A CHARACTERIZATION OF PARALLEL SYSTEMS

By

A. L. Davis

W. M. Denny

and

I. Sutherland

August 1980

COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF UTAH

A CHARACTERIZATION OF PARALLEL SYSTEMS

By

A. L. Davis

W. M. Denny

and

I. Sutherland

UUCS-80-108

August 1980

ABSTRACT

A taxonomy for parallel processing systems is presented which has some advantages over previous taxonomies. The taxonomy characterizes parallel processing systems using four parameters: topology, communication, granularity, and operation. These parameters are used repetitively in a hierarchical fashion to produce a taxonomic structure which is extensible to the level of detail desired. Topology describes the structure of the principle interconnections. Communication describes the flow of data and programs through the system. Granularity describes the size of the largest repeated element, or grain. Operation describes the important functional properties of each grain, especially the ratio of storage to logic circuitry. Granularity and topology are structural parameters, while operation and communication are functional parameters which describe the behavior of the system components. A final section of this paper includes examples of the application of the taxonomy to several parallel processing systems.

1. INTRODUCTION

Modern VLSI technology has created the opportunity to fabricate information processing systems containing many replicated subunits. In many cases these subunits might be called processors, even though they are often much simpler than traditional general purpose processors. The result is that a new generation of parallel processing systems is coming into use. These systems range from small collections of microprocessors to massive collections of high speed processing engines. Currently it is easier to produce multiprocessor systems than to use them. One possible aid to their use would be a classification scheme having enough validity to clarify the important distinctions among parallel processing systems and thereby help to indicate how they could be used. In this paper we present such a taxonomy for systems whose processing power is derived from the replication of functional subunits.

It is unlikely that any taxonomy for machines will be as useful or as valid as taxonomies for the plant and animal kingdoms. This is due primarily to the dynamic nature of machine systems. Life forms change, but not significantly over many thousands of years. Machine systems are relatively recent, considerably more dynamic in nature, and a new one is created every time sufficient imagination and financial resources are available. Another problem for computer taxonomists is that many of a system's characteristics are due not to circuit properties, but to program properties. Software exists at every level in a system from the high level programs that are the "operating system," through the lower level programs termed "firmware," to very low level programs which specify state tables for state machines that are typically held in ROM.

This wide variety of programs in a machine system must be considered a major influence on its functional behavior. Because

these programs dictate how the overall system structure can be used, any taxonomy based solely on a machine's physical structure is unlikely to be very useful. In highly integrated designs, firmware level programs are often combined with hardware to produce what the manufacturer calls "hardware". Changing the firmware could thus result in changed "hardware". It is conceptually possible to represent this "hardware" through a purely structural representation, which logically corresponds to a memory dump of the appropriate ROMs. A more useful analysis would include functional aspects of the system. For example, rather than examine the ROM contents directly, one could do so in terms of a higher level language. A major contribution of such languages is their ability to express a program's functional behavior in an intellectually manageable way [25]. The taxonomy presented here integrates both structural and functional system properties in an attempt to more completely characterize parallel systems.

It can be argued that hardware is not always a simple expression of software. If the software does not fit the hardware, gross performance degradation results. While this observation is valid, it is also true that software inefficiencies are often masked by high speed hardware. This fact alone makes any relationship between true computer system structure and performance very complex. This blurring causes any taxonomy to be at best only a guide to predicting actual system performance.

A number of taxonomies for parallel processor systems have already been proposed. Flynn's [11] is the most widely used. Flynn based his scheme on the distinction between single versus multiple data paths and single versus multiple instruction paths in a computer. The resultant SISD, SIMD, MISD, and MIMD classes have been used extensively in the literature. While this scheme is quite useful in

distinguishing many systems at the highest level, it avoids more detailed classification. As a result, it is not useful except for viewing systems in very general terms. For example, Flynn's taxonomy cannot classify the CRAY-1's high level of internal parallelism, nor can it describe the considerable parallel processing performed in large associative memories.

A taxonomy by Hobbs and Theis [14] classifies systems by the amount of parallelism in their control units, processing units, and data streams. It also includes distinctions such as network processors, array processors, functional machines, associative processors, and multiprocessors. As in the Flynn scheme, this taxonomy fails to distinguish between the granularity of large and small processors. It also has a very traditional view of processor and memory that limits its use to systems whose elements closely resemble traditional memories and processors.

A taxonomy by Higbie [13] expands on the Flynn SIMD category and distinguishes between location and value oriented addressing. Murtha and Beadles [20] concentrate on highly parallel MIMD processors, and distinguishes classes of machines on the basis of distributed versus centralized control and whether the processors are special or general purpose. Both of these taxonomies are useful for only a very few machine systems.

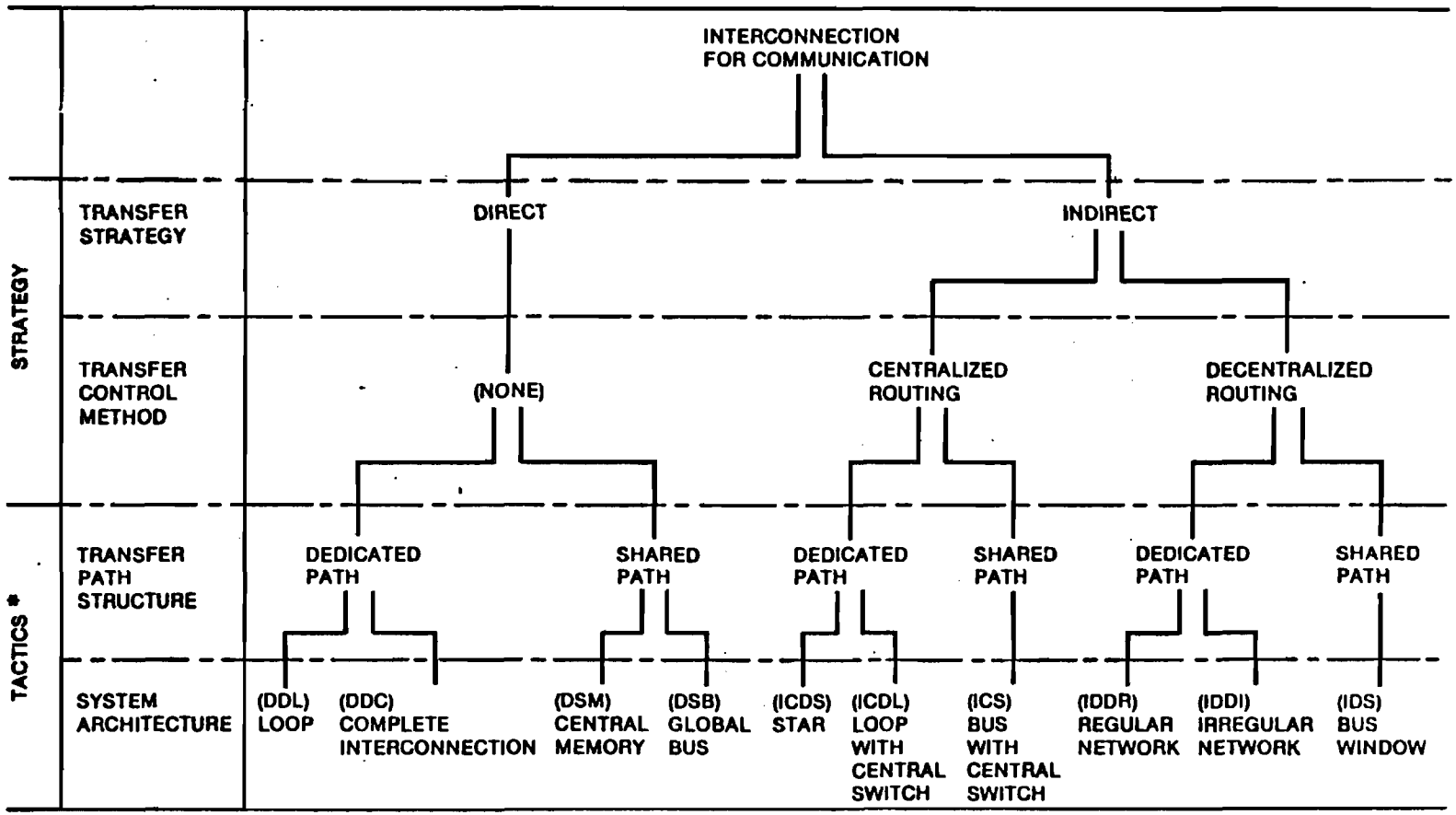
A more quantitative approach was taken by Handler [12]. His scheme is based on a triple [k, d, w] where k is the number of control units, d is the number of ALUs, and w is the logic circuit complexity. The w measure is a good one in that it is a start toward differentiating between small or weak processors and large or powerful processors. Classifications based on merely counting the number of "units" in a machine rarely aid in the understanding of how those units are used. In this respect Handler's ideas are also limited.

All of the above mentioned taxonomies have a major common failing in that they result in a partitioned classification scheme rather than a hierarchical one. All life form taxonomies are hierarchical; while this does not necessarily imply that machine taxonomies should be hierarchical, such an approach provides a good example. Hierarchical structuring is an excellent tactic for organizing any highly complex set. A hierarchical machine taxonomy may be naturally extended to as many levels as necessary to obtain the desired level of detail. Furthermore, a hierarchical description is easily and naturally extensible as additional machine systems are created.

An excellent hierarchical taxonomy has been presented by Anderson and Jensen [2]. The smallest element in their system is a processing element (PE). Any PE is capable of executing a process, where "process" is not exactly defined but taken to mean some reasonably sized piece of code. Other areas of concern in this taxonomy are message paths and switches. Anderson and Jensen limited the scope of their taxonomy to systems where any PE could communicate with any other PE, and chose not to distinguish between packet and circuit switched communication. This is probably useful since such distinctions at the PE level only serve to cloud more important taxonomic issues.

The Anderson and Jensen hierarchy has four levels: The first two are concerned with the strategic issues of how to do message transfers, and the second two are tactical implementation issues. The result is the taxonomy tree shown in Figure 1-1.

As this taxonomy is clearly dominated by interconnect issues and not by processor size, it is incapable of distinguishing a 4 x 4 array of 8080s from a similar array of CRAY-1s. The result is that it gives useful insight into the inter-PE message activity, but not the intra-PE processing activity. The taxonomy would also have some



*AN INTERCONNECTED COMPUTER SYSTEM IS THE RESULT OF A SERIES OF DESIGN DECISIONS, AND THE DECISION SPACE CAN BE CONSIDERED TO BE A TREE.

Figure 1-1: The Anderson and Jensen Taxonomy

difficulty in classifying structures such as Logic-Enhanced Memories [10] and systolic arrays [16] in a useful manner.

A taxonomy should strive to provide a framework for dividing systems of interest into equivalence classes. The vocabulary provided by the taxonomy should not only describe the various member systems, but also provide a basis for analyzing and comparing them. A good taxonomy should therefore select criteria for analysis which are rich enough to include many members of each class, and clear enough to be easily applied to the systems in question.

This paper proposes a taxonomy which will hopefully be more productive than previous efforts in studying and analyzing parallel processing systems. It builds on some of the ideas proposed by Anderson and Jensen, but suggests additional analysis criteria that should extend their taxonomy's utility. Whereas Anderson and Jensen propose a hierarchical taxonomy which uses processing element interconnect communication to define and name generic system types in the hierarchy, our purpose here is to provide a useful vocabulary which supports meaningful architectural discussion. While our vocabulary contains hierarchical elements, we feel that it is premature to use it to name generic system classes. We therefore leave this to future contributions. Furthermore, the Anderson and Jensen taxonomy bases its hierarchy solely on the parameters we call "communication" and "topology". The taxonomy presented here complements this approach, but bases its hierarchy on the granularity or size of the processing elements. In particular, we hope that this taxonomy will be useful in analyzing entire computing systems as well as small structures of wires and silicon gates. The approach is to provide a hierarchical method for analyzing both the functional and structural aspects of parallel computing systems. These features, we believe, determine the problems for which certain classes of digital

systems are well suited.

2. THE TAXONOMY

The proposed taxonomy is aimed at systems which derive their power from replications of the functional elements composing them. It is based on the premise that all parallel processing systems can be considered to be collections of two atomic entities: paths and elements. Paths are used primarily as carriers over which information is transmitted. Paths are considered passive in that they do not modify the transmitted information and, when viewed over a sufficiently long time window, cannot be said to store it. Elements perform some action on information such as storage or functional modification, e.g. add, multiply, execute program, etc. In any given system the division into paths and elements is a bit idiosyncratic, and reflects the background and interests of the person analyzing the system. This is a disadvantage only if one applies a taxonomy rigidly and without much common sense. Sensibly applied, the division of a system into paths and elements can be a powerful tool to clarify the designer's motives and aspects of the system's modularity.

A fundamental feature of parallel systems is that at least some of their elements and paths are replicated to produce physically distinct processing structures which can be active in parallel. For each type of atomic entity, we provide a pair of taxonomic criteria. One is a functional parameter; the other is the associated structural parameter. Figure 2-1 shows the four parameters. Column headings indicate whether the parameter applies to a path or an element. Row headings indicate whether the parameter describes a functional or a structural aspect of the system.

We concur with Anderson and Jensen in the belief that the topology parameter is the most important of the four. The path interconnection topology usually has the most dominant effect on parallel system behavior. Due to their limited connection capability,

	PATH	ELEMENT
STRUCTURAL	TOPOLOGY	GRANULARITY
FUNCTIONAL	COMMUNICATION	OPERATION

Figure 2-1: Four Primary Taxonomic Parameters

this is especially true of highly integrated systems. H. T. Kung [17] has done some excellent work which demonstrates how system efficiency depends on the fit between algorithm and machine topology. Existing systems have a large number of path topologies. The emergence of a new topology is limited only by imagination. A system's topology dictates how many ports each of its elements must have. Elements with many ports are typically more complex than those with fewer ports. Topology also has a major influence on whether or not a system can be implemented as a set of densely integrated VLSI chips. Planer topologies map nicely onto VLSI components whereas an irregular three dimensional connection scheme is likely to fit poorly onto a VLSI chip. In addition, the pin count of a subsystem or module is heavily dependent on the system's topology.

Many functional aspects of paths are of great interest. We use communication to describe the major behavior of system paths, concentrating on three important components of communication:

1. Mobility, the ratio of program to data information carried by the path.
2. ATR, the average transmission rate over the path.

3. Bandwidth, the maximum transmission rate supported by the path.

Precise numerical values are typically less interesting than relative terms in forming the following descriptive cross product.

	high bandwidth	heavy mobility	high ATR
Communication =	medium bandwidth x	moderate mobility x	medium ATR
	low bandwidth	slight mobility	low ATR

This cross product allows a rough characterization of the behavior of the communication paths in the system and provides some insight concerning the individual grains composing the system.

Granularity, or grain size, describes the size of the largest repeated element; as such it is a structural property used as a principal determinant of the hierarchical level of a system. Providing an exact numerical measure for grain size would probably be more difficult than useful. For simple descriptions, terms such as small, medium, and large could be employed. For example, in comparing the ILLIAC IV [4] and the CM* [8] at a level where the elements are processors, the ILLIAC IV's element granularity would be large and the Cm*'s medium. The grain size of an associative memory would be quite small.

The operation of an element describes its behavior in the system; the foremost is its functional transformation of input data to output. An important subsequent distinction which should be made among elements is whether they are primarily processing or storage elements. We call such a distinction the memory-logic mix of the element. Memory-logic mix describes the ratio of memory to processing in the repeated elements. The mix of an element in which a few registers and a fast multiplier are used is clearly more processor intensive than the mix of an element which is a million byte RAM store. Some other

computer taxonomies can be seen as attempts to describe the functions of the processing elements in additional detail. Items such as the element instruction set, if any, and the specific access properties of the memory, such as random, block sequential, or sequential provide additional detail.

To briefly illustrate the use of our taxonomy, consider the ILLIAC IV multiprocessor system. In accordance with our taxonomy, the ILLIAC IV's topology is that of a two dimensional, fully populated, square array of data paths. We will ignore control paths for the purpose of this brief example. The communication along the paths of the square array is supported by high bandwidth, low average transmission rate, heavily data intensive paths. The ILLIAC IV's grain size is medium by present standards although 10^4 gates was large in 1969, and each processor is a logic intensive, programmable arithmetic processor. Each grain has a simple, random accessed memory that lacks the index or base registers needed to support sophisticated access patterns individually, since all the processors share one common index register.

The following sections describe the principle features of the taxonomy in more detail.

2.1 TOPOLOGY

In any system which derives its power by replicating some set of physical resources, the constraints on communications that are imposed on these multiple resources must be considered a primary factor in its analysis. By maintaining the distinction that multi-resource systems are composed primarily of elements and paths, we can define communication as that aspect of system behavior which is concerned with what takes place on the system paths. We also choose to maintain the distinction between the functional and structural parts of communication, and call the structural aspect <u>topology</u>. If some

element A has a path to element B, then we say that elements A and B communicate directly. It is clear that "communicates with" is a transitive relation, but "communicates directly with" is not transitive. If any two elements C and D can communicate only with messages routed through one or more intermediate elements, then C and D are said to communicate indirectly with each other. Kung [17] has presented a rather convincing view that very high performance can be obtained if algorithms can take advantage of the direct communication possibilities of the hardware. We use the term topology to describe the direct communication structure of the system.

Topology can best be understood using a graphical representation. Any system S consisting of elements and paths can be mapped onto a topology graph (TG) consisting of vertices and connecting arcs. For each element of S there is a corresponding vertex in the TG, and for every path in S there is a corresponding arc in the TG. The TG represents the communication paths in S, because for any pair of elements in S which communicate directly with each other over a path, there exists an arc in the TG which connects the vertices in the TG corresponding to those elements.

Using a system's topology graph we can now study its direct communication structure by analyzing its TG. In some systems certain communication paths are one way paths, while in other systems all paths are bidirectional. In systems where one way paths exist, the concept of TG arcs may be extended to include directed arcs. For example, if E_c can send a message to E_d, but not vice versa, then there is a directed arc in TG which goes from V_c to V_d. Note that the existence of two complementary directed arcs in TG is not equivalent to a single non-directed arc as illustrated in Figure 2-2.

In both graphs of Figure 2-2 it is apparent that V1 and V2 communicate directly with each other, but that the relation

Figure 2-2: Two Different Topologies

"communicates directly with" does not contain enough information to identify senders and receivers. Such identification is often important and can be represented in topology graph form. In Figure 2-2a, the system designer has provided two distinct paths having predefined senders and receivers; that is, each path has a dedicated function. In Figure 2-2b, the path can be shared by two possible senders and two possible receivers, and is therefore a shared path.

The number of ports of a system element is often a clue to its complexity. While this information is not directly associated with system topology, port information can be obtained from the TG. In Figure 2-2a both system elements are dual-ported; in 2-2b both elements have only a single bidirectional port.

The distinction between shared paths and dedicated paths is important because communication contention for shared paths may cause a form of sequencing in concurrent systems. Anderson and Jensen [2]also chose to consider this distinction in their taxonomy. The main difference proposed here is that dedicated and shared path information can be deduced from a system's topology graph. Furthermore, much of the information contained in the other levels of

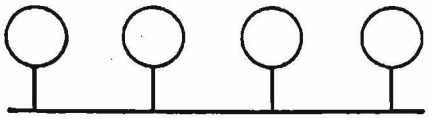
the Anderson and Jensen taxonomy is described by what we call topology. As such, we assert that the Anderson and Jensen criteria are subclasses of the topology considerations presented here.

One particular type of shared path is a bus. Bus organized systems are widespread and present a rather attractive communication medium for replicated systems in which only one message at a time need be transmitted between member elements of the bus connected subsystem. The TG representation of a bus connected subsystem varies with the type of bus control mechanism. Figure 2-3 shows several possibilities.

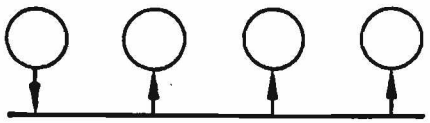
Figure 2-3a shows a number of single ported elements, all connected by a single path. This type of bus system requires that the sender's identity be determined by sensing the state of the bus. Such busses can be implemented using a number of techniques such as Tri-state, wired-or, etc. All possible senders on such a bus may try to drive it with some ID value. After looking at the resultant bus levels, the actual sender's ID can be known. Receiver ID can be known either by message tags or all non-senders can be defined to be receivers in a broadcast like mode. Figure 2-3b shows a simpler configuration where there is a single sender and up to three receivers. The receivers in this scheme can also be selected by message tag or in a broadcast mode. Figure 2-3c is an example of a two bus system where element A may transmit to element B, C, and/or D on one bus and receive replies on the second.

Often the n wires of a bus may be partitioned into a number of sub-busses, where each sub-bus has a special function. An example is the UNIBUS [6], which contains 56 active wires. Only 16 of these wires are used to carry data. The remainder are used in control functions. One useful way to initially partition the UNIBUS is illustrated in Figure 2-3d, where the top bus consists of the UNIBUS

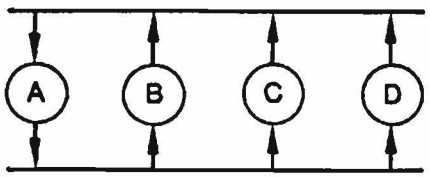
a) Bus state control



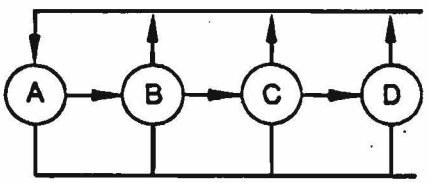
b) Single Sender Multiple Receiver



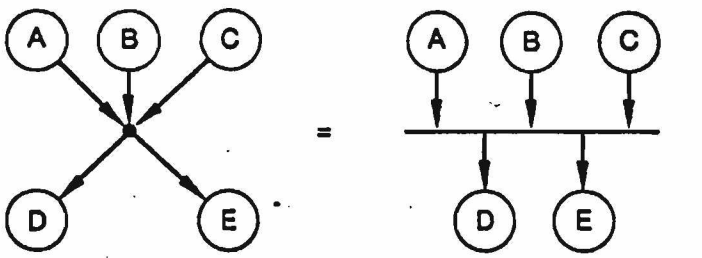
c) Two-bus subsystem



d) UNIBUS-like system



e) Unidirection ports: two equivalent TG structures



f) Bidirectional ports: two equivalent TG structures

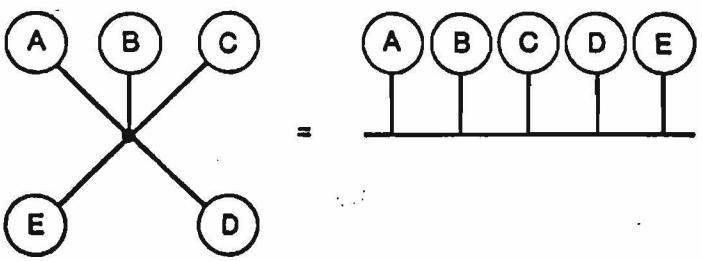


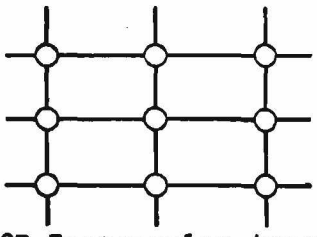
Figure 2-3: Topology Graph Examples for Bus Connected Subsystems

bus request lines. The middle set of lines, which begin at A and are passed successively to B, C, and D, are the UNIBUS bus grant lines. The bottom, bidirectional bus in this TG corresponds to the remaining wires of the UNIBUS.

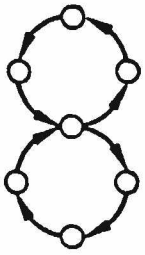
There are often a number of TG shapes that describe the same system topology. This difference may result from the type of analysis which is to be done, or it may merely reflect a personal preference for a particular graphical style. Figures 2-3e and 2-3f show two bus topologies, where each topology is represented in two ways.

Many system topologies have a large number of arcs, each of which connect two vertices of the corresponding TG. In these topologies, each arc represents a dedicated system path. Figures 2-4 and 2-5 show a number of important topologies.

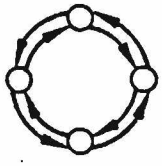
One of the most important topological considerations is the number of dimensions required to represent the TG while avoiding crossing paths. This is important because, if no crossing paths exist, logical adjacency (direct communication) in the TG can easily be exploited as physical adjacency (short wires) in an implementation. If logical and physical adjacency differ, second order communication delay effects may influence the system's behavior. These effects result from long-wire transmissions, and are not readily apparent by simple analysis of the direct communication structure of the system. For example, it is well known that one and two dimensional (1D and 2D) topologies present particularly nice communication structures for VLSI implementations, because on-chip communication is basically carried on in a small number of planer layers. Certain 3D communication structures can still correspond to physically implementable adjacencies for VLSI systems, whereas structures of four or more dimensions will necessarily result in some long and therefore slow wires. In addition to being slow, 4D structures often present



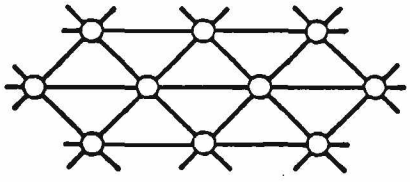
a) 2D Rectangular Array



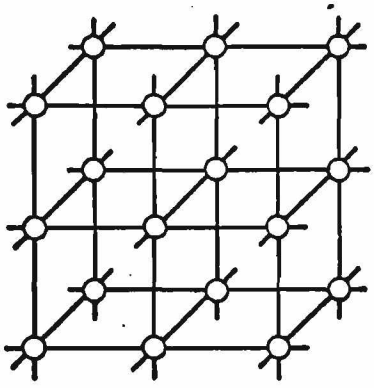
b) 2D Multiply Connected Dual Ring



c) 2D Doubly Connected Ring

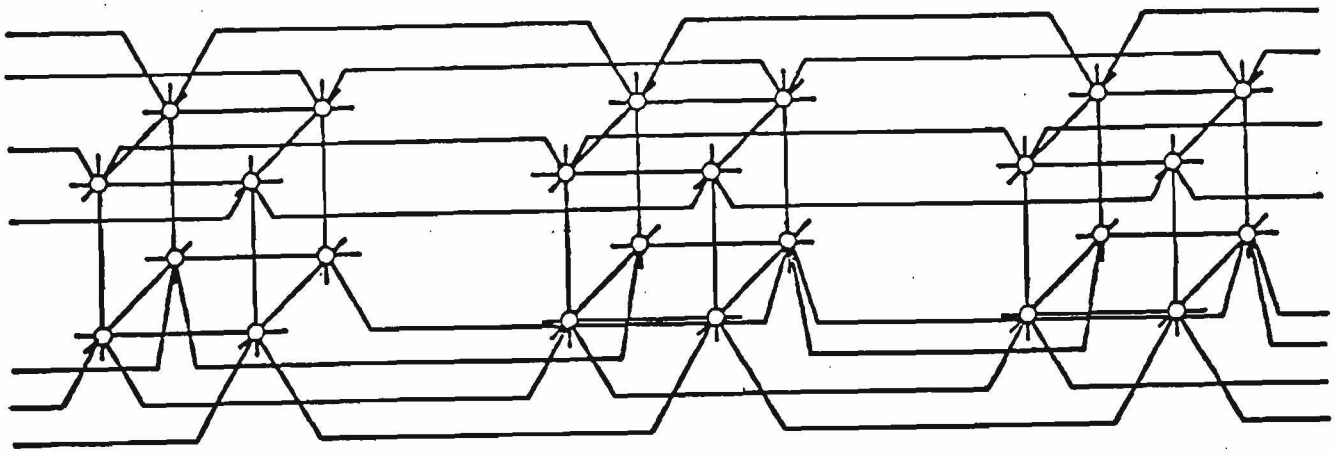


d) 2D Multiply Coupled Triangular Array

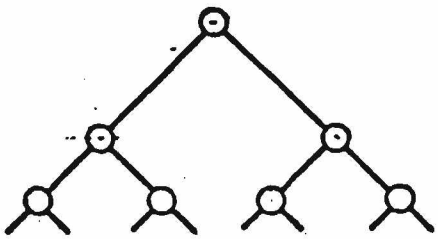


e) 3D Cubic Array

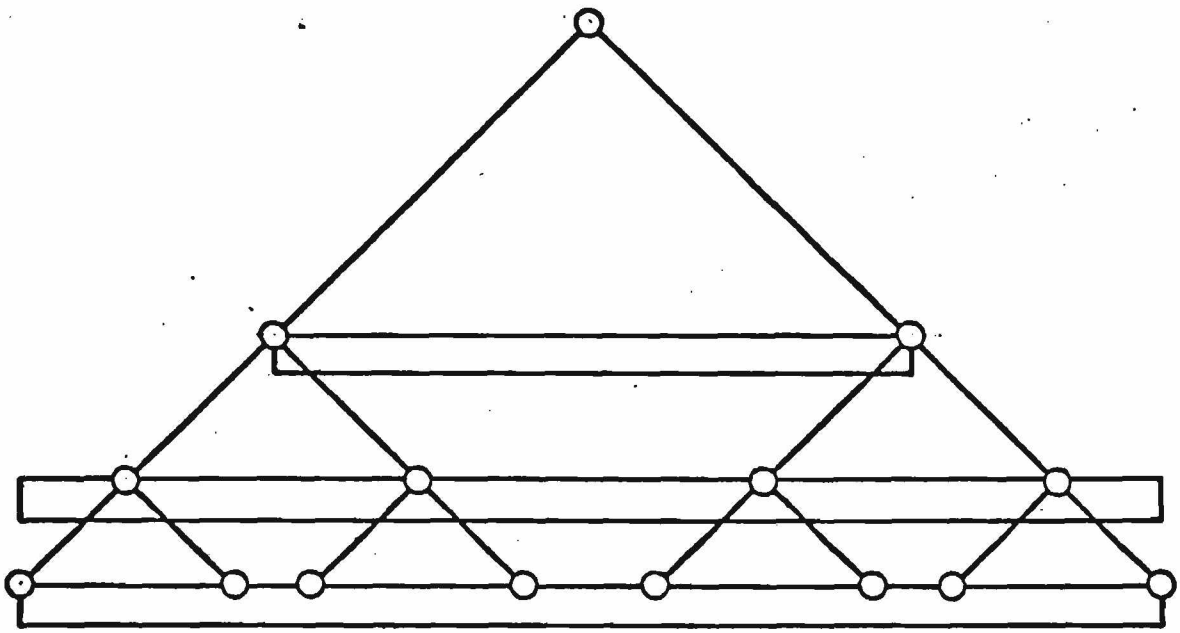
Figure 2-4: Sample Topologies



F) 4D HYPERCUBE ARRAY



G) 2D BINARY TREE



H) 3D GENERATION CONNECTED BINARY TREE (X-TREE)

Figure 2-5: Sample Topologies (Continued)

communication requirements which are extremely, if not impossibly, complex structures to lay out as a two dimensional integrated circuit. Kung [16] has shown that a number of topologies are particularly attractive for implementing certain algorithms and has also related a few such topologies to their suitability for VLSI. A generalization of Kung's techniques may well be the key to future evaluation of architectures using the topological considerations described here.

2.2 COMMUNICATION

We use the term communication to refer to the functional aspects of communication between two system elements over a system path. Communication information answers questions as:

- How fast can information be transmitted?
- What is the average amount of information being passed over the path?
- What type of information is being carried over the path?

Of particular importance in answering the last question is a distinction between the amount of program being carried and the amount of data being passed. We use the term mobility to denote the ratio of program to data being passed over the system path. If most of the information passed over a particular path is used as data by any receiving element, then the system function is relatively static. This does not imply that activity is low, but rather that the distribution of the functions throughout the system is relatively stable. We could therefore say that the functional properties of such a system are less mobile than in the case where mobility of the system path is high.

In systems where the elements do not correspond very well to traditional notions of a processor, it is unwise to retain traditional definitions of a program. For our purposes, we refer to program information as that information which helps to define the function a

system element will perform. Data is then information which affects result values but not element function. If a system path has high mobility i.e., if most of the information which passes over the path is program, then that path plays an important role in the functional modification of system activities. Typically, high mobility systems are more general and have lower throughput than low mobility systems.

A second important aspect of path function is the average transfer rate (ATR) of the path. The path may place a physically imposed upper bound on the ATR. This upper bound is the bandwidth of the path. If the ATR is consistently a small percentage of the bandwidth for a given path, then that path is being used inefficiently. This inefficiency may not hurt overall system performance, but such an observation provides a good starting point for further analysis. For example, low ATR may result from an almost ideal partitioning of system activity onto resource grains which rarely need to communicate.

It is easy enough to provide quantitative values for mobility, ATR, and bandwidth, but qualitative terms will usually provide a more intuitive description. For example, low mobility, high bandwidth, high ATR systems tend to be specialized structures which are very efficient in solving a very restricted class of problems. A logic-enhanced memory structure [10] is an example of such an architecture, which is extremely good at certain correlation problems, but which cannot be easily called upon to do inventory accounting for a hospital. A high mobility, low ATR, medium bandwidth system may indicate an array of microprocessors each working on a resident database using a large number of possible functions. PASM [24]is such a machine. In PASM each microprocessor is responsible for processing a predefined piece of an image database. A medium mobility, medium ATR, medium bandwidth system would represent a rather

efficient, general purpose, multiprocessor system; while a medium mobility, low ATR, high bandwidth system may correspond to a potentially high performance, general purpose multiprocessor.

Using these three subclasses of communication it is possible to classify and analyze a wide variety of digital systems which are beyond the scope of previous taxonomies.

2.3 GRANULARITY

At each level of a hierarchical description, the granularity is the size of that level's largest repeated element. In fact, it is probably the size of the grain more than any other parameter which determines the level at which the system is described. Top level descriptions concern themselves with the largest grains of a particular system, and progress hierarchically down through ever decreasing granularity to the smallest replicated structures. It is therefore important to select a metric for expressing grain size which not only allows comparisons between different processing systems, but also is relevant when comparing different hierarchical levels within one processing system. Simple relative descriptions such as small, medium, or large are intuitively useful in distinguishing microprocessor arrays from systems like PEPE [7] or ILLIAC IV from small grain systems such as Holland machines [15] and systolic arrays.

In precise terms, the gate count of a grain is probably the simplest useful metric to apply. Since a gate typically is the smallest functional unit in a digital system, this metric has an immediate appeal. The number of gates required to implement a given function in a stated time depends to some extent on the implementation technology chosen and the skill of the circuit designer. However, implementation issues such as technology are usually known and design skill is essentially constant when comparing various architectures.

Furthermore, determining the number of gates composing a grain to better than within a factor of 5 or so gives a false sense or precision to what is essentially a simple taxonomy. The temptation to include a host of other factors such as gate count, number of wires, computational capability, power consumption, weight, total area of substrate consumed, cost, etc., and express them in a weighted polynomial should be avoided for the same reason. If a specific application makes one of these additional factors especially relevant, it should be selected as the sole determinant of grain size, or else included in some simple combination with the gate count as the metric. The product of gate count and the number of gate interconnects is perhaps useful in this way.

In another sense, grain is the reciprocal of the number of elements one can reasonably expect to build into a practical system. Large grained systems are probably feasible only when at most a few hundred elements are used. Small grain systems, however, may well consist of thousands to millions of grains. Thus the grain size in part determines whether or not one can profitably spend engineering or computing time allocating tasks to each element. In large grained systems the high cost of each grain probably makes explicit allocation worth the effort. In systems with small inexpensive grains, a more cost effective strategy would be a statistical allocation or a "quick fit" policy that tacitly accepts wasting some of the inexpensive elements.

In this sense grain size also directly influences inter- and intra-grain control issues. Synchronous control of more than a thousand processing elements is a monumental engineering task [23]. Thus small grained systems can be expected to utilize more sophisticated inter-grain control schemes than large grained systems. On the other hand, small grained systems can be expected to require

simpler internal control than their larger cousins. It is because of this relationship that such structures as dataflow machines [1] and systolic arrays raise more complicated control issues than the synchronous machines of the past. As the fabrication technology achieves ever smaller feature sizes, these grain size induced control issues strongly influence the design of the chips themselves. It has been pointed out that self-timed logic is a virtual necessity in VLSI chips when the number of transistors reaches the order of a million [23].

Finally, grain size exerts an influence on the "computational generality" of the entire processing system (see Figure 2-6).

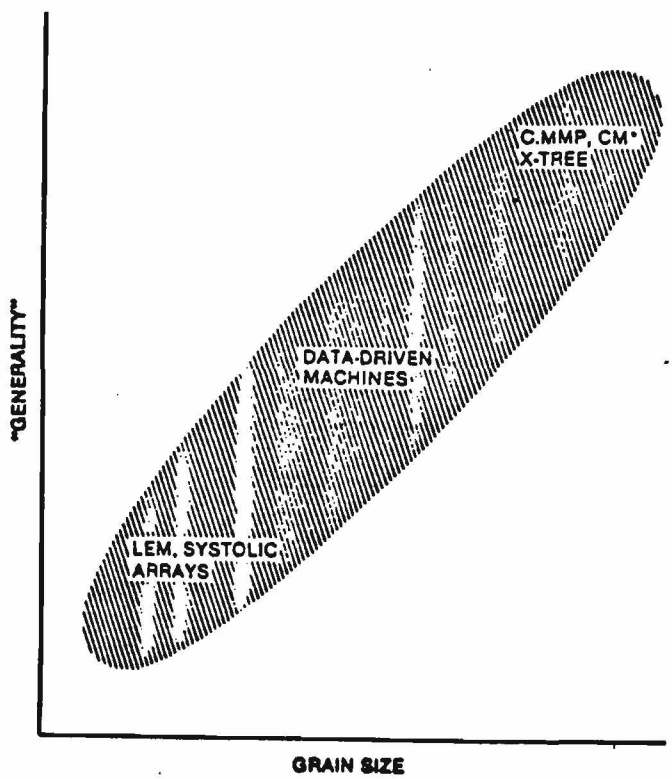


Figure 2-6: The Influence of Grain Size on Processing Generality

Large grained systems have so far been composed of programmed von

Neumann computers capable of ever increasing feats of general purpose processing within their throughput limitations. Small grained computing systems are relatively new, pose difficult control issues, and thus far have been hard wired for rather specific classes of problems. Moreover, the last 30 years of algorithmic research have focused most strongly on small numbers of von Neumann architectures, whereas small grained structures require algorithmic mappings onto very large numbers of simpler processing elements [26]. The representation methods for such highly parallel algorithms are as yet poorly developed, and there seems to be little agreement on the most promising approach [21]. Because of these historical limitations on small grained systems, it is as yet unclear whether the trend noted in Figure 2-6 represents a fundamental property of these systems or simply a "head start" gained by the earlier development of traditional, large grained processors.

2.4 OPERATION

Once the grain structure is characterized in terms of its total gate count or similar metric, its computational power can be crudely estimated. Operation further characterizes the chosen elements at each level of discussion in more functional terms. The first is a characterization of the gross behavior of each grain as a transformation of inputs to output states. Given this description, often in informal English or, for simpler systems, in terms of more formal state tables. The viewpoint of traditional combinational logic is that the first distinction which can be made on a collection of gates is to distinguish between storage elements and sequential logic elements. Storage elements use feedback or physical properties of the medium to guarantee stable states necessary for the memory function. The introduction of storage into such systems allows a significant reduction in the amount and complexity of sequential logic since intermediate calculations can be saved for future use rather than

explicitly calculated with combinational logic circuits. Conversely, sequential logic can be used to reduce storage requirements when it is cheaper to calculate values rather than to use massive amounts of storage to maintain the values as tables. There is thus a fundamental trade off between amounts of memory and logic to implement a given functional task.

A simple measure of memory/logic mix is the number of gates devoted to logic compared to the number of gates in the memory. Such a measure gives a crude measure of mix, but does not distinguish well between machines of varying word sizes. A more meaningful measure is the number of storage cells which can be accessed by the logic. This is relatively easy to determine for systems where a fixed word size exists. In systems which use variable length storage fields, the measure is more difficult but can be obtained statistically. Memory rich elements are generally limited in the complexity of the functional transformations they perform. However, their regular and simple internal topologies and their high circuit densities suit them for VLSI implementation. Logic rich elements, on the other hand, are generally able to implement more complex transformations, but possess lower circuit densities and a more complex internal topology.

The capacity of a memory is also determined in part by how effectively the various data types permitted by the organization are mapped onto the storage cells. Since most modern computers attempt to support several data types with a few basic cell sizes, the evaluation of the true capacity of sophisticated storage systems can be rather difficult. However, in many parallel systems the granularity is small enough to forbid such complexities and permit a realistic evaluation of the system capacity.

The logic function, too, can be only crudely characterized by a gate count (see Figure 2-7). This is because large numbers of gates

can be used to provide either high speed or a diverse functional repertoire. A more complete characterization of an element's operation must therefore include the "style" in which the element is implemented, and the number and kind of functional transformations which can be performed on the data. Where the need for simplicity and low gate count supercedes the need for speed, bit serial logic architectures can be used. However, most computer systems operate on data groups ranging from four to 64 bits in one clock pulse. Occasionally computers have provided sufficient logic to allow this degree of parallelism to be variable [27]. In this case, a characterization of the logic complexity includes a description of the appropriate bounds on the word size.

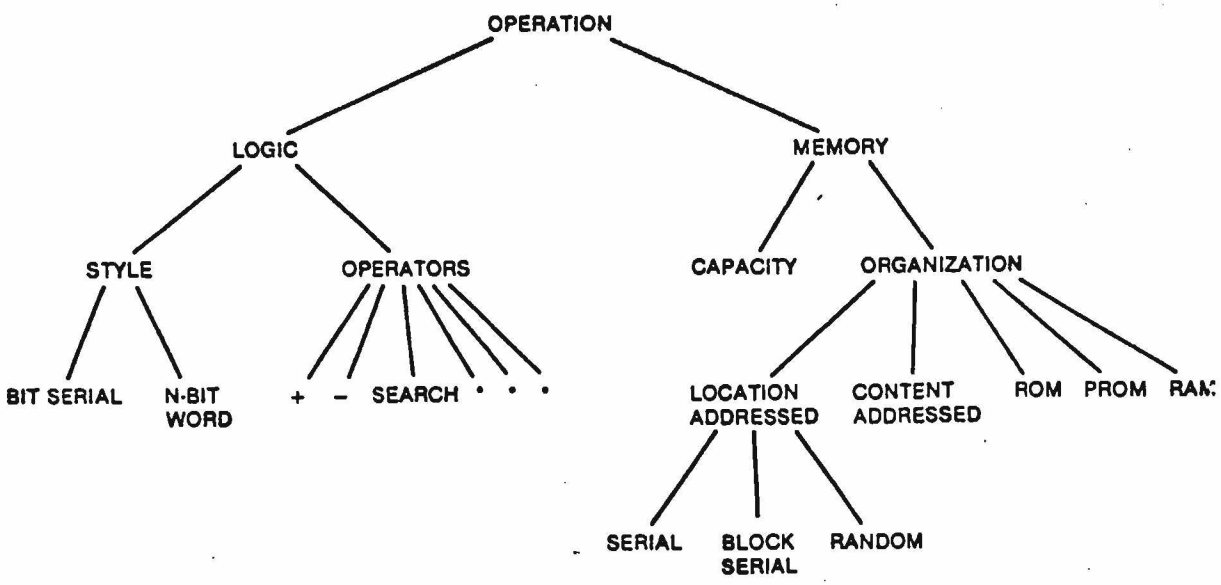


Figure 2-7: Important Aspects of Element Operation

The number of operators supported on the various data types is a measure of logic complexity which usually receives considerable

attention in processor handbooks and in the ISP notation [5]. In large grained elements, it may be necessary to form a cross product of the number of supported operators with the number of data types to form a realistic impression of the functional power of the system's structure. Forming this cross product reasonably requires considerable familiarity with the system and its intended use to eliminate ridiculous combinations of operators and data types. For example, does anyone ever search a double precision floating point number for the occurrence of a given bit sequence?[1] In small grained systems such as systolic arrays, simple recurrence relations can be used to describe the operators of each element.

The organization of the memory influences the functional behavior of the computing element in ways which extend beyond the data types. The ratio of read to write time, and the physical mechanisms needed to write and read storage exert considerable influence on algorithm execution. If some of the memory is ROM, or PROM, the write process is far more complex than if it is implemented in RAM. These more complex processes require resources usually unavailable to the processing element, and this lowers the flexibility of the element below that traditionally associated with RAM. ROMs and PROMs are essentially logic implementation techniques rather than true memory structures.

The accessing organization provided by the memory also exerts a strong influence on the behavior of the element. First the logical address space seen by the operators in a random access memory must be

---

[1] A more detailed analysis would rank the various operator-data type pairs by frequency of use and then multiply each use-weighted pair by its execution time. Studies such as this are common among computer manufacturers and represent a finer level of detail than is usually appropriate in a taxonomic description.

mapped onto the physical cells in some fashion such as one to one, paged, virtual storage, cache memory, etc. Furthermore, search and compare logic is often closely commingled with groups of memory cells if it receives considerable use. Such content addressable memories can themselves form a portion of the elements of a large grained system such as STARAN [22]. In a small grain analysis or STARAN, each 256 word data block and its associated search circuitry forms an even smaller grained element. In the former case, the logic in the large grained element needs to do little searching since the CAM performs this function quite well. In the latter analysis, the logic in the processing element performs only searching and comparing. Logic-enhanced memories [10] are subject to a similar analysis with more complex arithmetic functions replacing the search and compare process of the CAM.

From another point of view, the access patterns supported by an addressable memory exert a strong influence on the behavior of the algorithms executed by the logic. This observation is especially relevant as random access memories are increasingly complemented by CCD and bubble memory technology. These memory organizations permit rapid serial transfers and considerably slower random access. Such block serial devices permit very dense memory structures combined with rapid and simple iterative algorithms implemented in small amounts of logic [10] in cases where data accesses are for the most part sequential.

3. EXAMPLES

3.1 Systolic Arrays

Systolic arrays have been developed by Kung [16] to perform matrix manipulations such as LU decomposition and multiplication with a high degree of parallelism. The systolic array of Leiserson [18] implements a priority queue on a vector of data. While these designs differ, they share many features.

The fundamental communication topology of the systolic arrays developed thus far is illustrated in Figure 3-1. The structure is a fully populated regular array where each node is directly linked via a separate path with its six nearest neighbors. This structure is the fundamental topology of systolic arrays because the four nearest neighbors rectangular array of Figure 3-1b, and the binary tree of Figure 3-1c are subsets of this topology.

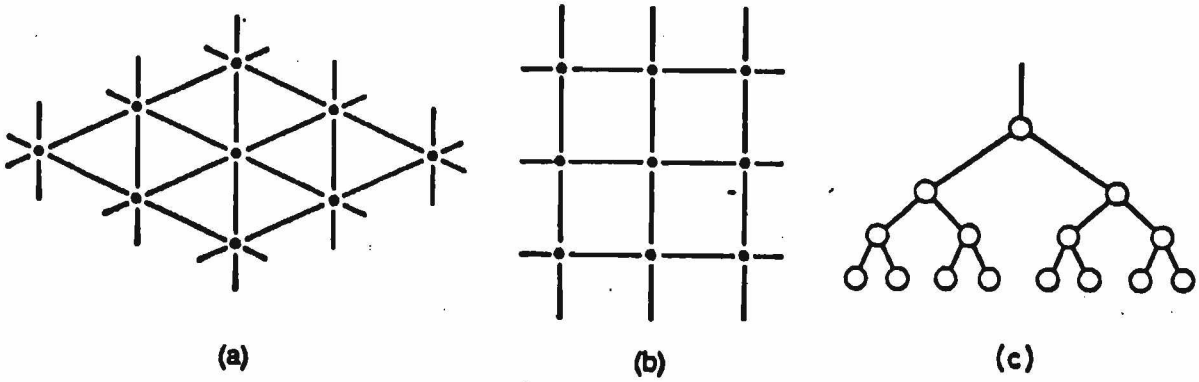


Figure 3-1: The Principal TG of Systolic Arrays (a) and two Subsets (b) and (c)

Analysis of many of the features of systolic arrays is facilitated by examination of the dual graphs of Figure 3-1 shown in Figure 3-2.

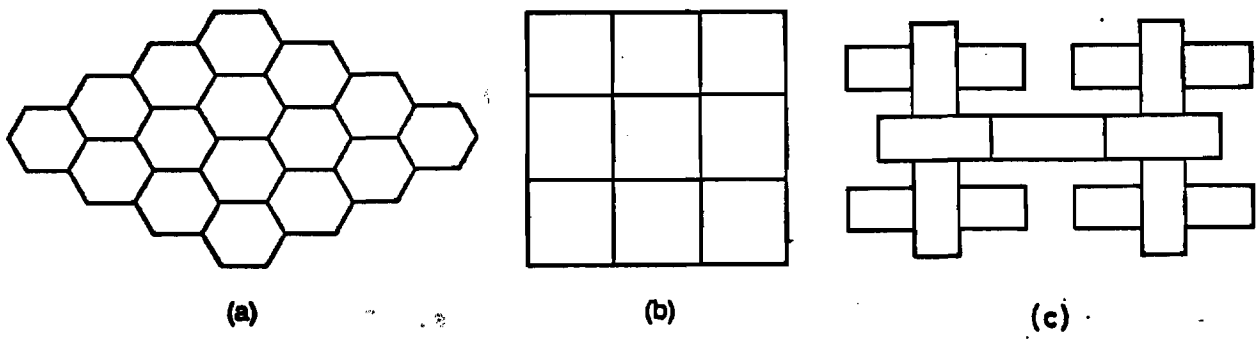


Figure 3-2: The Dual TG of Figure 3-1

We notice immediately that Figure 3-2a is a tessellation of the plane into hexagons. This property implies several advantages in a VLSI design. First, if we imagine the hexagons containing computational elements and communicating at their edges, then the communication paths are inherently non-intersecting and very short. Furthermore, if the granularity of each cell is the same, and if the operation of each cell permits them to have a hexagonal geometry, then the chip plane is well packed with processing elements. This should maximize the cell density of a systolic array and make effective utilization of the planer communication and high device density of VLSI chips.

Secondly, the regular tessellation suggests that each cell could function identically. So far, no systolic arrays have been designed

which have more than one fundamental shape in the dual TG. Interposing multiple shapes in a regular pattern is certainly possible and could present a promising direction for future investigation. Regular tessellation should minimize the design costs of such arrays when implemented in VLSI chips.

Systolic arrays possess low mobility. In fact, since the control signals have the same communication topology as the data, systolic arrays could make use of "self-timed" logic elements, a style which is attractive in VLSI designs [23]. The bandwidth of the communication paths is quite high, since the planer topology permits the cells to communicate at on chip speeds. Moreover, the ATR of each path is generally quite high. This is so because the timing of the communication in a systolic array is regular, with each cell outputting data on each "systol" as it inputs new data. Thus, with proper buffering, the paths are in use most of the time.

The grain size of systolic arrays is very small - generally several hundreds of gates. This has been possible because the functional behavior of each cell is simple and because only a small amount of data storage is necessary at each cell. Small grain sizes have the advantage that they are relatively easy to design compared to a microprocessor. However, they have the disadvantage that they do not permit the flexibility that large grained systems have. Systolic arrays can be expected to be applied to rather limited classes of problems. This observation is consistent with the low mobility of the inter-element communication paths, itself another sign of a more special purpose architecture.

The functional behavior of a systolic array is expressed by recurrence relations which relate the output of element i to the output of element i + 1. Generally, these recurrence relations take the form of inner products such as:

$$C_{i+1} = C_i + A_i B_i$$

The amount of logic needed to perform the computations can be kept small in such a design. Depending on specific problem requirements, the design could manipulate either fixed or floating point numbers and be implemented in either serial or parallel fashion to yield the appropriate processing speeds. Because the granularity and ATR suggest that the systolic array is a special purpose architecture, it would be reasonable to suggest implementing the logic with a ROM or PLA rather than a programmable RAM. However, much work remains to be done on the effect of small changes in the recurrence relations to the overall algorithms allowed by a given topology.

The memory used by each cell is quite small - consisting only of input/output buffers and queues between the cells. This memory is generally written into by only one cell and read from by only one other cell, thus simplifying buffer control logic. Moreover, the data is accessed in a FIFO manner, thus eliminating the need for complex addressing schemes. The number of data types supported by the memory is small because the operators are simple arithmetic ones and few in number. This is consistent with the special purpose nature of the machine.

Logic-enhanced memories [10] are described in much the same way as systolic arrays, except that their granularity tends to be somewhat larger and each grain more memory heavy because of the deliberate attempt to incorporate memory into each element and to keep the amount of logic minimal. The internal topology of each element is made simple and regular by replacing traditional logic with table lookup to perform finite difference calculations, and other techniques which use large amounts of memory and little logic.

3.2 THE CM* MULTIMICROPROCESSOR SYSTEM

The CM* system [8] is an attempt to utilize a microprocessor, the LSI-11, as a fundamental module in a computer architecture and capitalize on the high degrees of parallelism possible using large numbers of such modules. The architecture is chosen as an interesting application of the hierarchical nature of our propsed taxonomy.

At its highest level, the CM* is a collection of bus linked "clusters" (Ks) with selected clusters attached to two busses (see Figure 3-3). Each cluster is itself comprised of up to fourteen "computer modules" (CMs) and one K-map, all sharing a common map bus, (see Figure 3-4). The CM* is thus a "bus of busses" - a not surprising design considering its PDP-11 heritage.

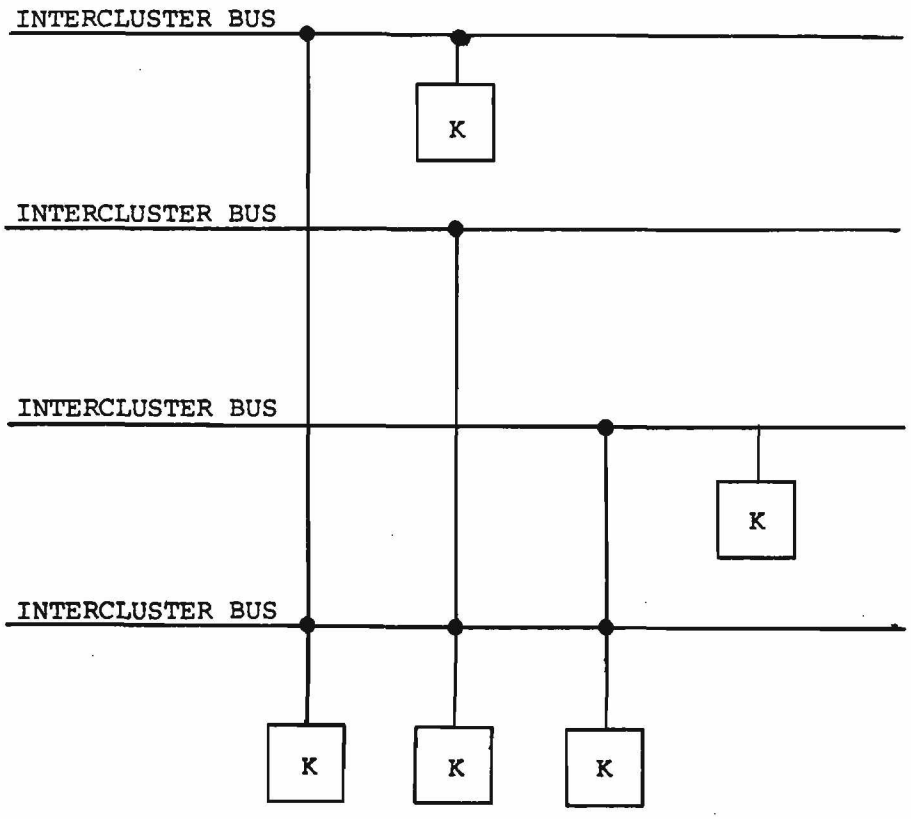


Figure 3-3: The High Level Topology of CM*

The communication profile of the CM* indicates that the principal bus traffic consists of data words. Program information is also

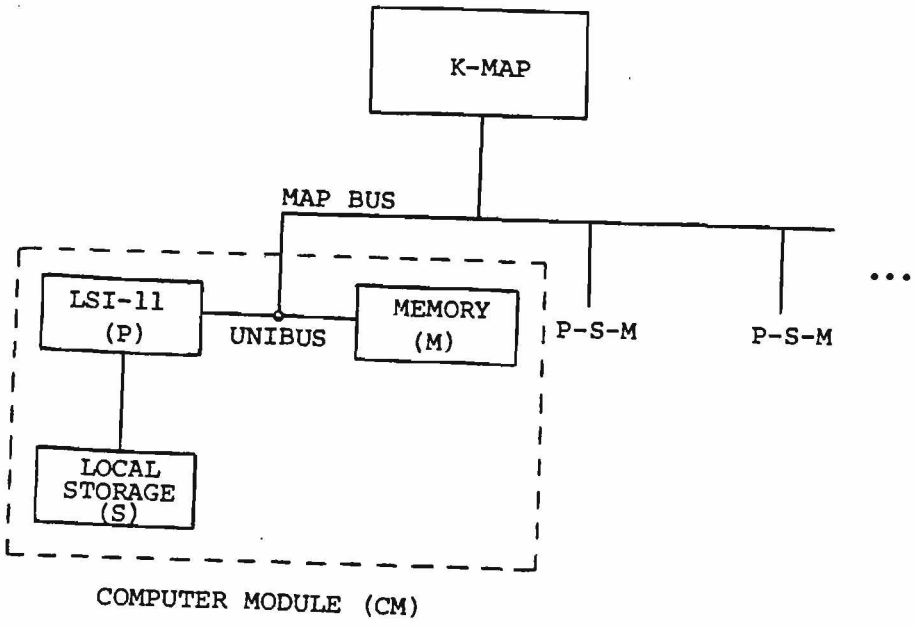


Figure 3-4: The Internal Topology of Each Cluster (K)

passed along the busses, but it does not initiate specific control sequences during its passage, and so is regarded as data. However, since both map and intercluster busses are packet switched, the control headers associated with the data increase the bus mobility over that of a simple circuit switched bus. The mobility is therefore low to medium. This suggests a computing structure whose adaptability to various problems is due more to the flexibility of the computing elements than to the adaptability of the communication paths connecting these elements. The bandwidth is high, but not as high as the on chip communication of systolic arrays, and the ATR is medium to low, indicating that the clusters and the elements comprising them can act with relative independence. This is consistent with the use of a general purpose microprocessor as the fundamental module in the CM* structure.

Each cluster is composed of up to fourteen computer modules and each computer module consists of up to 32K words of memory. Therefore a particular CM* must be configured before the grain sizes can be specified at the two levels distinguished here. This configuration is described in [8]. In this configuration, each CM has 32K words of memory, and each cluster has three CMs and a K-map. For the purposes of this discussion, the lines to the intercluster busses are considered to be a part of the K-map. Each CM has approximately 8,000 gates and 32,000 words of memory, while each K-map has approximately 17,000 gates and 10,000 words of storage. We conclude that the CM* clusters consist of rather large grains, each containing approximately 62,000 gates and 115,000 words of storage, including the S-local and the serial line units. Because the grain is so large, we expect the generality of processing within each cluster to be high and the cost of producing a system containing more than one hundred such clusters to be high by 1980 standards. Thus we expect the intercluster control to be rather "straightforward" bus arbitration logic well suited to busses serving 10 to 50 clusters. Were the cluster grain size to be smaller, permitting 1,000 or so clusters within economic constraints, we would expect the simple intercluster bus management to become a serious problem.

Within a cluster, the smaller CMs form a subsystem whose granularity is about 15,000 gates and 35K words of storage. These CMs are also rather large grains, and much the same remarks which applied to the clusters also apply to the CMs.

At the cluster level of granularity, the inputs to the clusters are data requests and the outputs are other data requests or the requested data. When a cluster receives a data request, it is the function of the K-map of each cluster to determine whether the request can be satisfied by storage of the cluster and satisfy it if possible.

If not, the request is passed to another cluster. The CMs within a cluster generate intercluster requests for all data not represented in storage within the cluster. The rather high ratio of logic gates to storage cells within the K-map, 17,000 to 10,000, is indicative of the complexity of address management, the virtual to physical address mapping, and the maintenance of what is in effect a cache memory of most recently accessed segment descriptors. Most of the memory is bipolar RAM in keeping with its cache-like operation.

The logic of each K-map is horizontally microcoded with 1K x 80 bit control storage. The wide control word suggests several concurrent operations and indicates that the speed of the logic was an important factor in the design.

Within the cluster, the CMs contain a memory/logic mix typical of general purpose computers: 8,000 gates and about 32,000 words of memory. Each processor is in fact an LSI-11, possessing a wide variety of addressing modes and a few data types. The processor and the memory are implemented in 16 bit parallel words following the minicomputer architectures of a few years ago.

3.3 THE CRAY-1

The Cray-1 [9] is well known as a very high performance machine. Much of the basis for this performance does not have anything to do with the replication of its functional components, and therefore is not well explained by the criteria presented in this paper. For example, the million word main store, the 12.5 nsec memory access time, and the basic pipelined organization of the processor are represented poorly in our taxonomy. The fact that the main store is so large removes much of the swap time which drastically limits the performance of many large compute bound programs. Storage intensive programs are further sped up by the extremely fast access time to the main store of the machine. The Cray-1 also has a large number of fast

registers, many of which can be loaded in parallel or in a pipelined manner. The processor organization also allows a long "look ahead" in the instruction stream to allow for a highly parallel execution of the apparently sequential instruction stream.

The Cray-1 represents a rather perverse case for our classification scheme and therefore presents a good opportunity to test the ideas. While it is possible to say that the Cray-1 does not primarily derive its power from replication, it is possible to explain much of the high performance nature of the machine via the four analysis parameters of our taxonomy structure.

At the highest level of analysis the Cray-1 CPU has a TG similar to many other commercial main frames. The TG is shown in Figure 3-5.

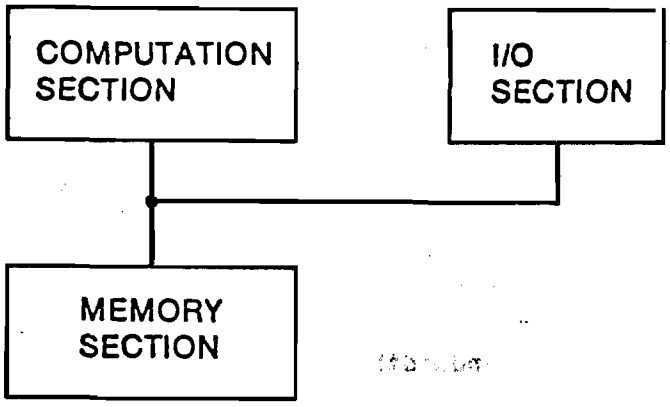


Figure 3-5: TG for Cray-1 CPU (Top Level)

The grain size for each of the elements of the TG is huge, the

processor alone consuming some 448 boards of about 110 chips each. The operation of the memory section is storage intensive, while the operation of the I/O and computation sections are both logic intensive. The topology of the system is evident from the figure and indicates sharing of the memory section by the I/O and computation sections. What cannot be determined at this level of analysis is the priority that the computation section has for access to the memory section. The analysis of communication for both paths starts to indicate the potential performance of the system; the bandwidths of both paths are very high. For example, a 64 bit data item, enough to contain four instructions, can be transferred in a single 12.5 nsec clock cycle. The very high bandwidth between elements indicates that the elements themselves must be capable of very high speed activity. At this level in our hierarchical analysis, we cannot yet determine why each element is so fast. Still only three basic possibilities exist:

1. The elements are made from very high speed circuit elements,
2. The elements contain subunits which are capable of utilizing parallelism to achieve composite high performance, or
3. A combination of the two.

In general, bandwidth is the key to raw speed analysis in that it describes an upper bound on the performance of both elements and paths. If the grain size is at the gate level, then bandwidth is an indication of circuit speed. At higher levels it indicates element speed which may be affected by both architectural and component attributes.

The mobility of each path is medium in that each path is used to carry large amounts of both program and data. The ATR for the computation section to memory section paths is very high and typically

is a relatively high percentage of the bandwidth. This indicates that neither the computation nor the memory section is capable of much independent activity without the other. The memory section to I/O section path ATR is much less than the communication section to memory section path ATR. This is due to the nature of the machine that programs tend to be resident in the CPU and therefore require relatively infrequent I/O.

In summary, the huge grain, very high bandwidth, very high ATR, medium mobility characteristics of the top level analysis of the Cray-1 indicate a very large (indicated by granularity) and very fast (indicated by bandwidth) general purpose (indicated by mobility) processor.

At this point we will dispense with looking deeper into the characteristics of the memory and I/O sections. We do this for two reasons:

1. The memory and I/O sections are less interesting in an architectural sense than the computation section, and
2. The computation section provides more interesting types of replication to illustrate our taxonomy ideas.

At the next level of analysis of the computation section we see several instances of replication. The topology of this level is shown in Figure 3-6.

This figure is the standard computation section diagram shown in the reference manual [9].

The granularity at this level varies between medium in the floating point units to small in the address registers. The topology is primarily three dimensional, using replication of the register classes and function units to form the third dimension. The topology shows the possibility for rather high parallelism in the computation section which results from replication of function units, registers,

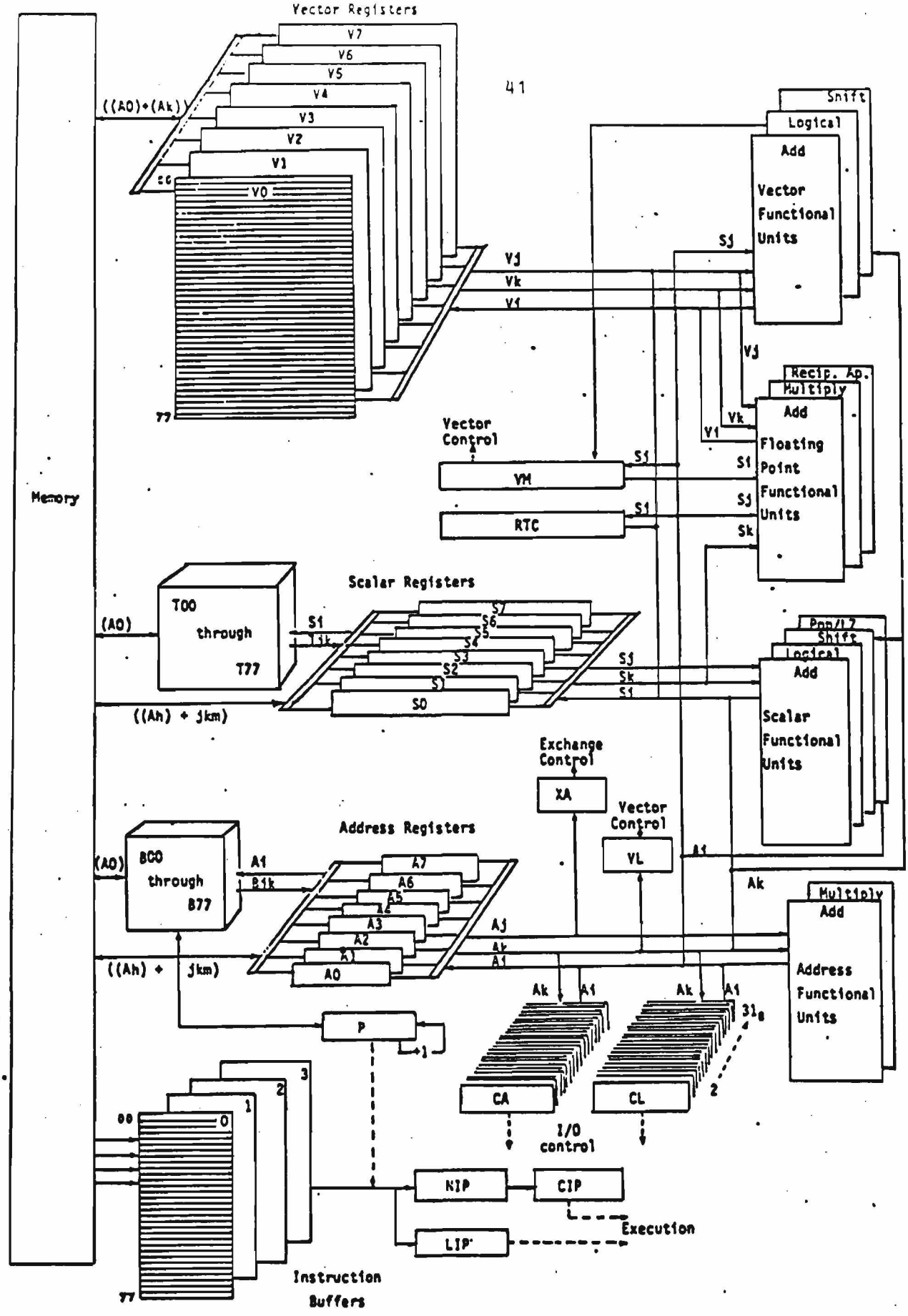


Figure 3-6: Cray-1 Computation Section

address buffers, and instruction buffers. In two dimensions, the pipelined nature of the computation section begins to become apparent. The operation of the elements at this level is also varied, but for any given element the mix is predominantly logic.

The communication parameter again gives the best clue to system performance. The mobility of the paths to and from the instruction buffers is high, and low for all other paths. This indicates the very specialized nature of the elements at this level. The bandwidth of all paths is still very high, indicating the very high performance of which each element is capable. The ATR varies both with path and the type of program being executed, but is typically low to medium. All these characterize a system of very high speed specialized elements.

At the next lower level of analysis, we mostly find that granularity of the replicated elements is at the gate level, but bandwidth is still very high. This indicates that the Cray-1 is constructed from very high speed circuit elements.

The discovery of the circuit speed, memory size, and the fact that the computation section allows both parallel and pipelined activity provides a fairly accurate view of why the Cray-1 is such a high performance computer.

3.4 THE MAGO REDUCTION MACHINE

An interesting machine which does derive its power from replication is the reduction machine of Mago [19], hereafter referred to as MRED. MRED differs from the Cray-1 in that it is not a commercially marketed machine; however, neither is it a paper machine in that a prototype of the MRED is being considered by Mago and his associates. It has been seen that the bandwidth of any system is affected by the circuit speed of the logic elements. In the case of MRED, it is inappropriate to make exact bandwidth measures because the prototype could be cast into a number of circuit types. The spirit of

the effort indicates the use of commercially available microprocessors, and therefore NMOS circuit speeds could largely determine bandwidth.

At the top level of analysis, MRED is a leaf connected binary tree. The leaf elements, L, are of one type and the non-leaf elements, T, are of another type. The TG of MRED at the top level is shown in Figure 3-7. Note that each of the T elements of MRED is doubly connected to father and son elements via a bidirectional path. For any given program only a subset of these paths will be active as part of the partitioning scheme used by MRED.

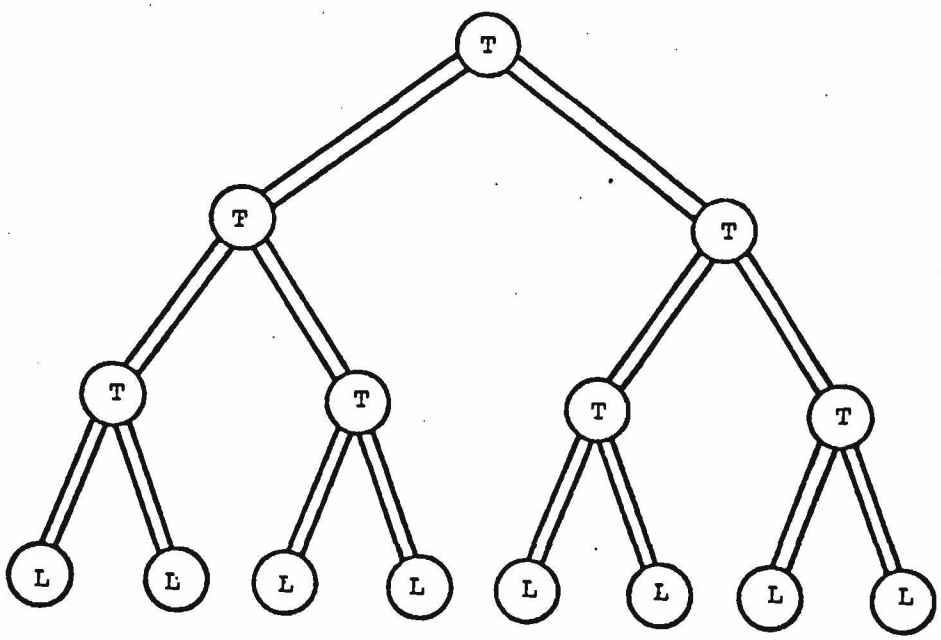


Figure 3-7: TG for MRED at the Top Level

The grain size of the T elements is medium and will be seen at the next level to be about the size of four microprocessors. The grain size of the L elements is small and about the size of a single microprocessor. The operation of the T elements is logic intensive;

each T element acts like up to four processors which are used in executing Backus like functional programs [3]. There is some storage in each T element for microcode, and each T element also has a small number of registers for result queuing. The operation mix of the L elements is also logic intensive, but the function of the L elements is only to store program symbols and microcode. The use of logic in the L elements is therefore relegated to storage management tasks, and to microcode execution.

The communication bandwidth of all of the paths is medium, but the ATR is low. Mobility tends to be high because both microcode segments and machine language segments are carried over the paths in order to set up the parallel evaluation of reduction style programs. The high mobility, low ATR, medium bandwidth communication profile suggests a machine which tends to customize itself for a given special computation. It also suggests that each element is of medium performance, but the topology indicates that high performance is possible if parallelism can be exploited.

Examination of MRED at the next level down does not yield much more useful insight into the nature of MRED, but we conduct such an analysis for the sake of completeness.

At the next level the L element consists of a number of registers connected to a logic block which is a microprocessor. The TG for such a configuration is straightforward and we therefore omit it. The grain size of the microprocessor is small and the operation is logic intensive, while the registers are very small memory grains. Below this level, gates are seen as the replicated structure. The paths remain high mobility, medium bandwidth, low ATR in nature.

The TG for the second level analysis of a T element is shown in Figure 3-8.

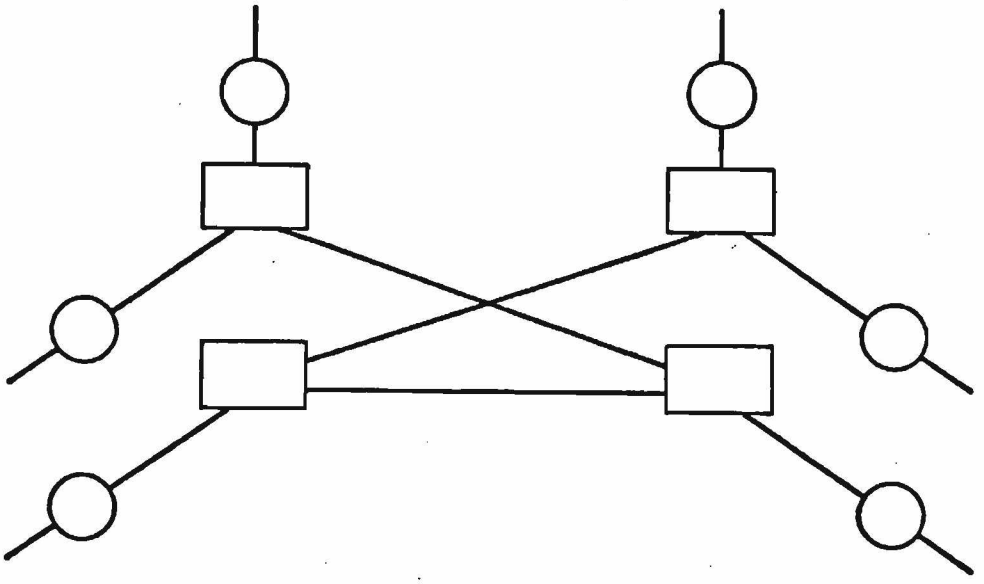


Figure 3-8: The TG for Second Level MRED T Elements

The round elements in the TG are register groups and as such are very small grain, memory intensive elements. The square elements are microprocessors and as such are small grain logic intensive devices. The paths are still medium bandwidth, low ATR, high mobility with respect to their communication characteristics. The next lower level analysis is again the gates themselves.

This deeper analysis of MRED reinforces the first level view of a possibly high performance collection of medium performance grains.

4. CONCLUSION

We have presented a scheme which should be valuable as an aid in characterizing parallel systems. The main use of this scheme is intended to be for analyzing digital systems which derive their power from replication of system sub-elements. It extends previous classification attempts to account for functional properties rather than simply considering structure. The use of this scheme has been shown via example analyses of 4 different computer systems. It is our belief that replicated systems will be the predominant architectures of the future. This is primarily caused by improvements in the technology of integrated circuits. We feel that use of the analysis criteria presented here can result in very useful insights into the suitable application and performance of these future parallel systems.

REFERENCES

1. A. L. Davis. The Architecture of DDM1: A Recursively Structured Data-Driven Machine. Tech. Rept. UUCS-77-113, University of Utah, Computer Science Dept., 1977.

2. G. A. Anderson and E. D. Jensen. "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples." Computing Surveys 7, 4 (December 1975), 197-213.

3. J. Backus. "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs." CACM 21, 8 (1978), 613-641.

4. G. H. Barnes et al.,. "The ILLIAC IV Computer." IEEE Transactions 8, 6/E17 (August 1968), 746-757.

5. C. G. Bell and A. Newell. Computer Structures: Readings and Examples. McGraw-Hill, 1971.

6. C. G. Bell, J. C. Mudge, and J. E. McNamara. Computer Engineering: A DEC View of Hardware Systems Design. Digital Equipment Corp., 1978.

7. R. O. Bert et al. PEPE--An Overview of Architecture Operation and Implementation. Proc. National Elec. Conference, 1972, pp. 312-317.

8. Computer Science Department. "A Case Study of C.mmp, CM* and C.Vmp: Part II--Predicting and Calibrating Reliability of Multiprocessor Systems." Proceedings of the IEEE 66, 10 (October 1978), 1200-1220.

9. Cray Research Corporation. CRAY-1 Computer System reference manual. Tech. Rept. 2240004, CRAY Research Incorporated, 1976.

10. M. Denny, E. R. Buley, E. Hatt. Logic-enhanced Memories: An Overview and Some Examples of Their Application to a Radar Tracking Problem. Proceedings of Caltech Conference on VLSI, Caltech, January, 1979.

11. M. J. Flynn. "Some Computer Organizations and Their Effectiveness." IEEE Transactions on Computers C-21, 9 (September 1972), 948-960.

12. W. Handler. The Impact of Classification Schemes on Computer Architecture. Parallel Processing Symposium, August, 1977.

13. L. C. Higbie. "Supercomputer Architecture." IEEE Transactions on Computers C-22, 12 (December 1973), 48-58.

14. L. C. Hobbs and D. J. Theis. Survey of Parallel Processor Approaches and Techniques. In Parallel Processor Systems, Technologies, and Applications, Spartan Books, New York, N.Y., 1970, pp. 3-20.

15. J. Holland. A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously. Proc. 1959 Eastern Joint Computer Conference, AFIPS, 1959, pp. 108-113.

16. H. T. Kung. Systolic Arrays for VLSI. In Introduction to VLSI Systems, McGraw-Hill, 1979.

17. H. T. Kung. Let's Design Algorithms for VLSI Systems. Proceedings of Caltech Conference on VLSI, Caltech, January, 1979.

18. C. E. Leiserson. Systolic Priority Queues. Proceedings of Caltech Conference on VLSI, Caltech, January, 1979.

19. G. A. Mago. "A Network of Microprocessors to Execute Reduction Languages, Part I." International Journal of Computer and Information Sciences 8, 5 (March 1979 revised), 349-385.

20. J. C. Murtha and R. L. Beadles. Survey of Highly Parallel Information Processing Systems. Tech. Rept. 4755, ONR, November, 1964.

21. E. I. Organick. New Directions in Computer Architecture. Euromicro Journal, July, 1979.

22. J. A. Rudolph. A Production Implementation of an Associated Array Processor - STARAN. AFIPS Proc. FJCC 41, AFIPS, 1972, pp. 229-241.

23. C. L. Seitz. System Timing. In Introduction to VLSI Systems, Chapter 7, McGraw-Hill, 1979.

24. H. J. Siegel et al.,. PASM: A Partitionable Multimicrocomputer SIMD/MIMD System for Image Processing and Pattern Recognition. Tech. Rept. TR-EE-79-40, Purdue University, August, 1979.

25. S. Smoliar and R. E. Frankel. Beyond Register Transfer: An Algebraic Approach for Architectural Description. Proceedings of the International Symposium on Computer Hardware Description Languages and Their Applications, Palo Alto, California, 1979.

26. I. E. Sutherland and C. A. Mead. "Microelectronics and Computer Science." Scientific American 237, 3 (September 1977), 210-228.

27. W. T. Wilner. The Design of the Burroughs B1700. Proceedings of FJCC, 1972.