MODULE V

Dataflow computers - Data driven computing and Languages - Data flow computers architectures - Static data flow computer -Dynamic data flow computer -Data flow design alternatives.

DATA-DRIVEN COMPUTING AND LANGUAGES

- Data flow computers are based on the concept of *data-driven computation,* which is drastically different from the *operation* of a conventional von Neumann machine
- The fundamental difference is that *instruction execution in* a conventional computer is under *program-flow control*, whereas that in a *data flow* computer is driven by the *data (operand)* availability

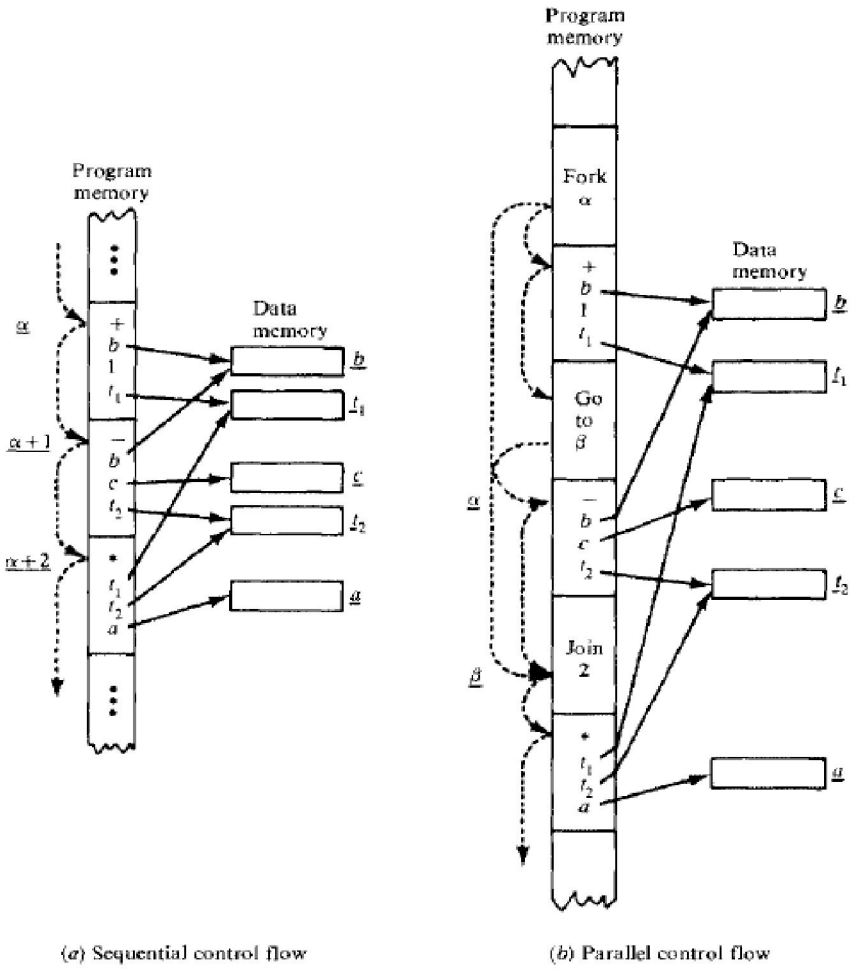
CONTROL FLOW VS DATA FLOW COMPUTERS

- The concepts of control flow and data flow computing are distinguished by the control of computation sequences in two distinct program representations.
- The statement $a = (b + 1). (b - c)$ is specified by a series of instructions with an explicit flow of control.
- Shared memory cells are the means by which data is passed between instructions. Data (operands) are referenced by their memory addresses (variables).

In the traditional *sequential control flow* model (von Neumann), there is a single thread of control, as shown in Figure which is passed from instruction to instruction.

Explicit control transfers are caused by using operators like GO TO.

In the *parallel control flow* model (Figure b), special parallel control operators such as FORK and JOIN are used to explicitly specify parallelism; These operators allow more than one thread of control to be active at an instant and provide means for synchronizing these threads, as demonstrated in Figure b. All underlined variables refer to addresses of operands and instructions.

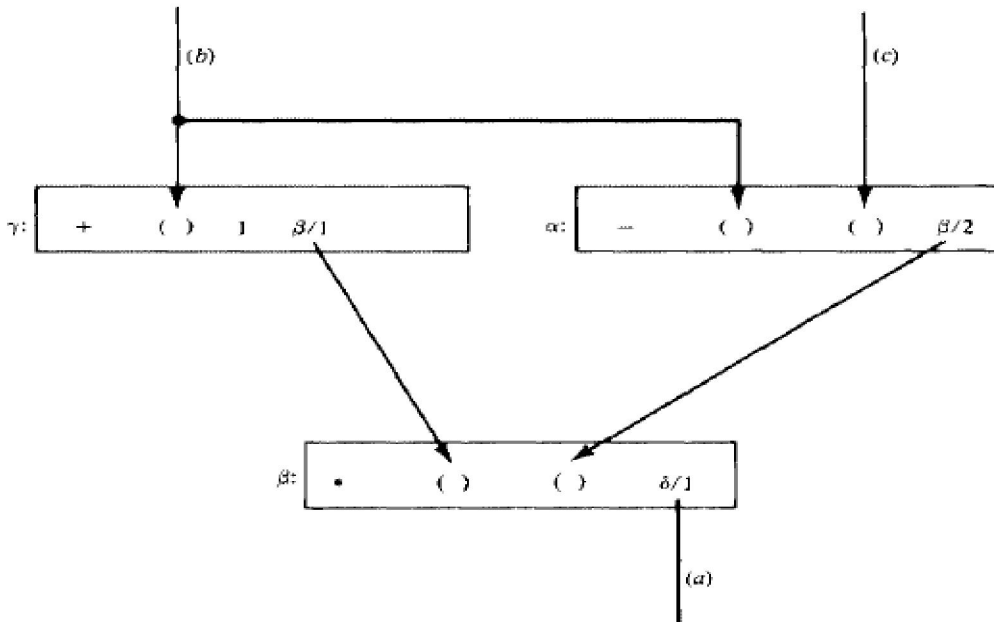


(a) Sequential control flow (b) Parallel control flow

Special features are identified below for either the sequential or parallel control flow model:

- Data is passed between instructions via references to shared memory cells.
- Flow of control is implicitly sequential, but special control operators can be used explicitly for parallelism.
- Program counters are used to sequence the execution of instruction in a centralized control environment.

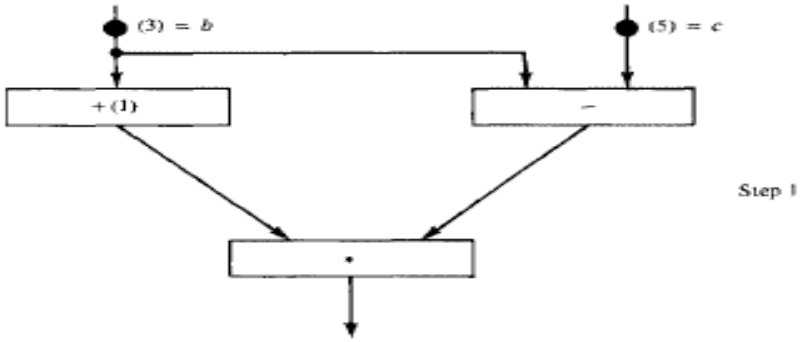
In a data flow computing environment, instructions are activated by the availability of data tokens indicated by the ( ) in Figure given below.Data flow programs are represented by directed graphs, which show the flow of data between instructions. Each instruction consists of an operator, one or two operands, and one or more destinations to which the result (data token) will be sent.



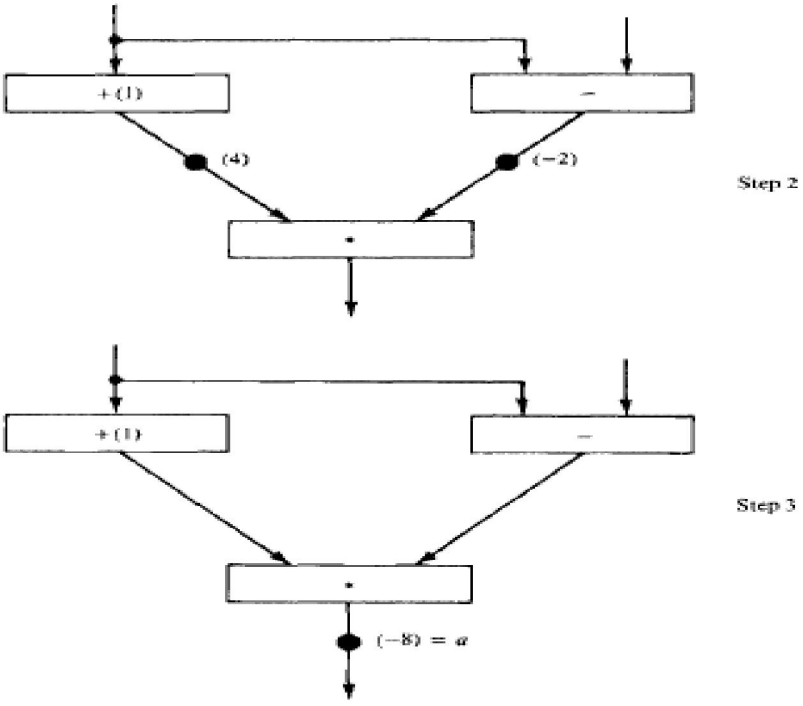
Three snapshots of the data flow graph for $a = (b + 1). (b - c)$ are shown in Figure below

Each instruction consists of an operator, one or two operands. The dots correspond to data tokens being passed between instructions. Listed below are interesting features in the data flow model:

- Intermediate or final results are passed directly as data token between instructions.
- There is no concept of shared data storage as embodied in the traditional notion of a variable.
- Program sequencing is constrained only by data dependency among instructions
-



Step 1



Control flow computers have a *control-driven* organization. This means that the program has complete control over instruction sequencing. Synchronous computations are performed in control flow computers using centralized control.

Data flow computers have a *data-driven* organization that is characterized by passive examine stage. Instructions are examined to reveal the operand availability, upon which they are executed immediately-if the functional units are available.

Depending on the way of handling data tokens data flow computers are divided into

- **STATIC MODEL**
- **DYNAMIC MODEL**.

- In a static data flow machine data tokens are assumed to move along the arcs of the data flow program graph to the operator nodes.
- The nodal operation gets executed when all its operand data are present at the input arcs. Only one token is allowed to exist on any arc at any given time, otherwise the successive sets of tokens cannot be distinguished.
- This architecture is considered static because tokens-are not labeled and control tokens must be used to acknowledge the proper timing in transferring data tokens from node to node.

- A dynamic data flow machine uses tagged tokens so that more than one token can exist in an arc.
- The tagging is achieved by attaching a label with each token which uniquely identifies the context of that particular token.
- This dynamically tagged data flow model suggests that maximum parallelism can be exploited from a program graph.
- If the graph is cyclic, the tagging allows dynamically unfolding of the iterative computations.
- Dynamic data flow computers include the Manchester machine developed by Watson and Gurd at the University of Manchester, England, and the Arvinds machine under development at MIT, which is evolved from an earlier data flow project at the University of California at Irvine.

STATIC DATA FLOW COMPUTERS

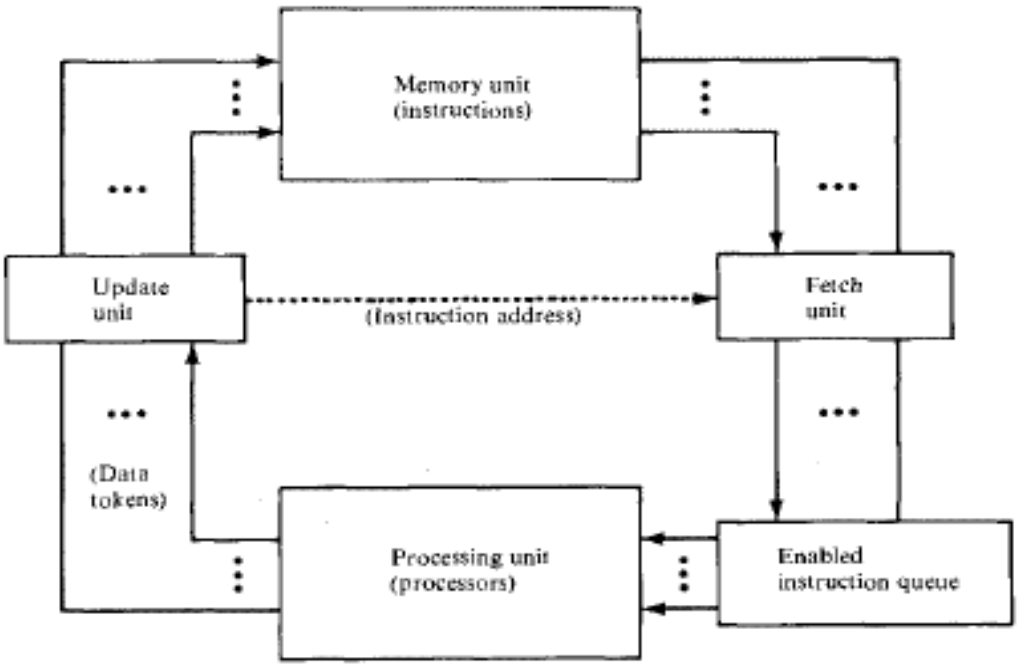


Figure 10.4 A static dataflow computer organization.

- The data tokens are in the input pool of the update unit.

- This update unit passes data tokens to their destination instructions in the memory unit.
- When an instruction receives all its required operand tokens, it is enabled and forwarded to the enabled queue.
- The fetch unit fetches these instructions when they become enabled.

DYNAMIC DATA FLOW COMPUTERS

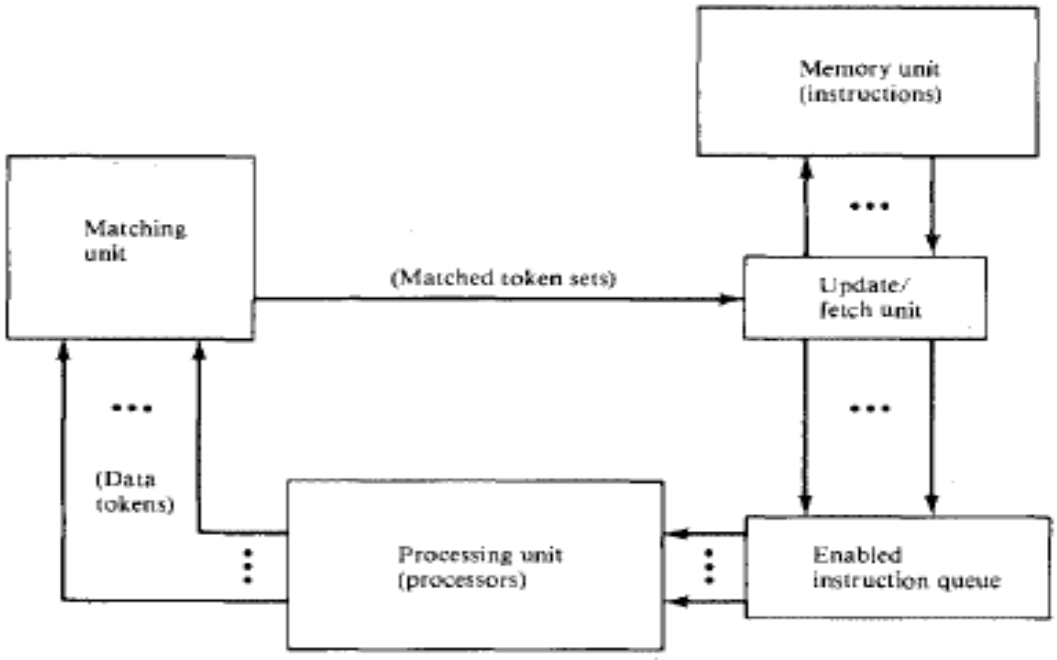


Figure 10.5 A dynamic dataflow computer organization.

A dynamic data flow machine uses *tagged tokens* so that more than one token can exist in an arc. The tagging is achieved by attaching a label with each token which uniquely identifies the context of that particular token. This dynamically tagged data flow model suggests that maximum parallelism

The system synchronization is based on a matching mechanism

- Data tokens form the input pool of the matching unit .

- This matching unit arranges data tokens into pairs or sets and temporarily stores token until all operands are compared, where upon the matched token sets are released to the fetch-update unit.
- Each set of matched data tokens (usually two for binary operations) is needed for one instruction execution. The fetch-update unit forms the enabled instructions by merging the token sets with copies sent to their consumer instructions.
- The matching of the special tags attached to the data tokens can unfold iterative loops for parallel computations

Generalized architecture

Both static and dynamic data flow architectures have a pipelined ring structure.

- If we include the I/O, a generalized architecture is shown in Figure.
- The ring contains four resource sections: **the memories, the processors, the routing network, and the input-output unit.**
- The memories are used to hold the instruction packets. The processing units form the task force for parallel execution of enabled instructions. The routing network is used to pass the result tokens to their destined instructions.
- The input-output unit serves as an interface between the data flow computer and the outside world. For dynamic machines, the token matching is performed by the I/O section.

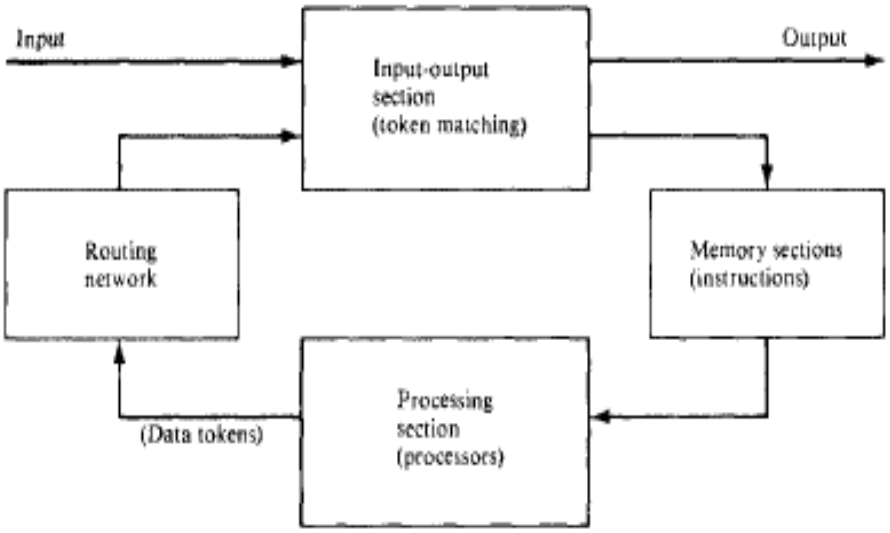


Figure 10.6 A ring-structured dataflow computer organization including the I/O functions.

Data flow graphs and languages

- There is a need to provide a high-level language for data flow computers.
- The primary goal is to take advantage of implicit parallelism
- An efficient data flow language should be able to express parallelism in a program

Examples of data flow languages include the Irvine Data- flow language and Value Arithmetic Language (VAL).

In a maximum parallel program, the sequencing of instructions should be. constrained only by data dependencies and nothing else. Listed below are useful properties in a data flow language. We shall describe their implications and usages separately.

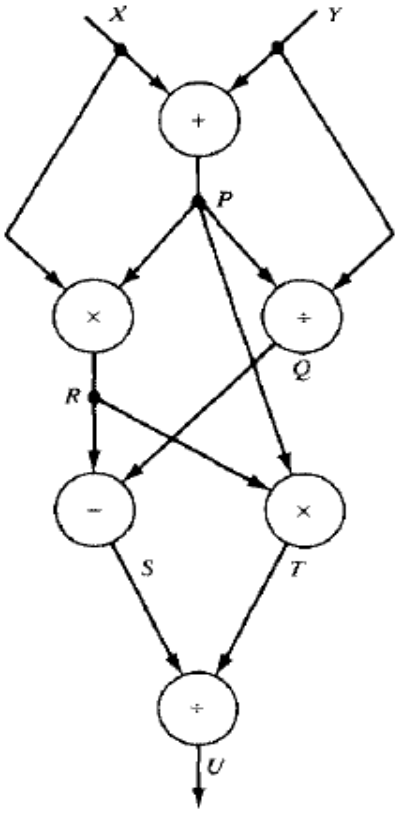
- Freedom from side effects based on functional programming .
- Locality of effect without far-reaching data dependencies
- Equivalence of instruction-sequencing constraints with data dependencies .
- Satisfying single-assignment rule with aliasing
- Unfolding of iterative computations into parallelism .
- Lack of "history sensitivity" in procedures calls

In a conventional computer, program analysis is often performed at compile time to yield better resource utilization and code optimization and at run time to reveal concurrent arithmetic logic activities for higher system throughput.

Data flow graph for the computation of
U=f(X<Y)=(X*(X*Y)-(X+Y)/(X*(X+Y)*(X+Y))

1. P = X + Y must wait for inputs X and Y
2. Q = P ÷ Y must wait for instruction 1 to complete
3. R = X × P must wait for instruction 1 to complete
4. S = R − Q must wait for instructions 2 and 3 to complete
5. T = R × P must wait for instruction 3 to complete
6. U = S ÷ T must wait for instruction 4 and 5 to complete

This program can be represented by the following data flow graph



A data flow graph is a directed graph whose nodes correspond to operators and and arcs are pointers for forwarding data tokens.The firing rule is based on data availability.

Different nodes and links are given below.

- T gate passes a data token from its input arc to its output arc when it receives the true value on its control arc.
- F gate :same as above except control value is reversed.
- Identity operator sends its input value unchanged.

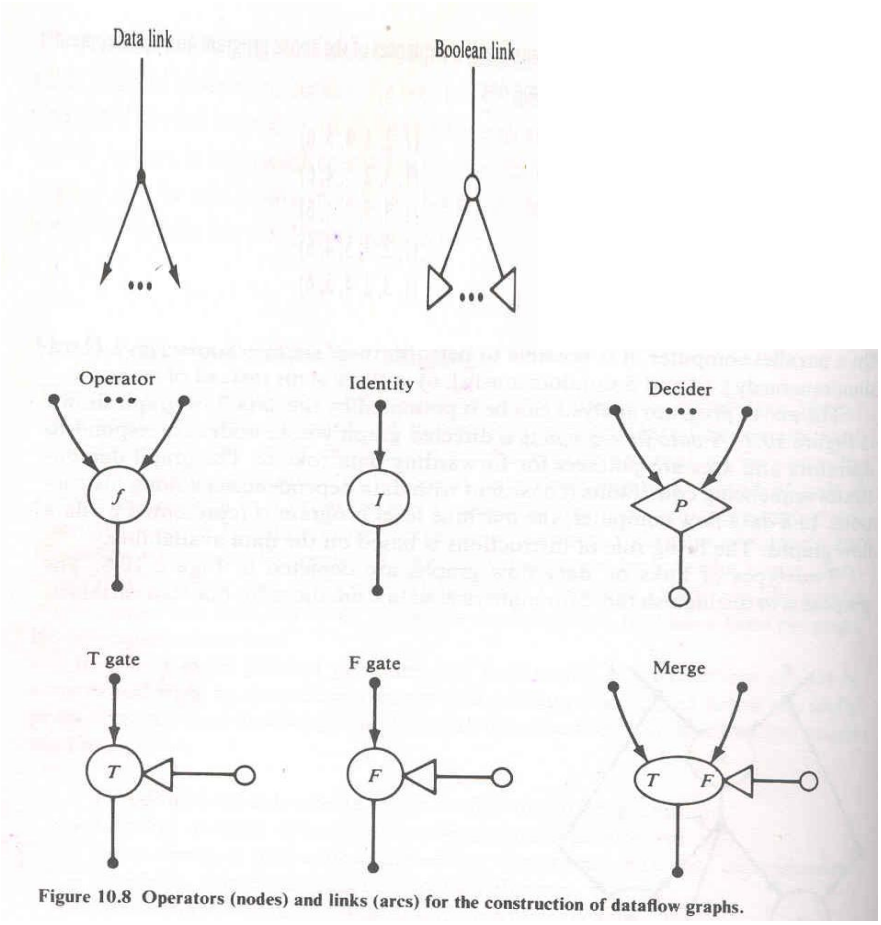


Figure 10.8 Operators (nodes) and links (arcs) for the construction of dataflow graphs.

Advantages and Potential Problems

Highly concurrent operations:

Parallelism can be easily exposed in a dataflow program graph. The data flow approach offers a possible solution to the problem of efficiently exploiting concurrency of computation on a large scale.It benefits not only regularly structured but also arbitrary parallelism in programs. ", direct use of values instead of names of value containers (addresses) enables pure functional programming without side effects. Asynchronous parallelism can exploited at the instruction level or at the procedure level. Inherently sequent computations can be unfolded to enable parallelism. The data flow approach I

applied pipelining, array processing, and multiprocessing techniques discussed previous chapters for control flow computers.

The data flow language does not introduce instruction-sequencing constraiJ1 other than the ones imposed by data dependencies in the algorithm. In theory maximum parallelism can be achieved if sufficient resources are provided. This approach extends naturally to an arbitrary number of processors in the system The speedup should be linearly proportional to the increase of processor number The high concurrency in a data flow computer is supported by easier prograJ1 verification, better modularity and extendibility of hardware, reduced protection problems, and superior confinement of software errors.

Matching with VLSI technology

Recall the basic architecture of a data flow computer (Figure 10.6). The memory section contains instruction cells which can be uniformly structured in large-scale memory arrays. The pool of processing units and the network of packet switches can be each also regularly structured with modular cells. All this homogeneity and modularity in cellular structures contributes to the suitability of VLSI implementation of major components in a data flow computer. As introduced in Chapter I, the impressive progress in microelectronics technology has made it possible to challenge the fabrication of large arrays of processors, memories, and switches on VLSI chips.

The interconnection between chips can be built into highly densed packaging systems. It is fair to say that data flow machine architecture matches nicely with the technological supports that we anticipate to have. The potential of VLSI and VHSIC technologies can be fully exploited in the development of data flow machines. The operations in a data flow computer may be asynchronous. How- ever, the hardware components can be designed with synchronous functional pipes and clocked memory and switch arrays. With more lessons to be learned and the data flow hardware properly evaluated, it would be appropriate to consider VLSI implementation of some large-scale data flow systems.

Programming productivity

In a control flow vector processor or a multiprocessor system, the percentage of code that can be vectorized ranges from 10 to 90 ' percent across a broad range of scientific applications. The non vectorizable code (scalar operations) tends to become a bottleneck. Automatic vectorization requires sophisticated data flow analysis, which is difficult in Fortran because of the side effects caused by the global scope and aliasing of variables. A well-designed data flow computer should be able to overcome these difficulties and to remove the bottleneck caused by assorted scalar operations.

Shortcomings of data flow computing

Critics of the data flow approach have pointed out quite a number of potential problems in the development and application of data flow computers at the instruction level. It is instructional to learn n these reserved positions and to explore other alternatives to achieve high performance. In the conventional computer with centralized control hardware, an imperative language such as Fortran is used and an intelligent compiler is needed normalize the program and to generate the dependence graph, which guides vectorization and optimization processes we have studied in previous chapters. h-level use of the dependence graph is practiced here primarily at compile e. The major advantages of this high-level approach are summarized below:

- There is no need to use a new functional programming language, which ordinary programmers may be reluctant to learn.
- Existing software assets for vectorizing,compiling and dedicated application software can continue to be utilized.
- Data flow analysis at the higher level of procedures and loops will result in less overhead when averaged over all instructions to be executed.

In the data flow approach, special functional programming languages must used which can be easily compiled into a dependence graph. The object code generated to efficiently map the dependence graph onto the data flow machine h distributed control hardware. Some advantages of high-level data flow machines become potential shortcomings in the data flow computer, which exploits parallelism at the lowest level of instruction execution. Apparent disadvantages instructional-level data flow computers are summarized below:

The data driven at instruction level causes excessive pipeline overhead per instruction, which may destroy the benefits of parallelism. The long pipeline filling problem is attributed to queueing all enabled instructions at the input ports of every subsystem in the data flow ring. The queue lengths absorb some of the parallelism in a program, thus, performance becomes weak for improper buffering and traffic congestion. Data flow programs tend to waste memory space for the increased code length due to the single assignment rule and the excessive copying of data arrays. The damaging effects of the memory access conflict problem are so far not well addressed by data flow researchers.

When a data flow computer becomes large with high numbers of instruction cells and processing elements, the packet-switched network used becomes cost-prohibitive and a bottleneck to the entire system.

Listed below are useful properties in a data flow language.

- Freedom from side effects based on functional programming .
- Locality of effect without far-reaching data dependencies
- Equivalence of instruction-sequencing constraints with data dependencies .

- Satisfying single-assignment rule with aliasing
- Unfolding of iterative computations into parallelism .
- Lack of "history sensitivity" in procedures calls

Locality of effect This property can be achieved if instructions have no unnecessary far-reaching data dependencies. In Algol or Pascal, blocks and procedures provide some locality by making assignment only to local variables. This means that global assignments and common variables should be avoided. Data flow languages generally exhibit considerable locality. Assignment to a formal parameter should be within a definite range. Therefore, block structures are highly welcome in a data flow language.

Freedom from side effects This property is necessary to ensure that data dependencies are consistent with sequencing constraints. Side effects come in many forms, such as in procedures that modify variables in the calling program. The absence of global or common variables and careful control of the scopes of variables make it possible to avoid side effects. Another problem comes from the aliasing of parameters. Data flow languages provide "call by value" instead of the "call by reference." This essentially solves the aliasing problem. Instead of having a procedure modify its arguments, a "call by value" procedure copies its arguments. Thus, it can never modify the arguments passed from the calling program. In other words, inputs and outputs are totally isolated to avoid unnecessary side effects.

Single assignment rule This offers a method to promote parallelism in a program. The rule is to forbid the use of the same variable name more than once on the left-hand side of any statement. In other words, a new name is chosen for any redefined variable, and all subsequent references are changed to the new name. This concept is shown below:

$$\begin{array}{lll} X := P - Q & & X := P - Q \\ X := X \times Y & \rightarrow & X1 := X \times Y \\ W := X - Y & & W := X1 - Y \end{array} \qquad (10.2)$$

The statements on the right are made to satisfy the single assignment rule. It greatly facilitates the detection of parallelism in a program. Single assignment rule offers clarity and ease of verification, which generally outweighs the convenience of reusing the same name. Single assignment rule was first proposed by Tesler and Enea in 1968. It has been applied in developing the French data flow computer LAU and the Manchester data flow machine in England.

Unfolding iterations A programming language cannot be considered as effective if it cannot be used to implement iterative computations efficiently. Iterative computations are represented by "cyclic" data flow graphs, which are inherently sequential. In order to achieve parallel processing, iterative computations must be unfolded. Techniques have been suggested to use tagged tokens to unfold

DATA FLOW COMPUTER ARCHITECTURE

1. **Static data flow computers**
   - Dennis machine at MIT
2. **Dynamic data flow computers**
   - Irvine machine and its successor machine at MIT.
   - the EDDY system in Japan
   - the Manchester machine in England.

Static Data Flow Computers.

Jack Dennis and his associates at MIT have pioneered the area of data flow research. They have developed a static data flow computing model and the associated language supports.

- Data flow graphs used in the Dennis machine must follow a static execution rule that **only one data token can occupy an arc at an instant.**
- This leads to a static firing rule that an instruction is enabled if a data token is present on each of its input arcs and no token is present on any of its output arcs.
- Thus. the program graph contains control tokens as well as data tokens, both contributing to the enabling of an instruction.
- These control tokens act as acknowledge signals when data tokens are removed from output arcs.
- The Dennis machine is designed to exploit the concurrency in programs represented by static data flow graphs.

It consists of five major sections connected by channels through which information is sent in the form of discrete tokens (packets):

- **Memory *section*** consists of instruction cell which hold instructions and their operands.
- ***Processing section*** consists of processing units that perform functional operations on data tokens.

- **Arbitration network** delivers operation packets from the memory section to the processing section
- **Control network** delivers a control token from the processing section to the memory section.
- **Distribution network** delivers data tokens from the processing section to the memory section

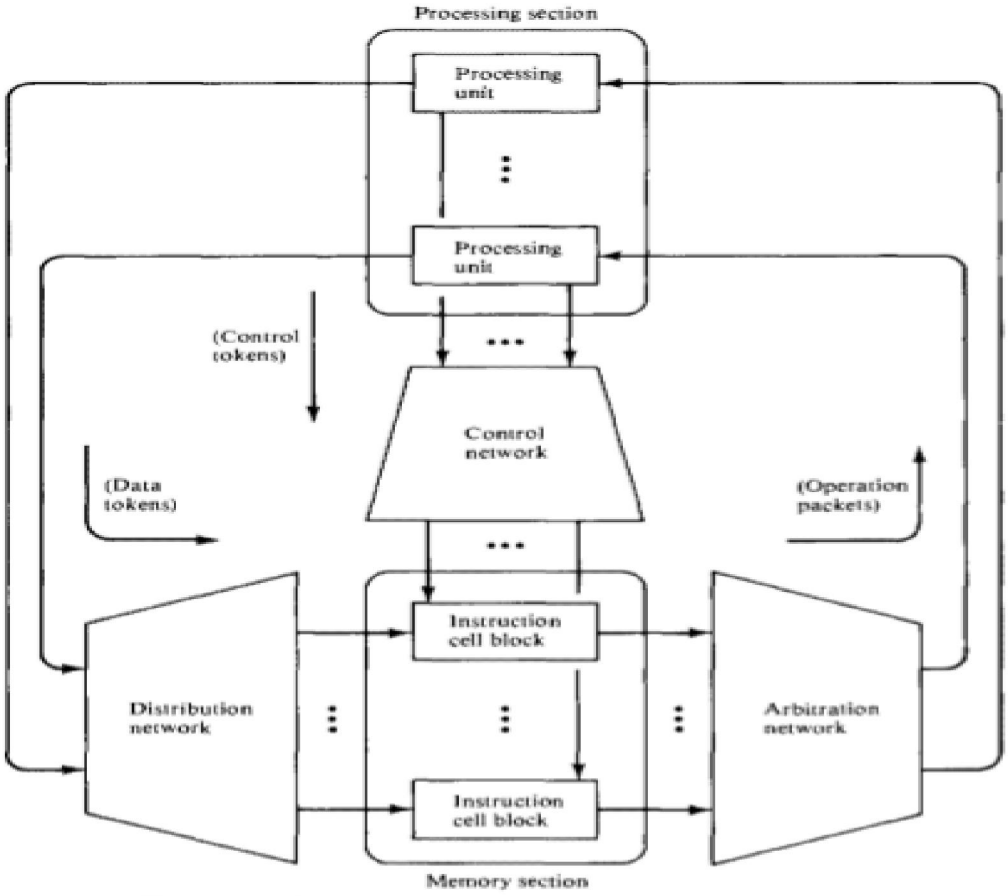


Figure 10.10 The static dataflow computer architecture proposed at MIT. (Courtesy of Dennis et al., 1979.)

When the number of instruction cells become large, concept of cell blocks can be used.

Cell block is a collection of instruction cells which share same set of input and output ports from the distribution, arbitration and control networks.

Cell block implementation is shown below.

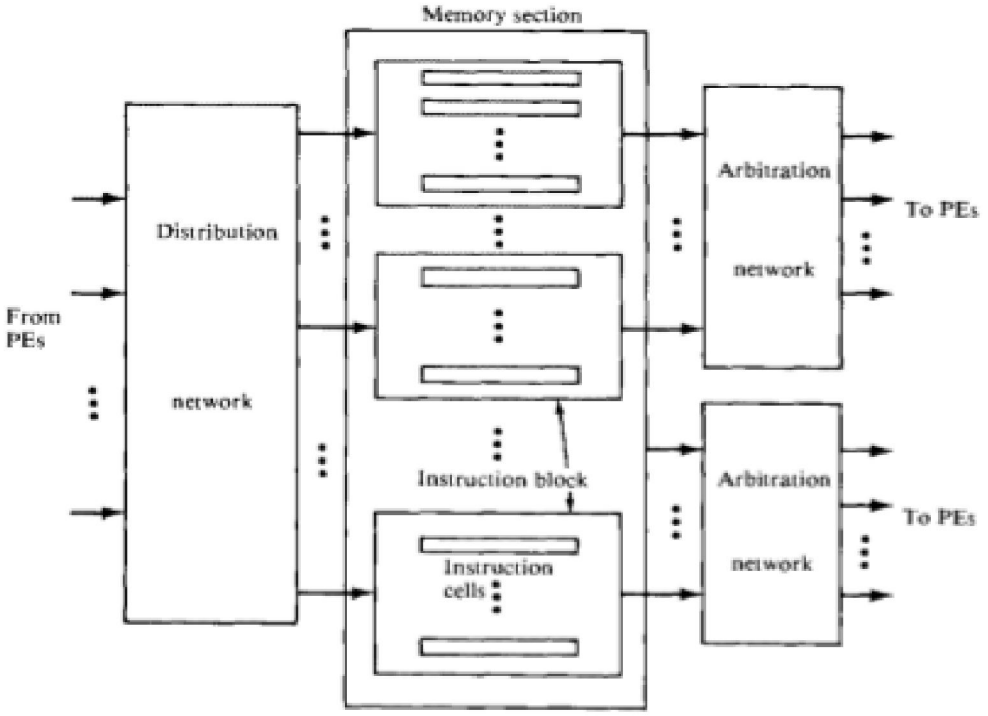


Figure 10.11 The concept of grouping instruction cells into cell blocks.

Design of cell block structured data flow multiprocessor system is shown below. System consists of 4 processors and 32 cell blocks. Three building blocks can be used to construct the 32X 4 arbitration network and the 4X32 distribution network.

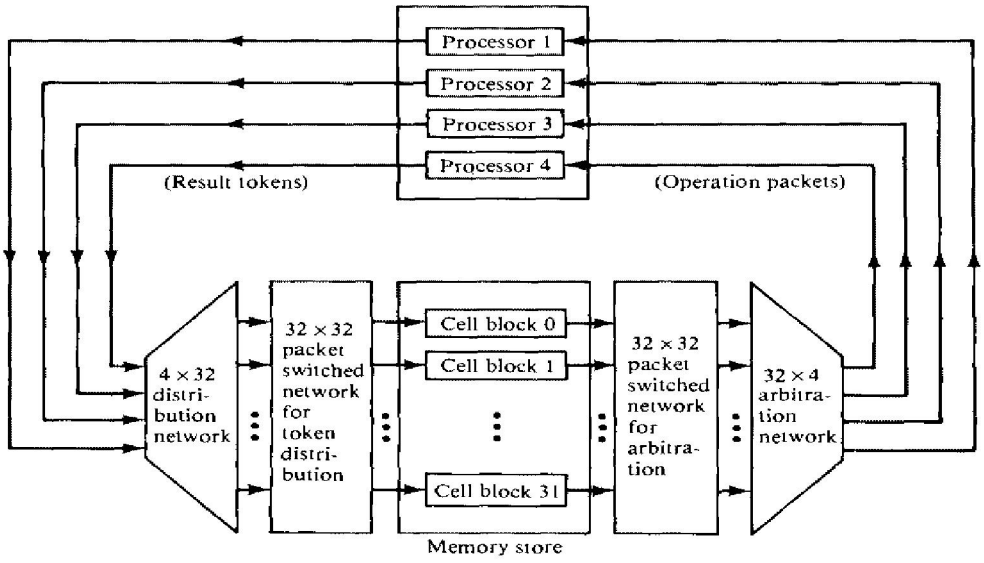


Figure 10.12 A 4-processor dataflow computer with 32 cell blocks interconnected by a 32 × 4 arbitration network and a 4 × 32 distribution network.

Dynamic Data Flow Computers

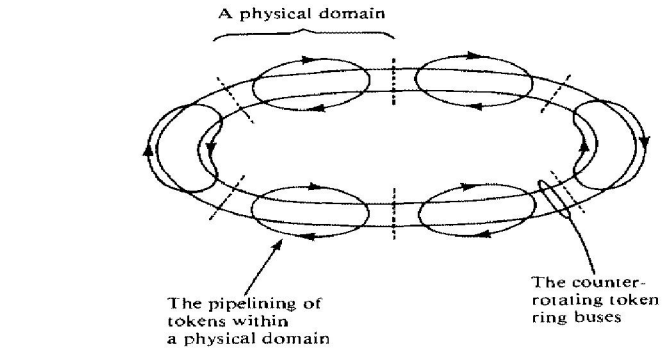
- In dynamic machines, data tokens are *tagged* (labeled or colored) to allow multiple tokens to appear simultaneously on any input arc of an operator node.
- No control tokens are needed to acknowledge the transfer of data tokens among instructions.
- Instead, the matching of token tags (labels or colors) is performed to merge them for instructions requiring more than one operand token.
- Therefore, additional hardware is needed to attach tags onto data tokens and to perform tag matching

1. The Irvine dataflow computer.

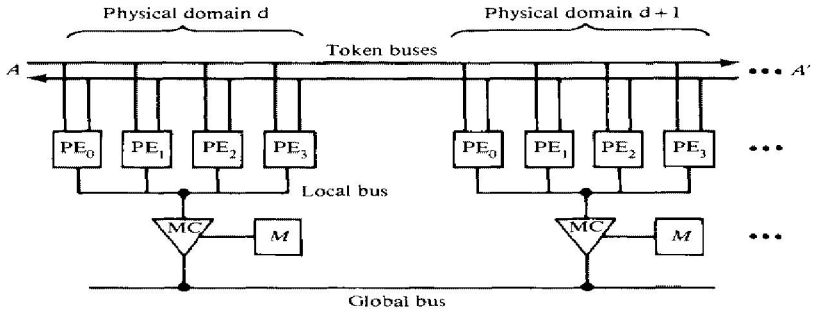
- The Irvine machine was proposed to consist of multiple PE clusters.
- All PE clusters can operate concurrently
- The physical domains are interconnected by two system buses.
- The token bus is a pair of bi-directional shift-register rings.

Each ring is partitioned into as many slots as there are PEs and each slot is either empty or holds one data token.

- Each cluster of PEs share a local memory through a local bus and a memory controller.
- A global bus is used to transfer data structures among the local memories.
- Each PE must accept all tokens that are sent to it and sort those tokens into groups by activity name.
- The U- interpreter can help implement iterative procedure computations by mapping the loop or procedure instances into the PE clusters for parallel executions



(a) Token rings and local pipelining

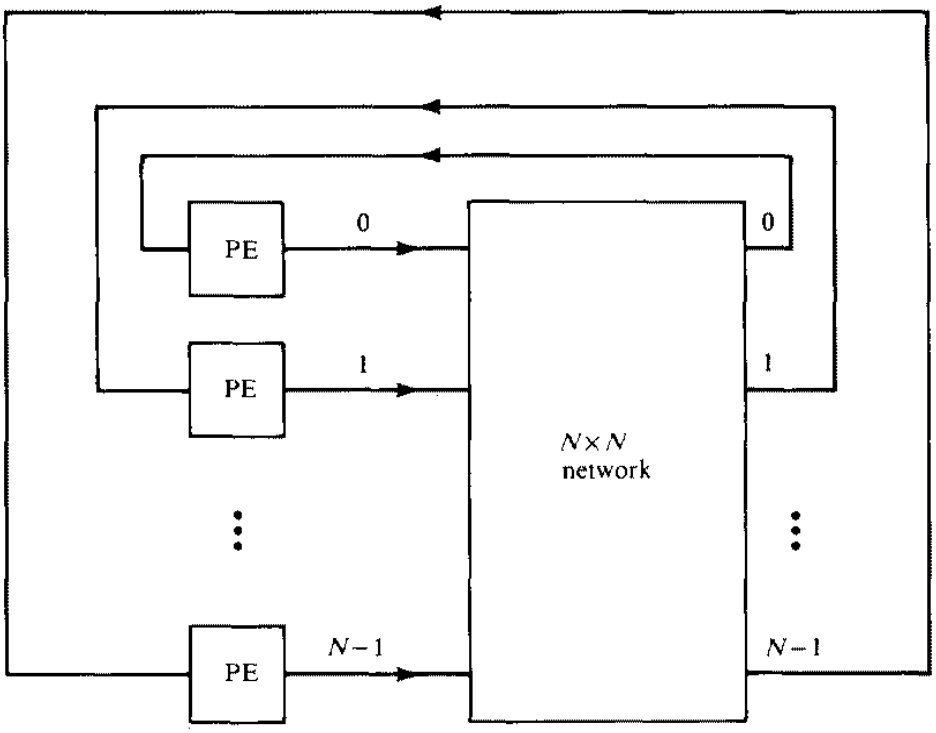


(b) Processor clusters and interconnection buses

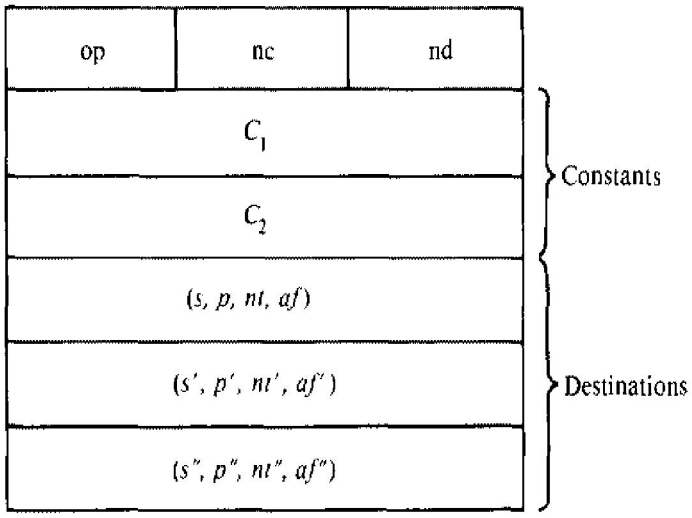
Figure 10.15 The Irvine dataflow computer. (Courtesy of *IEEE Trans. Computers*, Gostelow and Thomas, October 1980.)

2. The ARVIND Machine

- The Arvind machine at MIT is modified from the IRVINE machine
- Instead of choosing token rings, the Arvind machine has chosen to use an NxN Packet switch network for inter PE communications
- The machine consist of N PE's , where each PE is a complete computer with an instruction set, a memory, tag-matching hardware, etc.
- Activities are divided among the PEs according to a mapping from tags to PE numbers.



(*a*) The machine architecture



(*b*) A typical instruction

- General format of instruction
  - op- opcode
  - nc- no. of constants stored in the instruction
  - nd- no. of destinations for a result token

Destination is identified by 4 fields

- (s, p, nt, af)
- s ---> destination address
- p--->input port at the destination instruction
- nt-->no. of tokens needed to enable the dest. instrn
- af-->assignment function to be used in selecting the PE.

The functional structure of each PE is shown below. The input section has a register which if empty can accept a token either from communication system or from the output of same PE. Each activity requires either one or two tokens as indicated by the m field of token. If the activity requires another token , the waiting-matching section is informed. A buffer is provided there to hold these tokens until matching tag is arrived. When the tags of two tokens match , both tokens are moved to the instruction-fetch buffer. Based on the number part of the tag, an instruction from the local program memory is fetched.

The I structure is a special tagged memory for storing array like data structures with constraints on their creation and access.

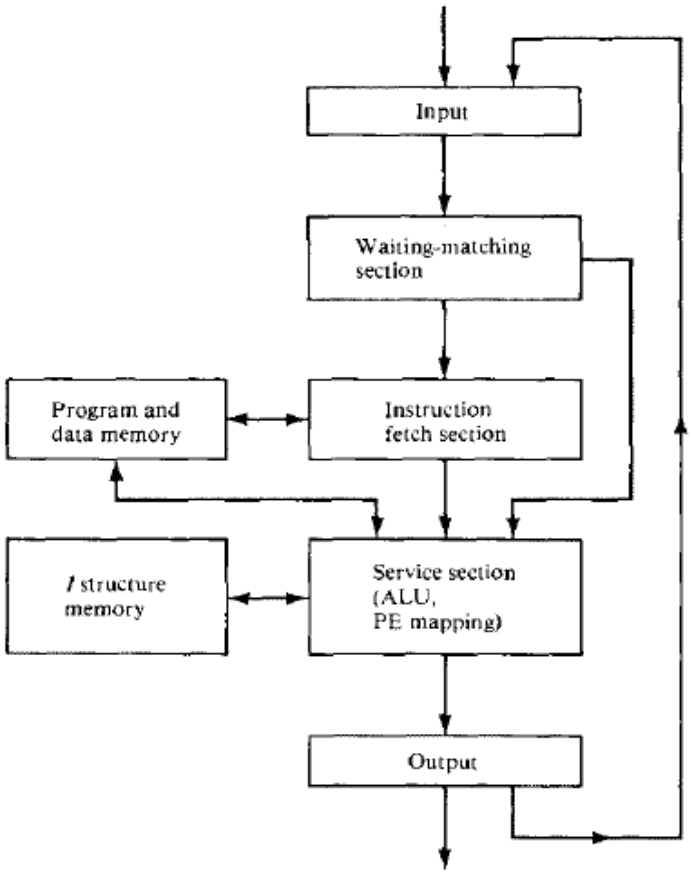


Figure 10.17 The processing element (PE) in Arvind's dataflow machine at MIT.

3. **The EDDY Dataflow Machine**

- Scientific data flow machine in Japan
- **E**xperimental system for **D**ata **D**riven arra**Y**
- **4 X 4 PEs and** two broadcast control unit(BCU)
- Each PE is composed of 2 microprocessors and is interconnected to 8 PEs
- The BCU can load or unload programs and data to and from all PEs in a row or column at the same time.

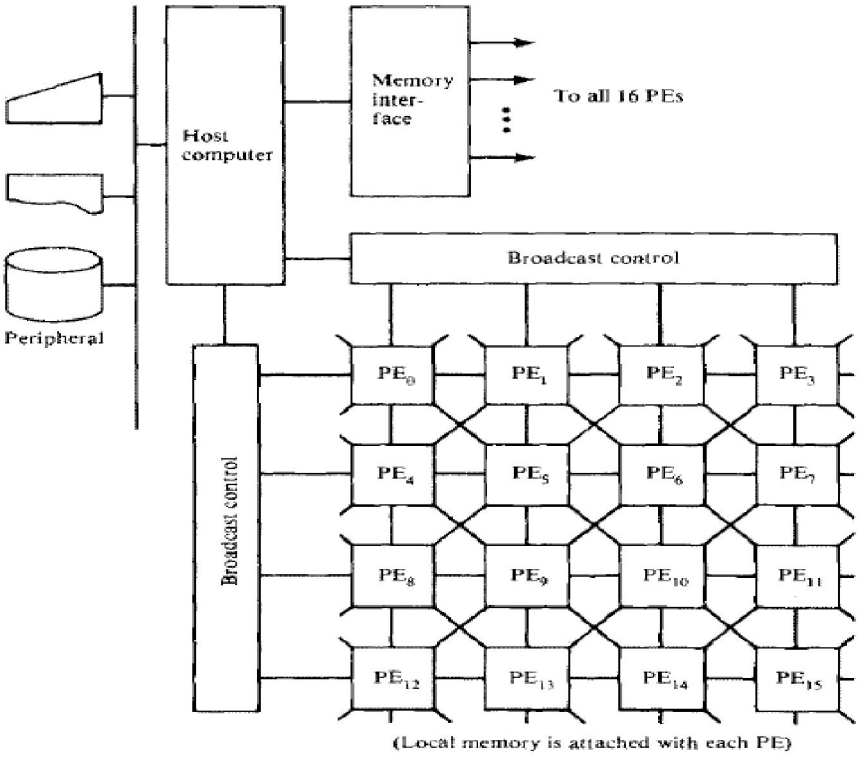


Figure 10.18 The EDDY dataflow machine in Japan. (Courtesy of *Conf. Proc. 10th Annual Symp. Computer Architecture*, Takakashi and Amamiya, June 1983.)

4. **The Manchester Dataflow Machine**

- Ring structure
- Five functional blocks communicate in clockwise direction

1. **Matching unit**
   Groups tokens.

2. **Token package**

   Main unit of information and comprise a data value, label and a destination node pointer

3. **Node store**

When sufficient tokens arrive to fire a node , an appropriate group package finds a destination node description in the node store**.** An executable package(containing operator, operands,label and pointers to destination node) is sent to the processing unit foe execution

4. **Switch** handles external input/output.

5. **token queue** saves excess tokens generated at about the same time.

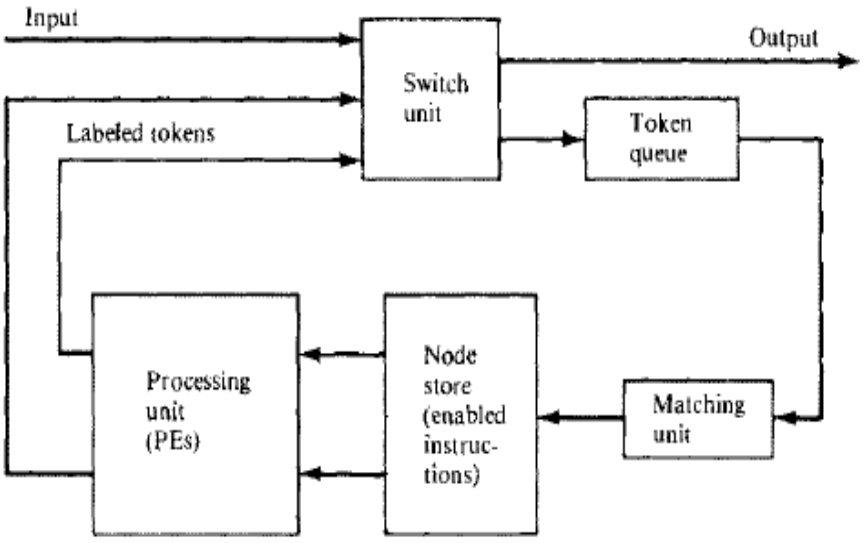


Figure 10.20 The Manchester dataflow computer organization. (Courtesy of AFIPS *Proc. of NCC*, Watson and Gurd, June 1979.)

Data Flow Design Alternatives

Two design alternatives Offers

- higher machine compatibility
- Better utilization of existing software assets

Two design alternatives are

- Dependence-driven approach
- Multilevel event-driven approach

Dependence-driven approach

This approach was independently proposed by Gajski et al. (1982), and by Motooka et al. (1981).

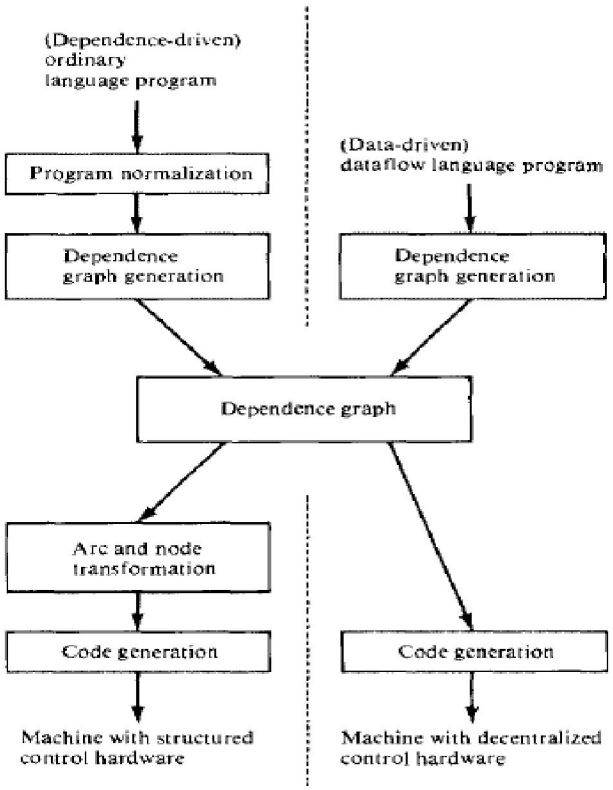
The idea is to raise the level of parallelism to compound-function (procedure) level at run time.

A compound function is a collection of computational tasks that are suitable for parallel processing by multiprocessors: Listed below are six compound functions investigated by the research group at the University of Illinois:

- array operations
- Linear recurrence
- For all loops
- Pipeline loops
- Blocks of assignment statements
- Compound conditional statements

- A program is a dependence graph connecting the compound-function nodes. Dependence-driven refers to the application of data flow principles over multiple compound-function nodes. In a sense, it is procedure driven.

- Instead of using data flow languages, traditional high-level language programs can be used in this dependence driven approach .



- Additional program transformation packages should be developed to convert ordinary language programs to dependence graphs.

  A hardware organization needed for dependence-driven computations is given in figure below.

  Such a dependence-driven machine has a global controller for multiple processor clusters and shared memory and other resources, instead of decentralized control as emphasized in a data-driven machine.

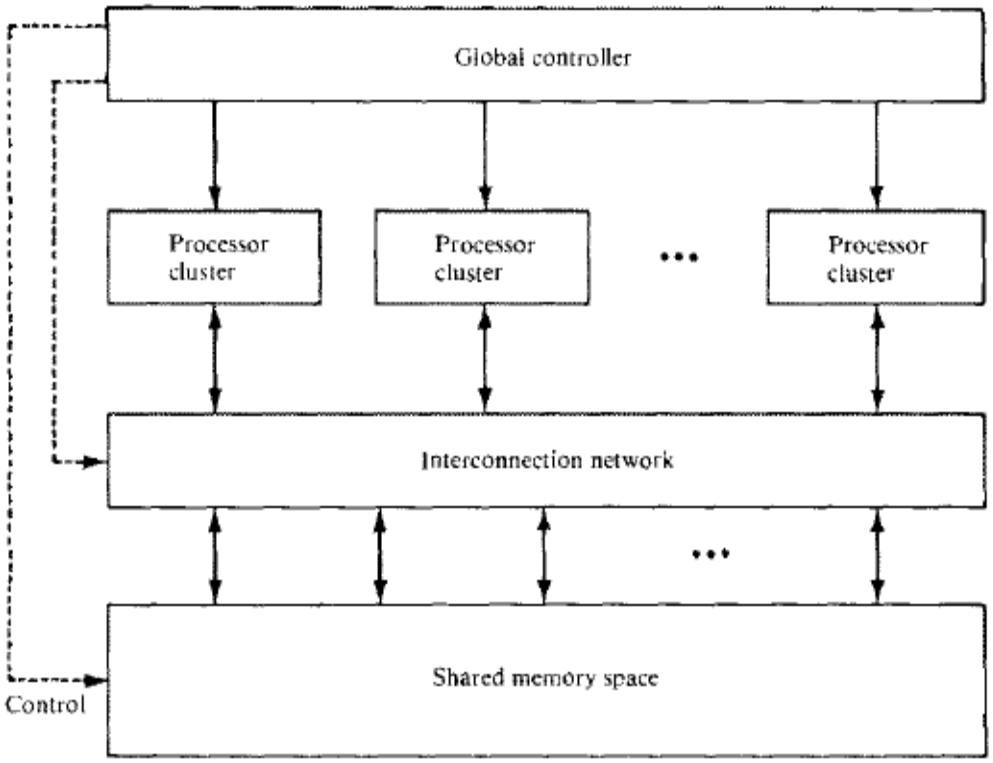
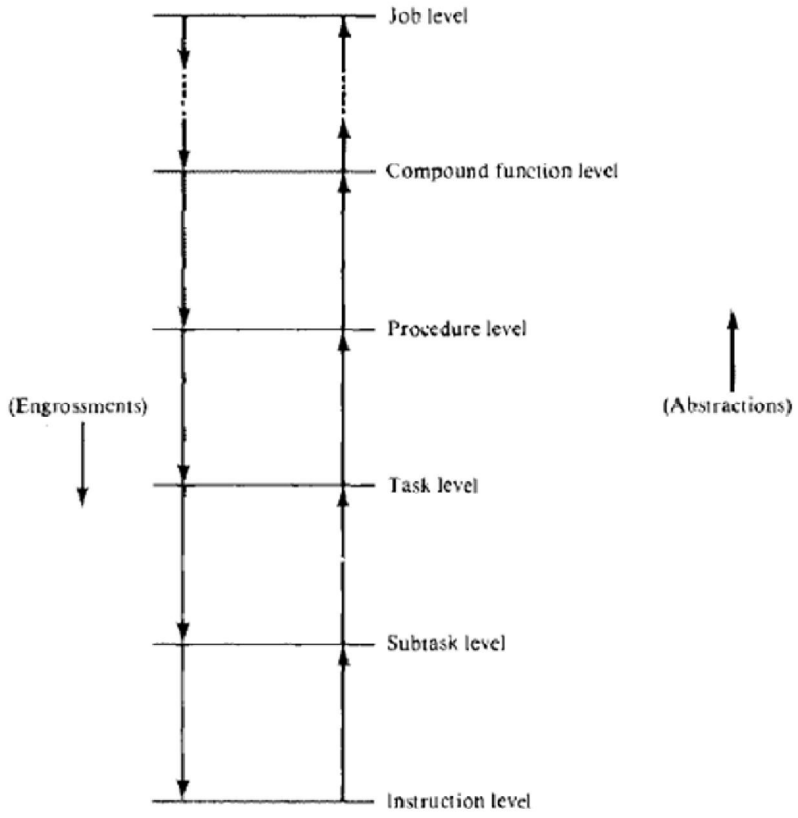


Figure 10.23 The hardware structure suggested for high-level data flow computing, for either the dependence-driven model or the event-driven model.

Multilevel event-driven approach

- The dependence-driven was generalized to an event-driven approach by Hwang and Su (1983c).
- An event is a logical activity which can be defined at the job (program) level, the procedure level, the task level, or at the instruction level after proper abstraction or engrossment as shown in figure.
- Hierarchical scheduling is needed in this event-driven approach.
- A mechanism for program abstraction needs to be developed. Such a mechanism must not require high system overhead. It could be implemented partially at compile time and partially at run time.
  The choice depends on the performance criteria to be used in promoting parallel processing at various levels. Hierarchical scheduling of resources is the most challenging part of research in this approach.



- Heuristic algorithms are needed for scheduling multiple events to the available resources in an event-driven computer.
- Instead of using the first-in, first-out (FIFO) scheduling policy on all enabled activities, as in a data-driven computer, this approach considers the use of priority queues in the event scheduling.
- The priority is determined by pre-run-time estimating of the time-space complexities of all enabled events.