vary tremendously in cost comparisons are difficult to make. But there is no apparent reason why controllers for SIMD systems should be inherently more expensive than controllers in MIMD networks. On the contrary, typically the SIMD restrictions make them far simpler (although the MPP controller is unusually complex).

Roughly, a processor needs 2 to 8 registers plus 2 to 8 memory locations to handle the data involved in the execution of an instruction. The MIMD processor needs 2 additional registers to handle the instruction. This gives striking differences: A 1-bit SIMD network with 10,000 processors needs 40,000 to 160,000 register and memory bits to handle the data flow generated by the currently active instruction. In sharp contrast, each traditional 32-bit processor (whether in a serial computer or in an MIMD network) needs 256 to 768 register and memory bits.

A frequent criticism of the SIMD array is that often many of its processors must be disabled, or, more subtly, may appear to be doing something but in fact are grinding out results that will never be used. The latter situation is almost impossible to detect. But rough estimates of between 30% and 80% active processors averaged over a wide range of programs and instructions seem plausible. This figure can also be controlled quite simply, by combining the array with a serial processor or network that handles instructions where too few of the array's processors would be active.

<u>Table 3</u>. <u>Active Resources in Different Types of Computers</u>. [Assume 1 bit/gate. Estimates, some relatively rough, are based upon the examination in the text.]

Computer:	Processor Gates:			Memory Gates:		
	Total	Active	%	Total	Active	%
Serial	10,000	<1000	<10%	64,000	18	0.004%
SIMD Array	300	200	66%	1,000	16	2%
CLIP4	300	150	50%	36	16	40%
DAP	100	75	75%	4,000	16	0.3%
MPP	600	400	66%	2,000	16	0.6%

<u>Table 4</u>. <u>Cost of Controllers Amortized Over Different Computers</u>.

Computer	Processors	Controller/Processor%
Serial	1	100%
MIMD	N	100%
SIMD	N	100/N%
CLIP4	10,000	0.001%
DAP	4,000	0.0025%
MPP	16,000	0.0006%

"Active Resource" Comparisons of Serial, SIMD & MIMD Computers

SIMD systems with 1-bit processors have a far larger percent of their gates allocated to processors, use a higher percent of processor gates for each instruction, and use a far higher percent of memory bits for each instruction.

MIMD networks, when each node is a traditional computer with its traditional large memory, start exactly like the traditional computer. But then they must pay a number of new overhead costs, to handle and store messages, synchronize processors, store multiple copies of data and of programs, resolve contention, and so on. And they must expect processors to be idled a good percent of the time simply because they are waiting for data that other processors have not yet finished generating. Careful designs, appropriate technologies, efficient operating systems, and programs that map efficiently onto the network can minimize, but not eliminate, the resulting degradations in performance.

This presents an extremely discouraging picture for MIMD networks, unless they use different kinds of computers from the traditional. Here, however, there are at least three important possibilities:

1) The local memory assigned to each processor can be made much smaller, since that processor will have to work with only a portion of the total set of information needed to execute the total program. It seems likely that a system's memory size is preponderantly a function of the programs being executed and their need to store data and intermediate results, rather than the total number of processors in the system.

Basically, there are three ingredients: A program must be executed by processors, which need memory. It seem plausible that, if anything, the more processing power the less memory needed, since there is less need to store intermediate results for long periods of time. (This assumes that the system's input and output bandwidths are sufficient, so that memory is not needed as a buffer.) But a good mapping of program onto network (see Bokhari, 1981) becomes more critical as memory is decreased, since it becomes more likely processors will not find the data they need, and the message-passing needed to get that data will waste the time of other processors as well.

2) Each processor might be made simpler, especially by eliminating most of the expensive, rarely used special-purpose processors. This would greatly increase the percent ultilization of processor gates. This may be justified by the fact that the problem is being decomposed, and spread over the many processors in the network. Each processor's computing burden is therefore reduced.

3) Processors might be made simpler by breaking the typical CPU apart, into a network of different types of special-purpose processors. But an increased burden may be put on the programmer, and/or the operating system, to map processes onto the appropriate processors, while keeping the overall mapping efficient. Possibly here a reconfiguring capability would be useful (see, e.g., Lipovski, 1977).

Discussion, Possibilities, Problems

The processors are the only gates that are actually working, and the memory registers they use are the only concurrently active memories. It therefore seems highly desirable to increase the percent of such active processors as much as possible, while keeping each processor and each processor's memory as small as possible.

But this is not nearly so simple as it appears, and there is a major danger. For gates can be made active (just as computers can be made active), although, or even because, they are doing very little, in a needlessly cumbersome way. When this is the fault of the programmer we can say, "compare systems on efficiently coded programs." But to achieve efficiency the programmer may well have to take the architecture into account; yet it is very difficult to achieve the most appropriate, most efficient algorithms for radically different new architectures. And the architecture itself can force needless inefficiencies, as when avoidable message-passing burdens the system.

When the architecture is appropriate, the SIMD array of 1-bit processors gains appreciably in several ways:

It has a much higher per cent of gates in processors as opposed to memory.

It uses a higher percent of its much simpler processor gates to execute each instruction.

It amortizes the cost of the single controller over the relatively large number of simple processors.

It uses a much higher percent of its memory registers for each instruction executed.

Potentially, this can give striking increases in computing power. However these several aspects can be unbundled, when desirable (and especially if this architecture is too restrictive), giving a variety of possible systems that might be explored. Most of today's SIMD systems are 2-dimensional arrays, using 1-bit processors. But they need not be, since any interconnection topology, with any type of processor, can be used with a single controller. Nor is a single-controller SIMD system necessary. A small set of controllers might be amortized among a relatively larger set of processors, as in a pyramid (Uhr, 1981, 1982) or a partially reconfigurable network like PASM (Siegel, 1979). Minimizing the amount of memory, and the amount of unnecessary processing (as in needless message-passing) both seem desirable. These depend upon appropriate architectures, where data need not be shipped around more than necessary, or stored for longer times than necessary.

MIMD networks of traditional computers have all the traditional inefficiencies, plus major new ones of their own, because of new overheads from message-passing and synchronization. Today these overheads are extremely heavy (e.g., it takes thousands of times longer to pass data between processors than to process those data). Unless these overheads can be drastically reduced, or the individual computers are redesigned to have a higher percent of active resources, it may be preferable to speed up a single computer as much as possible rather than to network several computers together.

Possibly the crucial feature of SIMD systems lies in the algorithm-structured architecture, which allows for efficient processing (when program and data are mapped properly into the architecture). To the extent that this is true, it may be possible to make MIMD networks and mixed SIMD-MIMD systems similarly efficient.

At least two other architectures should be examined, and more detailed, deeper comparisons devised to compare these radically different serial, parallel and parallel-serial systems. The two architectures are pipelines (e.g., the Cytocomputer, Sternberg, 1978) and systolic arrays (Kung, 1980).

A pipeline tends to use processors specialized for a particular type of problem (e.g., a window operation for image processing, floating point arithmetic for numerical matrices). Hence the processor can be relatively simple, with a relatively high percent of active gates. Memory traditionally is even smaller than in the arrays, since intermediate results are immediately pumped into the next processor in the pipe. But each processor must have its own controller, albeit a relatively simple one.

Systolic arrays are, basically, 2-dimensional pipelines that have very simple special-purpose processors, configured into a system that will execute a particular algorithm. Therefore the "processor" can be quite small, e.g., 25, 10 or even 5 gates, and the memory tailored to the absolute minimum needed for that algorithm. The program instruction and controller functions are taken over by hard-wiring.

Both pipelines and systolic arrays are more specialized, or even special-purpose. They handle a much smaller set of programs, but they may be more efficient for those they can handle. A set of special-purpose systolic arrays reminds one of a CPU's armory of special-purpose processors on a common bus. But now data might be pumped through sequences of these processors, appropriately configured so that many, rather than one, will be busy at each cycle.

Here we see specific examples of the general phenomenon of the price that must be paid for generality.

Systolic arrays are built exactly to achieve isomorphic mappings of hardware to algorithm.

Serial computers pay a heavy price in random memory access for a single processor, which allows isomorphic mapping via software. Serial computers have a very low percent of active resources because of the serial "Von Neumann bottleneck" (Backus, 1978).

Arrays and networks can be given a more or less general topology, and sometimes good algorithms map well, or even perfectly, but sometimes not.

Networks of traditional processors have all the inefficiencies of serial computers, plus many more of their own, because of the often excessively high overheads from message-passing and poor synchronization.

Networks might be made more efficient by specializing both processors and topology to (sets of) problems, in the spirit of systolic arrays.

Limited reconfiguring might help effect this (at the non-negligible cost, which must be taken into account, of reconfiguring switches).

Summary and Conclusions

Several radically different types of multi-computer array and network systems can now be built. Each has its advantages and its disadvantages.

The very large SIMD arrays of simple 1-bit processors, each with a small local memory, appear to use the highest percent of active resources and therefore, potentially, to be able to give the greatest, and fastest, throughput.

But arrays tend to be specialized, at least today. Judicious combinations of arrays (as in pyramids or other stacks of arrays) and of arrays with networks, and also limited reconfiguring, might serve well to increase generality while maintaining high utilization.

References

Backus, J., Can programming be liberated from the Von Neumann style: a functional style and its algebra of programs, Communic. ACM, 1978, 21, 613-641.

Barnes, G.H., Brown, R.M., Kato, M., Kuck, D. J., Slotnick, D.L., and Stokes, R.A., The ILLIAC IV Computer, IEEE Trans. Computers, 1968, 17, 746-757.

Batcher, K.E., Architecture of a massively parallel processor, Proc. 7th Annual Symp. on Computer Arch., ACM, 1980, 168-174.

Briggs, F., Fu, K. S., Hwang, K. and Patel, J., PM4 - a reconfigurable multimicroprocessor system for pattern recognition and image processing, Proc. AFIPS NCC, 1979, 255-265.

Duff, M. J. B., Review of the CLIP image processing system, Proc. National Computer Conf., 1978, 1055-1060.

Kuck, D.J., The Structure of Computers and Computation: Vol. 1, New York: Wiley, 1978.

Kung, H.T., The structure of parallel algorithms, In: Advances in Computers, Vol. 19, M.C. Yovits (Ed.), 1980, 293-326.(a)

Lipovski, J., On a varistructured array of microprocessors, IEEE Trans. Computers, 1977, 26, 125-138.

Mead, C.A. and Conway, L.A., Introduction to VLSI Systems, Reading, Mass: Addison-Wesley, 1980.

Reddaway, S.F., DAP - a flexible number cruncher, Proc. 1978 LASL Workshop on Vector and Parallel Processors, Los Alamos, 1978, 233-234.

Siegel, H.J., et al., PASM: A Partitionable Multimicrocomputer SIMD/MIMD System for Image Processing and Pattern Recognition, School of Electrical Engineering TR-EE 79-40, Purdue Univ., West Lafayette, 1979.

Sternberg, S.R., Cytocomputer real-time pattern recognition, paper presented at Eighth Pattern Recognition Symp., National Bureau of Standards, April, 1978.

Swan, R.J., S.H. Fuller and D.P. Siewiorek, Cm* - A modular, multi-microprocessor, Proc. AFIPS NCC, 1977, 637-663.

Uhr, L., Network and array architectures for real-time perception, Computer Sciences Dept. Tech. Rept. 424, Univ. of Wisconsin, 1981.

Uhr, L., Computer Arrays and Networks: Algorithm-Structured Parallel Architectures, New York: Academic Press, 1982.

Wittie, L.D., MICRONET: A reconfigurable microcomputer network for distributed systems research, Simulation, 1978, 31, 145-153.