CONSTRUCTING MULTI-LEVEL MULTI-COMPUTER NETWORKS

by

Leonard Uhr

Computer Sciences Technical Report #572

December 1984

CONSTRUCTING MULTI-LEVEL MULTI-COMPUTER NETWORKS

Leonard Uhr

Department of Computer Sciences
University of Wisconsin
Madison, Wisconsin

This paper presents a variety of different techniques, ones that are now potentially at the disposal of computer architects, for constructing well designed, very large, algorithm-structured, architecturally feasible multi-computers that have desirable mixes of local and global properties. It examines a few of the enormous variety of ways in which potentially extremely large multi-computer networks can be configured, using these techniques.

A number of different construction levels and techniques are described. Any, several, or all of these can be combined, in a variety of ways, to build a total architecture. Multi-computer networks already built, designed or proposed are briefly surveyed and examined from the point of view of these multi-level design possibilities. And several examples are given of possible multi-level designs.

1. INTRODUCTION

During the next 10 to 20 years VLSI technologies will make possible multi-computer networks constructed from many thousands or millions of individual computers. This paper explores how such potentially gigantic networks might be configured.

A great variety of interconnection topologies have been proposed for multi-computer networks. (Section 9 below briefly surveys and discusses these from the point of view of the multi-level constructions developed in this paper.) In almost all cases the topology that is chosen is either: a) specialized for efficient execution of certain types of problems (e.g., arrays for image processing, like Duff's 1976, 1978, CLIP4; Reddaway's 1980 DAP; and Batcher's 1980 MPP), or b) homogeneous and, it is hoped, general networks (e.g., Wittie's 1978 MICRONET, and augmented trees, like Despain and Patterson's 1978 X-trees, and Goodman and Sequin's 1981 Hyper-trees).

These constructions are usually completed in one homogeneous step. That is, the overall graph that represents the multi-computer is not composed of smaller sub-graphs (which in their turn may recursively be composed of smaller sub-graphs). Major exceptions are Cm* (Swan et al., 1977), with clusters joined into a star, and cube connected cycles (Preparata and Vuillemin, 1979), with N-node polygons embedded in the nodes of an N-cube.

Essentially, any possible graph can be used as the basis for a multi-computer (the graph's vertices represent the hardware's nodes - whether computer, processor, memory or other resource; the graph's edges represent the wires that link these nodes together). This is far too large a set of possibilities to be explored completely, and a number of different, more or less ad hoc choices have been made of particular graph topologies. These include, among a large variety of other proposals, arrays, trees, N-cubes, stars, pyramids, lattices (see

Stone, 1980, Hwang and Briggs, 1984; Uhr, 1984).

Some of these topologies are chosen to fit and work well for particular (classes of) algorithms or programs; they thus lead to algorithm-structured, process-structured, task-structured architectures. Others are chosen for more or less formal reasons, for example to maximize global density - that is, to pack as many nodes as possible within a graph of a given diameter (the shortest distance between the two nodes that are most distant from one another) and degree (the number of links from each node). Almost all of these multi-computers tend to be homogeneous; that is, with the same design considerations dominating all their parts and details and therefore with much the same local and global topology everywhere.

But the best local topology may well be quite different from the best global topology. Usually in a well organized program most processes will interact with only a few other processes. If these are all mapped into sets of nearby hardware nodes and executed with minimal and efficient message-passing, the network's execution of the program can be speeded up substantially. But (even though the programmer should try to make this a rare occurrence) many if not most programs will occasionally need to send messages between nodes that are more distant from one another. Therefore reasonably good global message-passing distances are also important.

There are a variety of desirable characteristics that might be chosen for a multi-computer's topology, related to the flow of processes specified by the algorithm(s) that it will execute, and also to such formal characteristics as density, connectivity, and symmetry. It is not at all clear today what set of formal criteria should be used, or whether there is any best set for all problems. But for purposes of illustration the characteristic of density (which has generally been considered the most important, and about which probably the most is known) will be emphasised in the examples given below.

2. SPECIFYING THE LOWEST-LEVEL ARCHITECTURE(S)

At the lowest level any kind of computer can be chosen for embedding into a node. In many cases each lowest-level node will simply be a general-purpose computer, whether a 16-bit M68000, a 32-bit VAX or transputer, a 64-bit Cray, a 1-bit CLIP4 node, or an 8-bit Z80; or a custom-built system.

There are several other interesting possibilities, but they are best introduced after all the levels have been

Since any type of architecture can be designated for each node, different nodes can have different architectures. Problems (but these are relatively minor, since a great variety of solutions are possible) arise only when we link several different types of lower level nodes into a larger topology.

3. LINKING RESOURCES VIA A COMMON BUS OR OTHER (VIRTUAL) COMPLETE GRAPH

A bus can handle only a few nodes, because of limited bandwidth. But a bus is a convenient way to link a small number of computers together, and therefore to increase the size of a multi-computer by roughly a factor of 8, 16, or, possibly, a bit more.

It will often also be desirable to have devices that handle memory (e.g., rams, disks,

tapes), input (e.g., keyboards, TV cameras), and output (e.g., printers, monitor screens) link as nodes to a common bus to which computers also link, rather than form a part of an individual computer node.

Today buses are occasionally used to link together more than 16 resources, as in Ethernet-like systems of many personal computers. But there are stringent limits to using a bus to cluster several computers that are all working together on the same problem, with the goal of speed, efficiency and power. More and more nodes can be linked to the bus, but relatively soon bus traffic will bog the whole system down. Typically performance begins to degrade with few more than 3, 8, or 16 computers: sometimes more computers can be added (usually because they communicate rarely). But an entire multi-computer on a single bus is a dead end impossibility: the bus could never handle even a few dozen or a few hundred, much less a few thousand or a few million nodes.

A bus can be used to increase the number of nodes by roughly one order of magnitude. It is probably true that this first level is the the simplest and most obvious place to use a bus to link components, but a bus could similarly be used to form clusters at any level.

Rather than use a bus to link together 8, 16, 32, or a few more nodes, a variety of other structures might be employed. The most obvious choices are a crossbar, ring (polygon), complete graph (every node linked to every other node), and star (one "center" node linked to every other node, e.g., with a "blackboard memory" forming the central node). But these all share with the bus the inability to handle more than a few nodes. The crossbar and the complete graph need far too many links; the ring and the star have severe bandwidth problems. Crossbars need N*N switches. As the number of nodes increases, complete graphs rapidly become very expensive and then impossible to implement, since they need N-1 interfaces and wires to each computer node. Rings have the same bandwidth problems as buses. Stars rapidly become unusable, since all information must be passed through the center node, which quickly becomes an impossible bottleneck. Other possibilities include a reconfiguring shuffle network, or one of the optimally dense Moore graphs (for our purposes best thought of as trees augmented with links between leaves that reduce all worst-case diameter distances by one half, to the radius distance). In all these cases the operating system and the programmer view the cluster as a (virtual) complete graph. That is, from the programer's point of view every processor and other resource is linked to (or one might better say is equi-distant from) every other.

4. BUILDING GOOD LOCAL STRUCTURES

At the next level, a number of nodes can be linked together into what will be named a "structure." Each node might be a single computer (whether a 1-CPU serial computer or some other architecture, like a pipeline, array, or other multi-computer). Or one node might be a processor, another a (shared) memory, another an input or output device, another a disk. Or an entire cluster of the sort described above can form a single node of the structure.

A wide variety of structures have been identified and suggested: only a very small number of these have been built.

The structures that have most commonly been used are complete graphs or stars (note that structures can also serve to cluster, as alternatives to a bus). Much more attractive are structures that are chosen for good formal properties, like density, connectivity, ease of finding paths between nodes, symmetries. Good examples are the optimally dense Moore graphs (the 10-node, degree=3, diameter=2 Petersen graph; and the 50-node, degree=7, diame-

ter=2 Singleton graph; see Bondy and Murty, 1976), other graphs with good density (e.g., augmented trees), and also small arrays.

5. COMBINING STRUCTURES INTO GOOD COMPOUNDS

At the next level, several structures can be compounded together. A variety of good compounding operations have recently been discovered that give relatively dense new graphs. These include a number of small graphs, ranging from less than 100 to several thousand nodes, as well as much larger graphs, with many thousands or millions of nodes, some of which are substantially denser than any structures of similar size used or proposed for actual multi-computer architectures.

5.1 Compounding Several Different Structures Together

During the past few years a large number of new topologies have been constructed using several recently devised compounding operations (Bermond et al., 1982); these new topologies are two to ten or more times as dense as the previously densest discovered graphs (Storwick, 1970). But even the graph topologies that Storwick summarized in 1970 are substantially denser than those used in almost all of today's actual multi-computer architectures. This is true despite the fact that most computer architects appear to consider density the most important criterion, and often try to maximize their architectures primarily for density.

Density has been the overriding single criterion for most research attempting to find "better" topologies for multi-computers. But a network architect who wants to choose a multi-computer topology that maximizes any other criterion, or any set of criteria, can similarly search for appropriate compounding procedures. Such compounded topologies have two desirable properties - their relatively good global density (possibly along with whatever additional set of properties for which they have been chosen), and the good local properties of the individual structures that are compounded together.

5.2 Embedding Graphs into Overall Structures

Rather than use a compounding operation to combine a number of structures (or clusters) together into a larger whole, structures (or clusters) can be embedded into the individual nodes of some appropriate larger global graph.

The cube-connected cycle (Preparata and Vuillemin, 1979) is such a structure, since it uses an N-cube to globally link a simple structure, a cycle (polygon), with N-1 nodes, each of which is joined to a different edge of the original N-cube. But, as the tables of densest graphs discovered to date show (Bermond et al., 1982), N-cubes are not nearly so dense as de Bruijn graphs (or even pure trees), and cycles are not nearly so dense as Petersen graphs (or for that matter most other graphs). However, other criteria, as suggested above, may override density. For example, when the cycle is used as a pipeline of processors through which data flow and are thus transformed, it can serve powerful algorithm-structured purposes. And the N-cube appears to many people to be attractive, e.g., from the point of view of ease of addressing the recipient of a message, and of finding alternative paths to that recipient if some paths are blocked.

A simple way to compound graphs is to embed an N-node graph into each node of a global graph each of whose nodes touches N+1 edges, and link each of these edges to a different node in the embedded graph.

But it is not necessary to embed the same graph structure into every node of the global graph. Nor is it even necessary to choose a compound or a structure that has the same number of nodes as the degree of the node into which that architecture will be embedded. We can assume that the bandwidth needed between the embedded graphs is extremely small compared to the bandwidths within these graphs, because message-passing between embedded graphs via the global graph will be relatively rare and limited (but at times necessary). Therefore it seems best to design at this level in terms of whatever bandwidth is feasible at a reasonable price. For example, a 50-node Singleton graph might link to a degree=8 de Bruijn graph from only eight of its nodes (see Uhr, 1983, 1985).

6. EMBEDDING SEPARATE TOPOLOGIES INTO A TOTAL GLOBAL ARCHITECTURE

At the top level, an overall structure can be built using a different type of procedure, one that tries to maximize some set of properties globally, without compounding good local topologies. In particular, a globally dense topology can be chosen to build this overall architecture. Lower-level structures can then be embedded into that global architecture's individual nodes.

6.1 Relatively Dense Global Constructions for Small and Large Graphs

For very small graphs (with fewer than a hundred nodes) it might be attractive once again to consider using the Moore graphs, or other reasonably dense graphs that have been discovered.

For graphs larger than 10,000 or so nodes the de Bruijn (1946) graphs (widely known as binary shift registers, and independently proved to be the densest discovered to date by Imase and Itoh, 1981) appear to be especially attractive candidates. A de Bruijn graph is for our purposes best thought of as a tree that is augmented by:

a) adding N-1 links to each bud,
b) increasing the degree of the graph by 1, and
c) linking according to Imase and Itoh's formula.

This gives a graph whose local structure is a degree=N graph that is a tree, but whose global structure is far denser than a tree, because of the added linkages that pull together otherwise distant parts of the degree=N tree into a degree=N+1 graph.

It appears from Goodman and Sequin's (1981) simulation runs that if average distance between nodes is used as the criterion of density (rather than diameter distance) then their Hyper-tree gives graphs roughly 10% denser than the de Bruijn graphs. Therefore if one is willing to use average distance (which may well be preferable) rather than worst-case diameter distance, the Hyper-tree becomes a most attractive candidate.

De Bruijn graphs and Hyper-trees come into their own only when we construct topologies with many thousands of nodes - or even more. But we are now considering using them for the highest level of global interconnections. Each of their individual nodes might contain a compound of tens to hundreds of structures, each containing ten to a hundred individual computers, or clusters of ten or so computers each. So the sizes with which we can poten-

tially deal are enormous. This probably suggests that when thousands of nodes are linked using de Bruijn graphs or Hyper-trees each individual node should be relatively small and simple, possibly only a small cluster or a single computer, and possibly a very simple computer at that. That is, only a few of the levels need be used.

6.2 Embedding High-Level Architectures into Low-Level Nodes

The architectures suggested so far can be made strikingly larger still, literally without limit, simply by using one of the higher-level graphs constructed by whatever set of procedures might be chosen as a single lower-level node. For example, the highest-level de Bruijn structure produced by embedding into a de Bruijn graph a compound of a structure of a cluster of individual nodes might now be used to replace the individual computers at the lowest level, and this procedure repeated as many times as desired.

7. RECONFIGURING NETWORKS OF SWITCHES TO INCREASE FLEXIBILITY

Networks of switches can be introduced to reconfigure the system, potentially in a great variety of ways.

The most widely used procedure to date has been an NlogN array of 2 by 2 switches linking computers to computers, or processors to memories. This allows the set of N processors to shuffle N pieces of data to or from N memories or processors. Essentially, this is just another graph structure that is introduced into the total structure, except that this is a structure of switches rather than of computers or processors. (Note that whenever computers are linked together into a network this switching capability is also always present, as a part of the computers' basic capabilities. For each computer must be able to switch to input from or output to any of the other computers with which it is directly linked.)

Reconfiguring switches can be used for other purposes. Among the most promising (Snyder, 1982) are to change one graph into another (for example, to change the topology back and forth between an array and a tree), and to reconfigure around faults.

Thus reconfiguring capabilities offer a tool that cuts across levels, and might fruitfully be used within any level.

8. A SUMMARY OF THE PROPOSED DESIGN PROCEDURES

The following summarizes the basic hierarchy presented above:

1) Individual node (computer or other basic resource);
2) Bus-linked clusters, or other (virtual) complete graphs, of individual nodes;
3) Structures of individual nodes or clusters;
4) Compounds of structures;
5) A total global architecture.

This can be expanded in at least three ways, to give still greater flexibility and substantially larger possible structures.

A) Step 2, the linking of a dozen or so individual nodes via a single bus or other simple structure, can be introduced after any of these steps, and not only after the first step.

B) Step 1 can start with any configurations as its primitives - not simply individual computers, processors or other resources, but also higher-level clusters, structures, compounds or total architectures.

C) A reconfiguring network of switches can be introduced at any step.

The following revised hierarchy of design procedures includes the possible extensions, by embedding higher-level architectures into lower-level nodes, and by clustering a few nodes via a common bus or other (virtual) complete graph. (Note that the total architecture includes individual, cluster, structure, compound, architecture.)

1') Individual node or other architecture:
2) Bus-linked clusters (or other virtual complete graphs) of individuals or architectures;
3) Structures of individual nodes or clusters;
2') Bus-linked clusters (or other virtual complete graphs) of individuals or architectures;
4) Compounds of structures:
2") Bus-linked clusters (or other virtual complete graphs) of individuals or architectures;
5) A single global architecture over compounds, structures, clusters or individuals.

9. SOME TOUCHSTONE EXAMPLES OF MULTI-COMPUTER TOPOLOGIES

The following multi-computers are briefly described from the point of view of the topology of the architecture that links together their individual computers. The systems in this survey are then briefly examined and evaluated in the light of the construction procedures and design issues presented in this paper. This should put the reader in a position to make similar evaluations of still other multi-computers of interest.

Among the best known multi-computer topologies are the following:

A) S-1: A crossbar is being used to link a small number (16 is the immediate goal, 64 is a possible future goal) of very powerful computers (each with the power of a supercomputer like the Cray-1). (Widdoes, 1980).

This system first attempts to put as much power as possible into each single computer. Then it uses a crossbar to link computers into a single cluster, one that simulates a complete graph. It therefore takes no advantage at all of the many different topologies available, or the possibilities of multi-level construction. For the near future this is an interesting and potentially powerful alternative. But once the single computer has reached its ultimate limit in power there will be a need to link more than a small number of them together.

B) C.mmp: A large common shared memory links to 8 or so mini-computers (PDP-11s were used). (Wulf and Bell, 1972).

The common shared memory quickly becomes a severe bottleneck. To enlarge such a system, clustering, or some other higher-level structurings, are needed.

C) Cm:* A small number (8 or 10) mini-computers (PDP-11s) are linked to a shared bus, thus forming a cluster. Several such clusters (3 or 5) were linked together to form a star. (Swan et al., 1977).

This is a good example of a 2-level system with compounds of clusters. But there are several attractive alternatives for the basic cluster (e.g., Petersen graphs, augmented trees). And compounding procedures that yield denser graphs might be attractive alternatives, especially as the number of computers in the total system grows larger.

D) Crystal: A somewhat larger number (20 to 40) of somewhat more powerful mini-computers (PDP-11/VAX-750s) are being linked to a common bus-ring. The Crystal ring can be partitioned to link together several separate clusters, and the set of computers can be partitioned into subsets each of which is executing a different program. (DeWitt et al., 1984). (Earlier ring-based systems include Farber and Larson's, 1972, and Rieger's 1980 Z-mob.)

The ring serves, much like a bus or a crossbar, to give (virtual) complete connectivity. But it shares with the bus possible bandwidth problems. And as with both bus and crossbar it cannot be made much larger than 16 or 32 (or a few more) computers in toto.

E) X-Tree, Hyper-Tree: A tree (usually a binary tree) is formed, but it is augmented with additional links (thus making it a graph with loops) into one of a variety of topologies (Despain and Patterson, 1978). The Hyper-tree (Goodman and Sequin, 1981) appears to be the augmentation that achieves the greatest density.

This kind of augmented tree appears to be among the best of the topologies discovered to date, from the point of view of density. It is also often appropriate for tasks where information is to be gathered together and absorbed, and/or disseminated and broadcast. Pure trees are poor for tasks where large amounts of data must flow through the single computer at the root, or through other nodes near the root. But the augmentations, if chosen judiciously, can offer good alternative paths to circumvent this problem.

F) The CLIP, DAP and MPP arrays: A large number (4,000 to 16,000) of each very small and simple general-purpose computers (specially designed and fabricated 1-bit processors) are linked together into a 2-dimensional grid, each computer linked directly to its 4 (or, in the case of CLIP, 8) nearest neighbors). (Duff, 1976, 1978; Reddaway, 1980; Batcher, 1980).

An array is a very simple kind of compound, one that tesselates and links together (small) arrays.

Arrays are specialized to handle very large numbers of local window operations in parallel. They are therefore very attractive architectures for several important domains of tasks, including image processing, pattern perception, and numerical problems that can be formulated in terms of near-neighbor operations on matrices of numbers. But an array by itself has a major input and output problem: Simply because the array is so massively parallel that it can gobble up information so fast, it is difficult and expensive to pass information in and out fast enough. There is a real need to develop good larger graphs that combine an array with other resources necessary for an efficient total system (e.g., a conventional serial computer, or a network of conventional computers, plus mass memories and input and output devices).

G) N-cube: 2^N computers are linked together to form an N-dimensional cube, with each computer at one of the N-cube's nodes. (Wittie, 1978).

The N-cube is a good example of a global architecture. It is not especially dense (e.g., a tree without any augmentations is already denser). But it has several desirable properties, including the fact that it is straightforward to find a path, and (when blocked) alternative paths, between nodes.

H) Cube-connected cycles: An N-dimensional cube is used to link together N computers (linked via a ring into a small cluster) that are embedded in each node of the N-cube. (Preparata and Vuillemin, 1979).

This is a good example of a 2-level system, with the N-cube combining into a total global architecture the cycles (polygons) embedded within its nodes. But since neither N-cube nor cycle is especially dense, one might consider using different micro and macro structures.

I) Systolic systems: A special-purpose "systolic" processor (designed to efficiently execute a particular process, e.g., convolution, on a stream of data pipelined through it) is added to a conventional high-speed processor or network. (Kung and Song, 1982, Kung and Lam, 1984).

A systolic processor appears to work best in a system designed for a particular special purpose. It is not clear how to combine several systolic processors together with a network of conventional computers. A general-purpose systolic processor has been fabricated, but when a large number of these are combined into a multi-computer the result is very similar to an array or a pipeline.

J) Virtual common memory systems: A set of N processors is linked to a set of M memories (N usually equals M) so that each processor can fetch from and store into any of these memories, usually (but not always) in a single cycle. (Batcher, Staran, 1973, 1974; Gottlieb et al., Ultra-Computer, 1983).

These systems require logN banks of N 2 by 2 switches. Therefore the Ultra-computer's 12 by 4,096 array of switches probably pushes this approach to its limits. This is probably best viewed as a way of achieving virtual complete connectivity for up to a few thousand processors, rather than the somewhat more complete connectivity that a crossbar, bus or ring can achieve for up to a few dozen processors.

K) Reconfigurable systems: A set of N processors is linked to one another via a reconfiguring shuffle network (Siegel et al., 1979, 1981; Briggs et al., 1979; Lipovski, 1977).

Here the straightforward approach is to link N processor nodes via an NlogN array of switches to themselves (rather than to N memories). But there are a large number of other possibilities, including mapping one graph into another, reconfiguring around faults (Snyder, 1982), and other types of partial reconfiguring, that are virtually unexplored to date.

L) Pyramids: A set of successively smaller arrays is linked together, forming a pyramid with an N by N array at its base, and logN arrays in toto (Tanimoto, 1981, 1983; Dyer, 1981, 1982; Uhr, 1981, 1984; Schaefer, 1985). One variant compounds a tree (or a suit-

ably augmented tree) to an array at its base.

A pyramid combines the good massively parallel local window operations of an array with the good message-passing, data-convergence, and broadcasting capabilities of a tree. But, much like the array, it appears to need to be combined with other resources, including a network of traditional computers (Uhr, 1983, 1985).

Virtually all of the multi-computers surveyed are 1-layered or 2-layered systems. The system typically uses either a single overall topology (like an array, N-cube, or augmented tree), or it compounds clusters together, or it tries to link as many computers as closely as possible in a virtual approximation to a complete graph. As multi-computers grow bigger it will become more and more desirable to use more of the possible levels of construction in the same single system.

10. EXAMPLE ARCHITECTURES, USING THE PROPOSED DESIGN TECHNIQUES

The following architectures are examples of attractive, or occasionally curious, candidates for serious examination and possible implementation - first from the point of view of the structural criteria examined in this paper, and also from the point of view of algorithm-structured systems. That is, these architectures appear to be indicated even before considering whether they handle efficiently particular classes of algorithms. But some also appear to meet the algorithm-structure criterion - and that is probably an even more important criterion. Note that, for purposes of illustration, these architectures are larger, and make use of more levels, than one would seriously consider designing and building in the immediate future.

I. a) Cluster 10 or so computers on a bus.
b) Link 50 clusters into a Singleton graph.
c) Compound 51 Singleton graphs (this gives an unusually dense graph, with degree=8, diameter=5).
This results in a total architecture with 25,500 computers, linked into local clusters of 10, and larger structures of 500 each.

II. a) Start with a relatively large de Bruijn graph.
b) Embed a 10-node Petersen graph in each of its nodes.
c) Each node of the Petersen graph can itself be a cluster linked by a bus, or by a Petersen graph.

III. a) Use the Petersen graph both as a basic cluster, because 10 computers often appear to be all that usually need to interact heavily,
b) use the Petersen graph at the next level also, to structure these clusters together.
c) Compound the Petersen graphs, e.g., by linking 11 together.
d) Use each of these compounds as a node in a Hyper-tree.

III'. Simple variants to III include:
a) Use structures of 50-node Singleton graphs (rather than Petersen graphs), compounding, e.g., 51 of them together.
b) If still more nodes are needed, use a bus or a Petersen graph to cluster 10 or so com-

puters for each node in the Singleton graph.

c) Build a global architecture over the compounds using a de Bruijn graph rather than a Hyper-tree.

IV. a) Use pyramids as 9 of the 10 lowest-level nodes in a Petersen graph, and a bus-linked cluster of micro-computers (e.g., M68000s) and disks as the 10th node.

b) Input data to the base of these 9 pyramids such that they form a 3 by 3 array of a larger pyramid.

c) Compound 2 such Petersen graphs with 9 other Petersen graphs, where the other 9 contain either:

1) only traditional micro-computers, or

2) 6 processors, plus 3 memories (with each processor directly linked to and sharing 2 memories) plus 1 disk.

IV'. Simple variants to IV include:

i) Use 50-node Singleton graphs, linking a 7 by 7 array of pyramids plus one minicomputer or cluster into 2 of them, and linking 51 of these Singleton graphs into a compound.

ii) Decompose a pyramid into one or another set of sub-pyramids (e.g., a few from higher layers, as well as a scattered set from lower layers) and embed them in a suitably scattered sub-set of the nodes of a Petersen, Singleton or mesh-structured graph. E.g., link the 4 4 by 8 sub-arrays of 1-bit processors in the 16 by 16 layer of the pyramid, plus another set of 256 processors spaced 8 apart in the 128 by 128 layer, as 8 of the nodes in a Petersen or Singleton graph.

V. a) Structure 50 computers into a Singleton graph.

b) Compound 51 Singleton graphs.

c) Cluster 10 such compounds together, using a bus or a Petersen graph.

d) Structure 50 of these clusters into a Singleton graph.

VI. For a really large graph that makes use of all the possible levels:

a) Cluster 10 nodes together with a bus or Petersen graph.

b) Link 50 clusters into a Singleton graph.

c) Compound 51 Singleton graphs.

d) Embed a copy of this compound into each node in a de Bruijn graph or Hyper-tree.

11. DISCUSSION

Only when researchers begin to design multi-computers that (from today's perspective) are extremely large will it be necessary or desirable to make use of all, or even most, of the construction levels described in this paper. But it will soon be feasible to build such massively parallel systems. And there are a number of extremely difficult and important problems (e.g., computer vision, artificial intelligence, data base management, 3-dimensional modeling, weather prediction) for which they are needed.

11.1 Heterogeneous Architectures

There are several important variations that can be made to the total structuring within the

different parts of the total architecture.

Each local sub-graph in the global graph that forms the overall architecture (e.g., each structure in a compound, or cluster of structures or compounds) can be the same, giving a homogeneity that allows for flexible allocation of programs' procedures to resources by the operating system. For example, 1024 10-node polygons or 1024 10-node Petersen graphs might be embedded into a 10-cube.

But it is also now possible to construct heterogeneous architectures that contain different graphs in different regions - so long as all these local graphs can be properly linked externally to other graphs in the larger architecture. For example, 1008 Petersen graphs and 16 10-node polygons might be embedded into a 10-cube; now the polygons could be used when short pipelines are appropriate, the Petersen graphs when dense local MIMD networks are needed to handle message-passing.

Any number of different types of local graphs can thus be used, so long as each has the proper number of nodes with which to link it into the larger structure.

11.2 Tailoring Links to Message-Passing Bandwidth Requirements

This means that the local graph can, when desired, have more nodes than the number needed for this external linking. In that case, only some of the local graph's nodes need be given an extra link to other parts of the architecture. The others can be left with no additional links. But they will often be good candidates for linking to specialized resources like input devices, output devices, mass memories and special-purpose processors.

The bandwidth of channels linking sub-graphs can thus be tailored to the needs of message-passing, much as the individual local graphs can be tailored to the needs of processing.

12. SUMMARY

A) At the lowest level, each *individual node* can be built from any desired simple or complex architecture of resources (processors, memories, controller, input devices, output devices, Hyper-trees, systolic processors, pyramids, etc.)

B) Up to a few dozen of these nodes can be linked into a *cluster* via a common bus, or other (virtual) complete graph (e.g., a crossbar, ring or star).

C) Ten to hundreds of individual nodes or clusters can be configured into a *structure* that is relatively dense - that is, one that has as many nodes as possible packed within a given diameter and degree, or that has any other desired formal or/and algorithm-structured set of properties.

D) Ten to thousands of structures can be linked together into a *compounded structure* using any of several compounding operations that make graphs that are relatively dense, or/and that have any other desired property or set of properties.

E) Hundreds to millions of these compounds can be linked together into a *total global architecture*, using any of several structures with appropriate global properties.

Not all of these levels need be used, and they need not and almost certainly would never all be used to maximum feasible size. But two additional construction tools make even larger structures possible.

1) Nodes at any level (not merely the first) can be linked to a common bus or other virtual

complete graph. This increases the number of nodes roughly by one order of magnitude each time that it is done.

2) "Any desired architecture" at level A recursively includes the architectures built by these procedures. Therefore the entire architecture can form a single node of an identical (or different) architecture, and this can be repeated as often as desired.

Thus multi-computers of potentially enormous size can be designed.

Reconfiguring shuffle networks can also be used, at any or all levels, to give greater flexibility and fault tolerance.

The potential maximum size of networks is far larger than anything feasible today. But this means that network architects have a variety of relatively good building-blocks and design tools at their disposal. The design process can use whatever mixtures of special-purpose, specialized and traditional computer resources are deemed most desirable. And the designer can work with a number of relatively good tools that appear to offer a great deal of flexibility, power and plasticity.

This paper might best be viewed as an exploratory first step toward developing a set of touchstone example architectures (some built, some designed, some pointed to by procedures of the sort proposed here). Suggestions are made as to how the design techniques described might be used to improve existing architectures, and to construct better architectures.

The details of the design of the separate modules, the ways they are linked together, and the total architecture will also be crucial. Indeed frequently an actual architecture will combine several structures in a way especially tailored to give the best design in terms of technology, tasks, and costs. For example, an array might be linked to a conventional serial computer that serves as its host, a pyramid to an MIMD network, or systolic processors to a ring.

The construction, testing, evaluation, and comparison of different architecture topologies as they execute programs of importance, and appropriate mixes of such programs, should in their turn cast light on which are the key abstract criteria. Thus an improved set of criteria and design techniques should, over time, be achieved.

13. ACKNOWLEDGEMENTS

The research on which this paper is based was partially supported by NSF Grant No. DCR-8302397.

14. REFERENCES

Batcher, K. E., STARAN/RADCAP hardware architecture, *Proc. 1973 Sagamore Computer Conf. on Parallel Processing*, 1973, 147-152.

Batcher, K. E., STARAN parallel processor system hardware, *Proc. AFIPS NCC*, 1974, *43*, 405-410.

Batcher, K. E., Design of a massively parallel processor, *IEEE Trans. Computers*, 1980, *29*, 836-840.

Bermond, J-C., Delorme, C. and Quisquater, J.-J., Grands graphes non diriges de degre et diameter fixes, *Technical Report*, Philips Research Laboratory, Brussels, Belgium, 1982.

Bondy, J. A. and Murty, U. S. R., *Graph Theory with Applications*, New York: Elsevier, 1976.

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.

De Bruijn, D. G., A combinatorial problem, *Koninklijke Nederlandsche Academie van wetenschappen et Amsterdam, Proc. Section of Sciences 49*, 1946, *7*, 758-764.

Despain, A. M. and Patterson, D. A., X-tree: a tree structured multi-processor computer architecture, *Proc. Fifth Ann. Symp. on Computer Arch.*, 1978, 144-151.

DeWitt, D. J., Finkel, R. and Solomon, M., The CRYSTAL multicomputer: design and implementation experience, *Comp. Sci. Dept. Tech. Rept.*, Univ. of Wisconsin, 1984.

Duff, M. J. B., CLIP4: a large scale integrated circuit array parallel processor, *Proc. IJCPR-3*, 1976, *4*, 728-733.

Duff, M. J. B., Review of the CLIP image processing system, *Proc. AFIPS NCC*, 1978, 1055-1060.

Dyer, C. R., A quadtree machine for parallel image processing, *Information Engin. Dept. Tech. Rept. KSL 51*, University of Illinois at Chicago Circle, 1981.

Dyer, C. R., Pyramid algorithms and machines. In: *Multicomputers and Image Processing* (K. Preston and L. Uhr, Eds.). New York: Academic Press, 1982, 409-420.

Farber, D. J. and Larson, K. C., The system architecture of the distributed computer system - the communications system, *Symposium on Computer Networks*, Polytechnic Institute of Brooklyn, April, 1972.

Goodman, J. R. and Sequin, C. H., Hypertree, a multiprocessor interconnection topology, *Comp. Sci. Dept. Tech. Rept. 427*, Univ. of Wisconsin, 1981.

Gottlieb, A., Grishman, R., Kruskal, C. P., McAaliffe, K. P., Randolph, L. and Snir, M., The NYU ultracomputer - designing an MIMD shared memory parallel computer, *IEEE Trans. Computers*, 1983, *32*, 175-189.

Hwang, K. and Briggs, F. A., *Computer Architecture and Parallel Processing*, New York: McGraw-Hill, 1984.

Imase, M. and Itoh, M., Design to minimize diameter on building-block network, *IEEE Trans. Computers*, 1981, *30*, 439-442.

Kung, H. T. and Song, S. W., A systolic 2-d convolution chip. In: *Multi-Computer Algorithms for Image Processing*, K. Preston, Jr. and L. Uhr (Eds.), New York: Academic Press, 1982.

Kung, H. T. and Lam, M. S., Wafer-scale integration and two-level pipelined implementations of systolic arrays, *J. Parallel and Distributed Computing*, 1984, *1*, 32-63.

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

Preparata, F. P. and Vuillemin, J., The cube-connected-cycles: a versatile network for parallel computation, *Proc. Twentieth Ann. Symp. on Foundations of Computer Sci.*, 1979, 140-147.

Reddaway, S. F., Revolutionary array processors. In: *Electronics to Microelectronics*, W. A. Kaiser and W. E. Proebster (Eds.), Amsterdam: North-Holland, 1980, 730-734.

Rieger, C., Bane, J. and Trigg, R., A highly parallel multiprocessor, *Proc. IEEE Workshop on Picture Data Description and Management*, 1980, 298-304.

Schaefer, D. H., A pyramid of MPP processing elements - experiences and plans, *Proc. 18th Int. Conf. on System Sciences*, Honolulu, 1985.

Siegel, H. J., et al., PASM: A partitionable multimicrocomputer SIMD/MIMD system for image processing and pattern recognition, *TR-EE 79-40*, School of Elect. Engin., Purdue Univ., West Lafayette, 1979.

Siegel, H. J., PASM: a reconfigurable multimicrocomputer system for image processing. In: *Languages and Architectures for Image Processing*, M. J. B. Duff and S. Levialdi (Eds.), London: Academic Press, 1981.

Snyder, L., Introduction to the configurable highly parallel computer, *IEEE Computer*, 1982, Jan., 47-64.

Stone, H. S. (Ed.), *Introduction to Computer Architecture*, Chicago: SRA, 1980.

Storwick, R. M., Improved construction for *(d,k)* graphs, *IEEE Trans. Computers (Short Notes)*, 1970, *19*, 1214-1216.

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

Tanimoto, S. L., Towards hierarchical cellular logic: design considerations for pyramid machines, *Computer Science Dept. Tech. Rept. 81-02-01*, Univ. of Washington, 1981.

Tanimoto, S. L., A pyramidal approach to parallel processing, *Proc. 10th Annual Int. Symposium on Computer Architecture*, Stockholm, 1983, 372-378.

Uhr, L., Converging pyramids of arrays, *Proc. Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management*, IEEE Computer Society Press, 1981, 31-34.

Uhr, L., Pyramid Multi-Computer Structures, and Augmented Pyramids, In: *Computing Structures for Image Processing*, M. Duff (Ed.), London: Academic Press, 1983, 95-112.

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

Uhr, L., Pyramid Multi-Computers, and Extensions and Augmentations, In: *Algorithmically Specialized Computer Organizations*, D. Gannon, H. J. Siegel, L. Siegel and L. Snyder (Eds.), New York: Academic Press, 1985, in press.

Widdoes, L. C., The S-1 project: developing high-speed digital computers, *Proc. Compcon 80*, 1980, 282-291.

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

Wulf, W. A. and Bell, C. G., C.mmp - a multi-mini processor, *Proc. AFIPS NCC*, 1972, *41 part II*, 765-777.