Process-Structured Architectures to Transform Information Flowing Through[†]

Leonard Uhr

Computer Sciences Technical Report #764

April 1988

[†]Invited paper for Parcell '88 - IVth Internation Workshop on Parallel Processing by Cellular Automata and Arrays, Berlin, 1988. (To be published by Akademie-Verlag, Berlin.)

PROCESS-STRUCTURED ARCHITECTURES TO TRANSFORM INFORMATION FLOWING THROUGH

Leonard Uhr[1]

Abstract

It is rapidly becoming possible to build networks of many millions of processors. These massively parallel networks should be capable of perceiving, remembering, and reasoning in real time, even approaching the brain's speed and power - IF we knew how to build them. This paper proposes and explores a promising but largely ignored information-flow approach.

For maximum speed and efficiency, massively parallel networks should be structured so that processes can, brain-like, transform information flowing through them with minimal delay. And all should be embedded in appropriate structures of processors. Therefore hardware, software, and information structures should all be designed to work together. This paper examines the set of issues involved in developing such systems, and describes several designs for structures that process information flowing through.

These information-flow structures are brain-like in that processes continually execute: integrate, differentiate, compare, decide, resolve, and compute a general-purpose repertoire of useful functions. The structures of processors that carry out these processes are all built from neuron-like micro-modular primitive structures. They must work without any global or local controllers, and without any operating system (of today's traditional sort) that synchronizes, handles communication, and avoids contention. These issues must be handled instead by the system's structure and the types of processes executed.

Introduction: Micro-Electronics; Information-Transforming Structures; Appropriate Processes

First, brain-like processes are contrasted with today's serial computers and parallel networks, and the possibilities offered by micro-electronics are described. These make clear what is feasible, and what near-future technologies will make possible - and also the constraints imposed if these technologies are used in conventional designs.

Then we will briefly examine already-developed structures of processes that can be used to transform information flowing through them, when embedded into structures of (ideally isomorphic) physical processors. Data-flow computers that embody Petri nets are a first step toward this goal; but the actual hardware topologies built today to handle what are called data-flow systems only simulate true data-flow processes, with major delays because at the hardware level the actual data-flow topology is destroyed and even more memory is needed that in conventional computers. When used appropriately, pipelines, arrays, and pyramids actually come closer; However these (at least as designed to date) appear to be chiefly usable for specialized purposes.

Next, several designs for possible future information-transforming structures will be described and assessed. These depend heavily upon the use of appropriate processes, ones that can repeatedly iterate, combining and contrasting information moving toward solutions. Several key, brain-like, examples of such massively parallel processes will be given, along with more global structures into which they can be combined.

---

1. Computer Sciences Dept., University of Wisconsin, Madison Wis. USA 53706

The Living Mind/Brain As Our Most Suggestive Approximate Model

The living brain/mind is probably our prime example of an information-transforming structure: It is certainly tremendously successful; it is capable of far more powerful processing than anything we have achieved with computers. It appears to be more like a massively parallel finite-state automaton (that is, a structure of processors that execute processes embedded in them), and to synchronize computations because of the very nature of the processes it executes.

In contrast to today's computers, it neither has a single global controller that executes the same instruction everywhere, in "Single instruction multiple data-stream" "SIMD" fashion; nor is each processor coupled to its own independent controller, in "Multiple instruction multiple data-stream" "MIMD" fashion.

Rather, the transformations the brain carries out appear to be of a kind that can self-coordinate, as processors interact, e.g., to integrate information, discriminate differences, choose among alternatives, and impose and relax constraints. Continuous cycling through these processes appears to approach and lock into solutions and resolutions.

This suggests the following conjecture: the brain uses a repertoire of operations that are appropriate for a data-flow, information-transforming system. These processes themselves serve to coordinate and synchronize as needed.

This further suggests that the particular operations such a system uses must also be chosen with great care, to fit and work with the overall processor/process/information structures.

The brain is an enormously powerful perceiver and reasoner. But it is not clear how efficient the brain actually is (judged by standards used for computers). Indeed, if the brain is as inefficient as it might well be (e.g., we consciously think and attend to only one thing at a time - is the rest of the brain therefore relatively idle?), that may throw some light on the bounds of efficiency in practical situations - that is, when we use massively parallel multi-computers to handle really large and difficult problems.

Technological Possibilities, Both Today and Into the Near Future

Computers are rapidly growing faster and more powerful, roughly doubling in power every 12-20 months. The continuing development of micro-electronic chips, on which increasingly large numbers of components are etched, underlies this rapid and phenomenal increase in computer power.

The Traditional General-Purpose Stored-Program Serial Computer of Today

A traditional serial computer is, basically:

* One single processor (including a controller),
* Plus memory (where data and program are stored),
* Plus necessary input and output devices (to communicate with the external world).

This is typically called a "single-CPU serial stored-program computer." It operates as follows (note how carefully everything must be handled and coordinated):

Information, including the program to be executed, is input to memory.

The "CPU" ("central processing unit") contains a general-purpose processor (that is, a processor capable of executing - possibly with a long sequence of instructions - any possible process) plus a controller that decodes instructions and tells the processor what to do next.

The CPU fetches the next instruction from memory; decodes it; fetches whatever data that instruction needs; executes that instruction; follows that instructions' orders in terms of storing results; and continues in this manner, fetching, decoding and executing instructions. Thus such systems are "serial," since one instruction or piece of information is fetched at a time, from memory to processor, and then processed. They are "stored program" since instructions are stored in memory, fetched and decoded.

The memory simply contains all the information that the CPU works with. Each cell in main memory has a location associated with it, so that information can be pointed to and fetched from or stored to it in one instruction cycle.

Usually many auxilliary, successively slower, memories are also used. Information is swapped into the main memory as needed. It is crucial to have whatever information the processor needs available as fast as possible. This often means that the system has additional memories that are even faster than the main memory - a few very high-speed registers and a cache memory into which information is loaded that clever algorithms anticipate the CPU will need in the future.

This kind of computer can be represented as a 1-node graph: ->O-> with input in and output out. At a somewhat more detailed level, the Processor, Memory, and additional registers and slower memories can be represented each by a different node, with links put where there are actually data paths. But so long as such a system has only a single CPU, it is still a serial computer.

The Enormous Range of Possibilities for Parallel Computers

A parallel computer has more than one processor - anything from 2 to a potentially infinite number. The traditional 1-CPU serial computer is a 1-node graph. Parallel computers are the class of all possible N-node graphs (N>1). In addition, processors, input-output devices, memories, and different types of memories, can all be distributed in whatever manner seems most desirable.

Thus the traditional computer is the simplest (by far), and the set of all parallel computers is overwhelmingly large. We will return to look at some of the most promising topologies. But first it is important to see how large such a system can be made.

Building with Chips (Modular Packages), Transistors (Processors), and Pins (Input-Output)

The 1880s was the era during which chip densities rose from a few thousand to a few million transistors. This startling increase has been accomplished basically by making each component successively smaller.

To do that, an intertwined set of technologies had to be developed, including the following:

* Light microscopes, and now X-ray microscopes, are needed to make the chip's layout plan small enough to fit.
* Appropriate materials (e.g., silicon and gallium arsenide wafers each containing several hundred chips, and the metals and plastics to be placed on and injected into these chips) must be developed.
* Materials must be pure enough so that imperfections do not ruin things.
* Immaculate clean rooms are needed to keep microscopic dust from similarly ruining things.
* Precise, fast, stepping-motor-based machines are needed to control the actual fabrication.
* Precisely controlled ovens are needed to cure and finish the wafer.
* The wafer must be perfectly diced into hundreds of individual chips.
* Each chip must be thoroughly tested, to be discarded if faulty.
* A complex range of good software is needed to help design, simulate, and test out new chips.

* Sturdy, compact packaging is needed to provide input-output for each chip, and to build the appropriate larger structures that combine many chips.

Each individual chip is a roughly 4-6 mm square or rectangle (roughly the size of a fingernail). A wafer is about 6-15 cm in diameter, each typically containing several hundred identical chips. The wafer is diced, and each chip tested and packaged in a plastic carrier with several dozen to several hundred pins that link to the outside world. This carrier is put into a socket, and, along with a number of other chips, embedded into a board. A number of boards are put into the computer's chassis.

The total computer might contain several chassis, that contain in total from one to several hundred thousand chips (plus power supply, memory devices like tape and disk drives, and input and output devices). A small computer might have 1-10 chips, a micro-computer 100-1000, a mini-computer 1000-10,000, a relatively large "mainframe" 10,000-100,000, a "super-computer" several hundred thousand.

But each individual chip will have thousands or millions of transistors! Hence most of the power results from the individual chip's enormous power.

The number of transistors on each chip has doubled roughly every 12-20 months for the past 20-30 years, and this startling steady increase will almost certainly continue for the next 10-20 years, and probably much longer.

Putting millions of transistors on a single 4mm by 4mm chip, all linked as and where needed by tiny buses (wires), is probably the most complex design problem that human beings have ever faced. Today a combination of design art, science of how to embed graphs into the plane, and computer-based tools for graphical displays and design tests, makes such enormous enterprises possible. But only barely, and they are extremely expensive multi-million dollar ventures. Fortunately, the needed art, science, methodology, and technology are all sufficient - and in their infancy, so that we can expect continuing major improvements.

Essentially, one must embed non-planar graphs as compactly as possible onto the 2-dimensional planar surface of a chip. It is possible, and often necessary, to burrow into the chip to make paths where wires must cross; but that's expensive and must be minimized. The more regular the design, the nearer the devices that must communicate, the fewer the needed wire crossings, the more compact and efficient will be the design.

The Size of Component Processors and Memories

What kinds of systems are built with such chips today, and what kinds might be built tomorrow?

Today a traditional relatively powerful CPU (one that executes 1-50 million instructions per second) can be put on a single chip, although more typically 2-10 or so chips are used. Up to several million bits of memory can similarly be put on a single chip. But traditional computers need as much memory as possible, hence many memory chips, so that the CPU is not stopped for want of the information it needs. Roughly 4-40 million bytes (each byte built from 8-10 bits, including several for error-checking and) is usually considered desirable. So a total computer might have 1-10 chips in its CPU, plus 40-400 chips in its main memory.

That is not a large number of chips, and can easily be handled. The cost of the chips will be less than the cost of their packaging into boards and chassis; and the total cost of that will usually be substantially less than the cost of display monitors, additional slower memories, and printers and other output and input devices.

In fact the cost of the chips - which are the actual computer's heart - is small enough so that one is tempted to be lavish, for example, adding still more memory, specialized co-processors (e.g., for

floating-point arithmetic or vector processing), additional high-speed register and cache memories, and processors to handle input from and output to disks and other slower memories.

To Achieve Many Processors in Massively Parallel Structures, Each Must Be Efficient

But in sharp contrast, parallel computers demand the simplest, most efficient possible, designs. We can lavish extras on one single serial processor; but when we have hundreds, thousands, or many millions, the costs multiply out of bounds.

The whole thrust in parallelizing is to speed up processing. One must parallelize the algorithms used as much as possible, and build appropriate parallel architectures and operating systems that execute these parallel algorithms as fast as possible.

It is often true that unless at least 50-100 parallel processors are used, the speed-ups will hardly be worth bothering with. This is because at best N processors give N-fold speed-ups, but in actuality they will often be used with far less efficiency. (What is known as Minsky's conjecture suggests that N processors will give only logN speed-up; but that is clearly not true when appropriate algorithms are run on appropriate hardware.) Today's parallel computers give 2-4 orders of magnitude degradations, and it seems likely that we may always have to live with 2-10-fold slowdowns. That is, to process 100 times as fast (or to process 100 times more information), rather than 100 processors, we may well need 1000, or more.

We don't know today how many problems can be handled with algorithms that can be sufficiently parallelized, and in ways that lend themselves to running on appropriate multi-computers that can make sufficiently effective use of those parallelizations.

* Will we need a different special-purpose multi-computer for each algorithm?
* Or will several different specialized designs be efficient enough, each handling the sub-set of algorithms for which it is appropriate? But will we be able to combine such diverse resources into a single efficient system?
* Or will we succeed in developing a single generally usable parallel architecture, one that can execute any problem with adequate efficiency and speed - or, the ultimate goal, with optimal efficiency and speed?

A Summary Look at the Resources Available, and Needed

Today's chip can easily have 10-50,000 transistors. With especially careful design and packing it can have 100,000 to 2,000,000. In 5 or 10 years these numbers will grow ten-fold, or more.

To store 1 bit of information, 1-10 or more transistors are needed (the more the faster).

The simplest general-purpose 1-bit processor needs 50-500 transistors. A 4-, 8-, 16-, or 32-bit processor needs roughly 4, 8, 16, or 32 times as many.

An even simpler special-purpose processor (e.g., to integrate, differentiate, compare, or choose) might need even fewer transistors, say 10-20.

The Total Computer's Resources Must Be Divided Among As Many Processors as Possible

We can assume that, roughly, the total cost of the computer will be a function of the total number of chips (e.g., 10 times the cost of the chips). And packaging and power problems will set an upper bound to what is feasible. Today "feasible" appears to be roughly $10,000,000-20,000,000 and 100,000-500,000

chips. More reasonably, a "mini" might cost $100,000-500,000 and use 1,000-50,000 chips, a "micro" $1,000-50,000 and use several hundred chips, a "personal" computer $200-900 and use a dozen or so.

Processors come in greatly varying levels of complexity. A traditional CPU needs 50,000-500,000 transistors. But a smaller, slower processor needs substantially less. The most striking examples are the 1-bit general-purpose processors used in the massively parallel arrays like Clip, DAP, MPP, and Connection machine. These are built with only 50-500 transistors for each entire processor!

Roughly, a 32-bit processor will need 32 times as many transistors as a 1-bit processor. The 1-bit processor will be (at least) 32 times as slow, since it must process information 1 bit at a time, rather than in 32-bit chunks (note that actually the traditional 32-bit word is a form of parallelization).

Thus a powerful processor will need a large percent of one entire chip, plus many other chips for the memory that contains all the information it needs to keep it busy. In striking contrast, hundreds of thousands of small and simple processors (but they are still general-purpose) can be put on a single chip - but only if two very important things are done. A. The topology over the processors must be embeddable into the chip with reasonable efficiency. B. Memory must be handled in radically different ways.

True Information-Flow multi-computers will not need traditional large memories. They should be built so that information flows through processors, and processors are judiciously placed so that each is right next to the processor to which it should pass its results. If everything were perfectly designed, the only memories needed would be the registers that hold the information the processor is to work on next; and that information would be put there by up-stream processors just before it is needed.

It is unreasonable to expect things to work out that perfectly. The design of systems that approach these goals will be a major research enterprise. It is not at all clear whether to be efficient these will be specialized, or whether generally usable information-flow systems can be achieved (brains appear to be generally usable and powerful; but are they efficient?). However, the crucial point for now is that memory becomes a relatively small part of the system, rather than by far the largest part.

The Major Multi-Computer Topologies Designed and Built To Date

A multi-computer might be given any possible graph topology. This is such an embarrasment of riches that it is hard to know where to start.

Yet almost all the computers built or even designed to date have had one of the following relatively simple topologies:

* pipeline (1-dimensional array),
* mesh (2-dimensional array),
* tree (graph without cycles whose root links to many leaves via internal nodes),
* pyramid (a logarithmic stack of each logarithmically smaller tree-linked arrays),
* hyper-cube (N-dimensional array with processors only at corners),
* NlogN network (logN banks of N switches that shuffles N pieces of information in logN steps).

Only the first four can be built with many processors on a single chip, short wires, few crossings, and compact designs that use chip area efficiently. But long pipelines are hard to keep full and busy; meshes are slow passing information great distances or executing global operations; trees and pyramids can bottleneck moving toward the apex; and trees are awkward for local operations (for which arrays and pyramids are superb).

There are many other possible topologies (some simpler, some far more complex) that, according to the criteria usually proposed for evaluating topologies, may well be substantially better than these. These

include optimally or nearly optimally dense graphs, De Bruijn networks, and graphs built using a variety of compounding operations. And there are almost certainly many other still undiscovered topologies that are better yet.

Packing Information-Flow Multi-Computers Into Chips

The particular topology makes little difference in terms of building multi-computers with their traditional processors and their very large memories. Since each processor needs several chips, a large network can have only a few thousand processors at most, and processors will be linked off-chip. It is only when several, or many, processors are put on the same chip that we must try to get an efficiently embeddable topology across processors. Small processors, minimal memory, and short links are needed to multiply these numbers. So information-flow structures seem ideal. Since pipelines, arrays, trees, and pyramids can be used in this way, let's look at them first.

Arrays pack most efficiently onto a 2-dimensional chip. Each processor is linked to its 2 (if a pipeline), 4 or 8 nearest neighbors. So wires are short. If each processor is also small and simple, arrays are densely packable.

Each processor can be given its own memory, of whatever size desired. These memories are today relatively small (1K-16K 1-bit words is typical). Small is appropriate: the total memory is very large, and each processor usually needs direct access to only a very small fraction of the total information involved. The only crucial necessary memory is the set of registers used to pass information from processor to processor. So if true information-flow processing is achieved memory might be made substantially smaller, e.g. 256, or even 64, 32, or 8 1-bit registers. Thus instead of memory using almost all the transistors (as in conventional computers), each 50-500 transistor processor might have only roughly 8-500 transistors in its directly accessible memory.

Trees cannot be packed as densely as arrays, but they are a close second. If a tree is packed with its apex at one corner, so that it grows its children nodes moving toward the opposite apex, and nodes are folded into empty spaces (so that the tree becomes irregular) all the chip area can be filled.

Other topologies, and in particular the today very popular hypercube (that is, an N-dimensional cube, where N is usually between 8 and 20 with nodes at vertices only) and NlogN network (that is, logN banks of N switches linking N processors) embed quite poorly on planar chips. So they either use a lot of chip area, or are linked off the chips (but that simply means much larger, more expensive linking hardware must be used).

Toward Good Information-Flow Topologies

Good information-flow structures should pack very large numbers of very simple processors so that information flows among them with minimal need for memories. They will therefore be radically different from networks of traditional computers. Each should be designed to mirror as closely as possible the structure of processes to be executed. This might be a specialized design for a single important problem, or it might be a more general design (e.g., an array, tree, or pyramid) that is reasonably appropriate for a whole range of problems. Or it might be an as-yet-unknown truly general-purpose design (e.g., a network that judiciously links several more specialized sub-networks, with sufficient switches interspersed to reconfigure everything as appropriate). But it is important to note that developing a massively parallel information-flow algorithm when this is carried out within the constraints of chip technology at the same

time designs the hardware structure of processors.

Let's assume that in the near future 10^6-10^8 transistors can be fabricated on a single chip, and that 10^2-10^6 chips can be combined into a single multi-computer.

Each 1-bit-worth of general-purpose or special-purpose processor needs 10-500 transistors.

Consider the following two types of chip, either of which can be used in data-flow fashion:

A. An R-by-C array of processors (R=rows; C=columns).

B. A tree of N processors.

Possibly the simplest use is to treat a 2-dimensional R-by-C array as a 1-dimensional 1-by-RC pipeline. Information will flow into and through the top row, then back through the next row, continuing to snake left-to-right, then right-to-left through the entire R rows. This would be used as are today's vector and image processing pipelines.

But the pipeline would almost certainly be far too big for programmers to be able to fill it effectively. The following appear to be more usable and more promising designs.

The array can input R pieces of information into its first column, and flow that information through to the opposite side. Or it can, via its lateral links, spread and diverge, or converge and combine, information, cycle it back, and effect whatever type of flow is desired (and handleable).

The tree can input information to its leaves (or, if a folded tree is used, to those leaves that are linked to the off-chip world). then this information can be combined, converged, and passed upward to the root. Or information can be input to (or passed to or computed by) the root and broadcast out to all, or to any, of the tree's other nodes. Or information can be flowed upward and also downward, in whatever pattern desired.

In such a system, not every pattern of simultaneously cycling information can be handled. It is up to the programmer/architect to design an appropriate system, and then use it in appropriate ways. And the actual operations executed by each processor become a crucial factor that impacts on the total system.

The following appear to be systems with interesting properties that are worth exploring.

Bundles of Pipelining Arrays

Each array chip can efficiently handle the flow of a 1-by-R set of information. If these pieces of information are related (as they are when the problem is to recognize objects in 2-dimensional images), then local structures of information can be examined, transformed, and combined as this information flows through. If pieces are unrelated (e.g., when separate strings are being processed), each line of processors can work separately, in traditional pipeline fashion.

There can be a certain amount of looping back, e.g., for feedback or constraint relaxation on the chip itself. But this is probably best handled by off-chip wiring that links chips to chips, and also links chips to themselves (e.g., one edge to the opposite edge).

One such chip will execute C processes on a 1-by-R slice of information. N such chips would handle N-by-R arrays of information. But they would be somewhat awkward, since the only place where adjacent slices could interact is off the chip.

Much preferable would be a 3-dimensional technology, where there could be interaction from slice to slice as well as within each slice. Bundles of polymers, or even of genetically engineered DNA or protein molecules, would appear to be most suitable for this. But they are in the farther-off future of promising ideas, rather than proved, relatively mature, technologies.

Design Constraints Imposed by Micro-Electronic Chip Technologies

Let's assume the following design constraints:

Each chip can contain 10^6 transistors for today's designs, and 10^8 for tomorrow's designs (that is, for 5-20 years into the future).

Assuming that simple processors with minimal memory are used, each processor plus its memory will need from 10 to 1,000 transistors for each bit of information that it processes.

Today's chips must be near-planar, although there is a real possibility of 3-dimensional chips or other technologies in the future.

To use the chip effectively, simple, regular processors and memory banks appear to be preferable, since they minimize wasted space. Similarly, wire crossings should be minimized, since they need more space.

Processors should be as close as possible to those processors they communicate with - that is, to which they send, and from which they receive, information.

Pins from each chip to the outside world (over which all communication, including controlling signals, must take place) are severely limited. The limit today is only several hundred, and it appears that this will grow only slowly. Roughly, the number of pins is $O(T^{1/2})$ (T=transistors).

Several Potentially Promising Designs for Information-Transforming Structures

A. Since 2-dimensional arrays are among the most compact and best designs, packing compactly into 2-dimensional chips, they form an attractive basis. They should be usable when desirable as banks of 1-dimensional arrays through which information is pipelined, the processors transforming it along the way. The lateral array links are quite appropriate for assessing and combining information from near-neighbors.

Additional links to more distant neighbors, and also feedback loops, would sometimes also be desirable. But since these would not be as efficiently fabricated on the chip, it might be best to use off-chip wires for these purposes.

Therefore the following range of possibilities should be investigated:

Put an N-by-N array on each chip, with input to one side and output from the opposite side. This has the problem that distant neighbors can interact only via the array topology, and backward flow of information can similarly take place only through the array topology. That means that these processes are slow, since distances through the array are great, and when they are executed they interrupt and stop the ordinary forward flow of information.

B. Put an N-by-M array on each chip (N much larger than M). Now the off-chip links can play a far larger role in providing channels for communication between more distant neighbors, and for feedback. But the limits on pins will probably keep N from growing to more than a few hundred or so.

C. Additional links can be put on each chip, to handle more distant interactions and feedback loops. These might be of any sort, but they all will use a good deal of space, and they should be chosen and designed with great care. Probably a tree-, pyramid- or multi-pyramid-based design would be the best, since they convert distances from linear to logarithmic. Alternately, an NlogN network might be used, but it is appreciably more expensive in chip area.

D. The pipelines might be linked to adjacent pipelines not only in the 2d dimension (that is, from row to row on the chip), but also in the 3d dimension. This would be very useful for processing 2-dimensional information, such as images. But it would be relatively expensive in terms of extra links, crossings and chip area.

E. Rather than use an array or pipeline that at each stage applies the same amount of processing power, the system should be able to converge, or/and diverge, information as appropriate. Processes usually output smaller structures of information than they input, since most processes organize, abstract, make succinct - so outputs should be converged, giving a cone/pyramid structure. Sometimes the volume of information remains constant; occasionally there is a need to expand (e.g., broadcasting to many processors working on different aspects of the problem). This suggests an overall converging-diverging multi-apex tree-linked structure that can expand and narrow as needed. We might try to design large structures of processors from which such information-transforming structures can be carved out as needed.

F. Most important, the particular structure needed for a particular problem, or class of problems, should be built when indicated and efficiently realizable.

G. Several different types of chip could be used to achieve the total design. Routing chips might contain NlogN networks, crossbars, or other appropriate linking topologies. Different types of processors might be put on special chips. Prime candidates are powerful 32-bit processors with their own controllers, arrays, trees, pyramids, and special-purpose processors of various sorts (e.g., to convolve, multiply).

The Feasible Types of Operations for Such Information-Flow Structures

It is crucial that this kind of information-transforming system, to work with speed and efficiency, be given an appropriate repertoire of operations. It must work effectively even though it has neither one global nor many local controllers. Processors cannot afford to wait and do nothing until the appropriate information arrives. As many processors as possible should work as much as possible, productively. Processors will not longer have an external controller to handle contention, synchronize, and in general tell them what to do.

There appear to be a number of feasible operations that fill these requirements, and ones that are highly appropriate at that. This is not surprising, since the brain does all the things we want such a system to do, and these are chiefly brain-like operations.

Integration of Information (to Combine)

Possibly the most basic operation is simply one that integrates. A neuron-like processor that receives many inputs combines them, e.g., by adding. This summation of incoming impulses might take place over time as well as over space. It might be modulated, e.g., decaying over time or space, or scaling down large changes. Thus the brain appears to modulate incoming impulses using Gaussian bell-shaped weights. This very naturally gives a detector a smaller voice the more distant it is.

This is especially appropriate when a processor receives redundant inputs from several nearby processors. For example, all might be identical detectors of the same feature, but at slightly different locations, or different detectors of the same feature, only some of which can be expected to succeed. And since each processor repeatedly, redundantly, sends its message that the feature was detected, these messages are best integrated, and also scaled down, as appropriate.

Integration can also serve to average, smooth, and eliminate random anomalies, as in noisy images.

More generally, integration gives a linear combination of the several components combined. This can be useful for many purposes - but only up to a certain point: where non-linear combinations are needed (as for spatially oriented structures), more complex compounding functions must be used, as described below.

It is important to note that in all these cases processors do useful work simply by repeatedly integrating, since repetition combines information over time. Most important, when this is part of a network where other processors are attempting to converge toward some solution, goal, or desired state, repeated integrations give a running summary of the current state of affairs.

Differentiation of Information (to Contrast)

Differentiation is the natural companion to integration - again, both in space and in time. Parallel integrations build up local summary pictures; differentiation assesses whether they contrast.

Brains routinely integrate nearby impulses, and at the same time differentiate. Neuron-like units can handle this by adding to integrate, and subtracting to differentiate.

Thresholding or in Other Ways Deciding Whether to Fire Out

The simplest kind of processor would simply fire out its integrated result (if it were a differentiator this would be a function of negative as well as positive values).

It is often useful to set a threshold below which the processor does not fire. Now there are several alternative possibilities when the threshold is exceeded: 1. The output is simply the input, or some function of the input; 2. The output is some independent value that the processor stores or computes; 3. The output is some function of both 1. its inputs and 2. its independent values.

A type 1 operator whose threshold is 0 acts like a basic integrate-differentiate operator, since it simply fires out what was input to it. A type 2 operator can effect any desired conversion to symbolic information.

Once again, the threshold function is intimately related to integrate-differentiate, and the system can usefully continue to combine, contrast, and threshold.

Compounding (Non-Linear Combining)

Rather than simply integrate, several inputs can be combined in a non-linear manner. This is crucial for image-processing systems, in determining objects' shape and other structural information. For example, depending upon how a vertical and horizontal bar are positioned the result might be any of 4 angles, or a T, +, or still other shapes.

Once again, it is often useful to do this repeatedly, to get a firmer estimate, and to continually evaluate what might be there as a function of attempts to iteratively converge.

Comparing Two or More Values

Another useful operation makes comparisons - again, in space, in time, or in both. This serves a variety of purposes, from noticing simple changes to leading toward decisions.

These Kinds of Information-Flow Operations Build Into Useful Larger Structures

Each individual operation of this sort can be usefully repeated if it is an appropriate part of a larger structure of operations. For example, outputs from functions of this sort are usefully integrated over time,

then these results differentiated and thresholded to find peaks, as when an object is resolved.

Choosing (Winner-Take-All)

Comparing indicates which is "more"; but then a whole structure of comparisons can choose a winner.

This can conveniently be handled by flowing the alternatives up through a tree each of whose nodes makes a local comparison and passes along the better. The apex of the tree's choice will be the grand choice, the winner over all the alternatives.

Alternately, a node representing each alternative can fire negatively into all other nodes, so that after iterations only the most highly activated node still fires.

Building Larger, More Complex, More Abstract, More General Structures

Trees and Pyramids that successively build larger compounds, using structures of operations of the sort suggested above, appear to be among the most promising relatively general examples of systems that can hierarchically compute complex functions of the sort needed for intelligent perception and cognition.

At least these are the most widely used to date. They reflect the basic requirements: that complex processes be decomposed (divided-and-conquered) into suitably simple processes - from which the tree/pyramid structure follows - and information be flowed upward until the processors toward the apex complete the global process.

Almost certainly, better structures can be found for a particular problem, for the structure of the total system should, ideally, reflect the structure of processes needed to solve the specific problem being handled. For these to be of the efficient information-flow type they would best be built to handle processes that can usefully be repeated, moving toward solutions.

It is too early to know whether it will be possible to achieve structures of this sort that are both general-purpose and also powerful and efficient.

Constraining-Relaxing, To Reduce Possibilities and Settle into Conclusions

The differencing and compounding operations can impose constraints. Successively combining, differencing, and choosing, then noticing how results of choosing change (by applying this same structure of operations) can effect resolutions of the various influences, and relax to what are judged good, or stable, or sufficient, states.

Reinforcing a Continuing Sequence of Converging Decisions

In general, a network that repeatedly integrates, differences, compounds, compares, and chooses - then attempts to monitor (by integrating, differencing, etc.) its success at choosing, can move successively toward better and better results, yet as appropriate decide to stop, judging no more can be done.

The Possibility of Still Other Structures of Usefully Repeated Information-Flow Operations

The operations described above are only examples of what information-flow systems can usefully do. They were mentioned because they are brain-like, and using them much can be done. But they also ap-

pear to be sufficient for much of perceptual recognition and cognitive reasoning.

An important goal to work toward is a general-purpose, and also efficient and powerful, set of such operations, ones that need little synchronization and are usefully repeated. These should be of the sort that can be embedded efficiently into an appropriate network of processors.

The crucial point is that processors, processes, and information should all be structured together, so that information is transformed as it flows through structures of processes appropriately embedded into structures of processors.

Acknowledgements and References

This paper grows out of research conducted over the years under a number of grants from the National Science Foundation, developing parallel-hierarchical brain-like systems for perception and cognition, and investigating appropriate parallel hardware architecture.

The following following books contain a number of references to this research, and to related work by others.

General References

[1] Uhr, L., *Algorithm-Structured Computer Arrays and Networks: Architectures and Processes for Images, Percepts, Models, Information.* New York: Academic Press, 1984.

[2] Uhr, L. (Ed.), *Parallel Computer Vision,* Boston: Academic Press, 1987.

Uhr, L., *[3] Multi-Computer Architectures for Artificial Intelligence: Toward Fast, Robust, Parallel Systems.* New York: Wiley, 1987.

[4] Uhr, L., Preston, K., Levialdi, S. and Duff, M.J.B. (Eds.), *Evaluation of Multi-Computers for Image Processing.* New York: Academic Press, 1986.