Lecture 31-32: November 4, 2020
Computer Architecture and Organization-II
Biplab K Sikdar

Parallel Computers -V

0.7 Parallel Processing

The parallel processing architectures can be classified into three configurations:

1. Pipelined Processors, 2. Array Processors, 3. Multiprocessor Systems.

Pipelined processor performs overlapped computation to exploit temporal parallelism. Pipelined processors category

1. Base scalar processor,
2. Superscalar processor,
3. Superpipelined processor,
4. Superscalar superpipelined processor.

0.8 Trends

Architectural development targets issues of program flow mechanisms, distributed/centralized processing, computing methods such as cluster/grid etc, modeling computation etc.

The following approaches are considered as the sustainable architectures.

1. Data flow architecture
2. Distributed processing
3. Cluster computing
4. Grid computing
5. Neural network based model
6. DNA computing
7. Optical computing
8. Quantum computing

0.9 Program Flow Mechanisms

Program flow mechanism points to driving factor in computational flow. These are:

1. Control-driven (control flow) architectures

   a) Reduced Instruction Set Computer (RISC)

   b) Complex Instruction Set Computer (CISC)

   c) High-level language architectures

2. Demand-driven (reduction) architectures.
3. Data-driven (data-flow) architectures

0.9.1 Control flow

In control flow m/c, order of program execution is explicitly stated in user program.

Such a machine consumes 60-70% of processor chip area for control unit (CU).

Complexity of CU depends on number of instructions in processor's instruction set.

If number of instructions is increased, the lesser space is left for including different functions. That is, space left for ALU is also less.

RISC is also a control flow machine. It has minimum number of instructions in instruction set and also implements minimum number of operand addressing modes. It enables more chip area for ALU and other functional modules.

Fewer number of instructions in RISC increases the semantic gap between HLL and machine instructions.

Further, RISC demands too complex compiler.

A HLL architecture reduces semantic gap between h/w instruction and the HLL.

0.9.2 Demand-driven (reduction) computer

It was developed by MIT.

Peripheral devices were not suitable for data driven before 2000.

Reduction machines trigger an instruction execution based on demand for its results.

Example: let consider computation of

e = ((a - 100)$\times$ b + ($c\times$ d)).

In demand-driven, demand of value e triggers evaluation of x = (a - 100)$\times$ b and y = $c\times$ d.

Further, it triggers evaluation of z = a - 100.

Each of these evaluated value is returned to the respective demander (that is, z = a - 100 to compute x and x & y to compute e) to get the final solution (e).

In demand-driven, instructions are executed only when their results are required by an other instruction and, therefore, it is also referred to as *lazy evaluation*.

Idea of lazy evaluation machines matches with functional programming concept.

Data-driven or DFM realizes bottom-up approach also called *eager evaluation*. For example, to compute

e = ((a - 100)$\times$ b + ($c\times$ d)),

operations are carried out immediately after all operands are available. It computes

z = a - 100 and y = $c\times$ d

then, x = (a - 100)$\times$ b

then, e = x + y.

Such a machine can also have high potential for parallelism and throughput.

0.10 Data flow processors

Control flow

Conventional von Neumann machines follow control flow (CF) mechanism.

Order of instruction execution in such a machine is explicitly stated in program.

The next executable instruction depends on execution of current instruction. PC stores the address of next executable instruction.

Instruction can be executed in CF m/c even if some of its operands are not ready.

Further, state-of-the-art control flow machines use shared memory for instructions and data. This prevents efficient parallel processing.

Data flow

Instructions in dataflow machine (DFM) are unordered.

An instruction I$_{i+k}$ can be executed before instruction I$_i$ at time t.

Objective is to execute an instruction instruction I$_{i+k}$ as soon as its operands are ready and do not wait for completion of instructions preceeding I$_{i+k}$.

Execution in DFM is driven only by availability of operands - doesn't require PC.

However, a DFM requires special compiler. Figure 54 illustrates computational steps in a DFM for Z = 8 + (x+y) * (x-y).

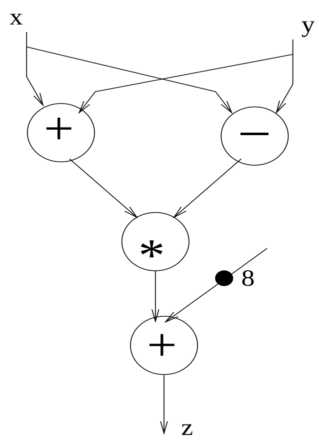

Figure 54: Data flow graph

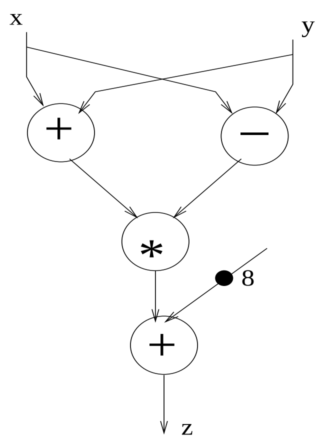

Figure 55: Data flow graph

The above figure represents a DFG (data flow graph), developed in 1960s.

A dataflow program is basically compiled into a DFG. DFG is a directed graph with two components - actors (nodes representing instructions) and arcs.

An arc represents data dependence between two instructions.

During execution of prog, data propagate along arcs in data packets, called tokens.

An arc may be free of tokens at some point of time when operands are not available.

An actor is executed (fired) at time t if its operands (tokens) at arcs are present at t.

Figure 56 shows the generic form of an actor and the firing principle.

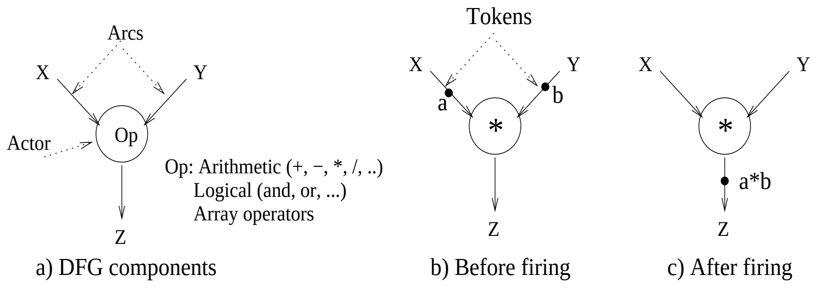

Figure 56: Data flow graph (DFG) components

Figure 57 denotes fork of DFG.

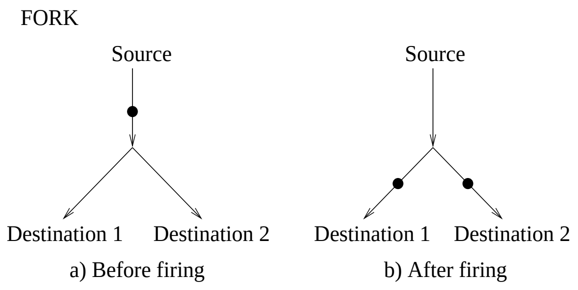

Figure 57: Fork

DFG operation may be arithmetic or logical (defined in Figure 58).

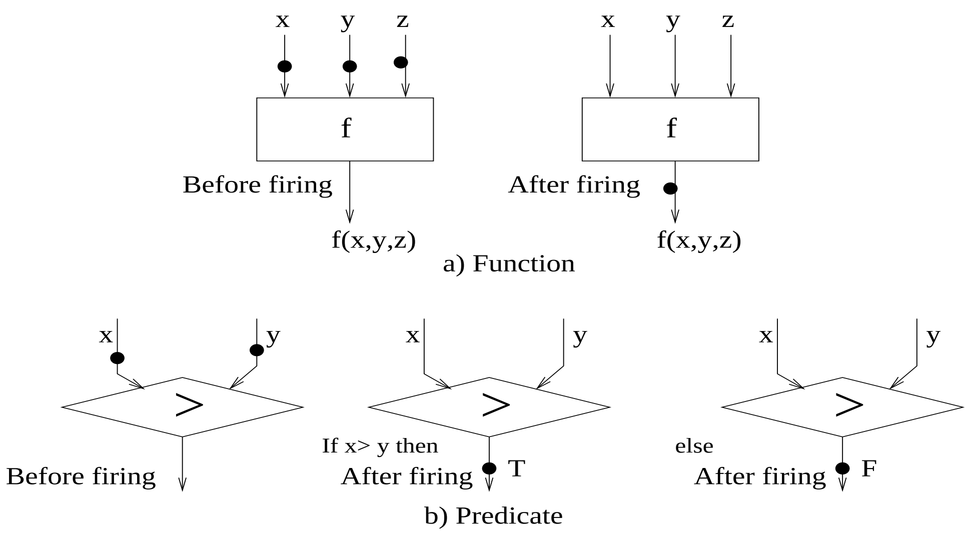

Figure 58: Function and predicate

Gate structures (True/False/Merge gate and switch) are shown in Figure 59.

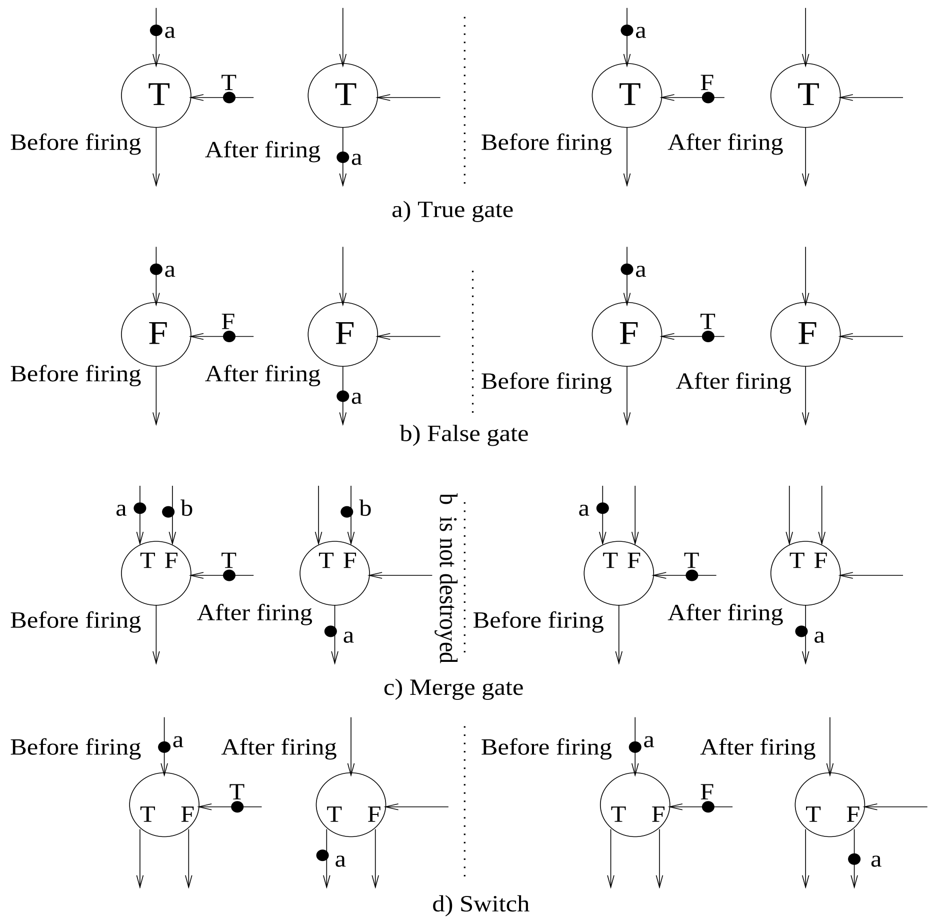

Figure 59: DFG gates

In True gate (Figure 59(a)), if input arc receives token (a) and its true arc receives token 'True' (T), then the gate fires and token a is transferred to output arc.

If true arc receives a 'False'(F) token, then after firing token a is destroyed.

Similarly, in case of False gate (Figure 59(b)), if input arc receives token a and its false arc receives token 'False' (F), gate fires and token a is transferred to output arc. But if false arc receives a 'True' (T) token, after firing token a is destroyed.

Each actor of DFG is represented as activity template referred to as *cell* (Figure 60).

Cell contains operation code, slots for operands, control information and destination address fields. A token consists of a value and a destination address.

Control info: data type of operand, ack required, error control information, ...

Ready flag indicates whether the operand (data i) is available for processing.

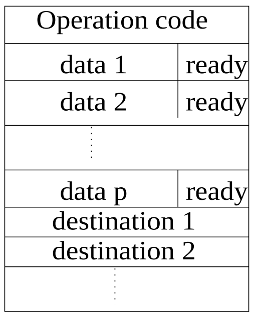

Figure 60: Activity template (cell) of DFG

For example DFG of Figure 54, packaged unit of activity templates is in Figure 61.

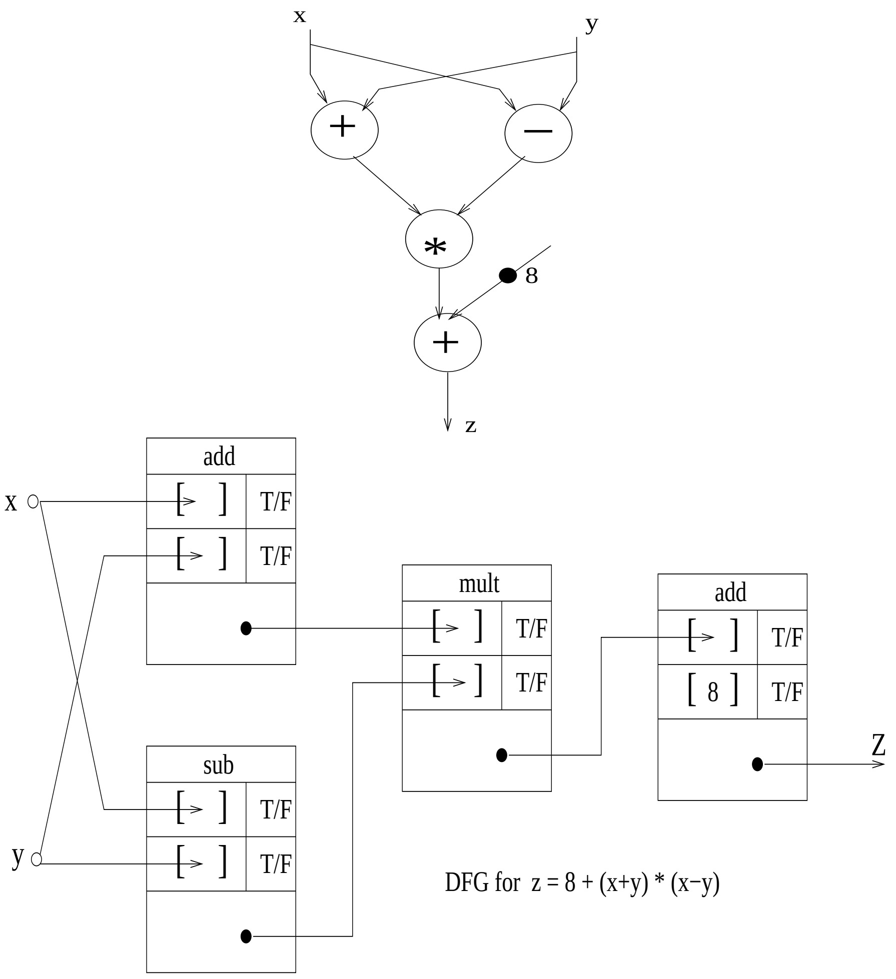

Figure 61: Packaged unit of activity templates

DF computer is suitable for distributed implementation. However, DFMs, in general, are built as an attached processor to a host computer. Host computer handles code translation and input-output functions.

The basic structure of a DF computer is shown in Figure 62.

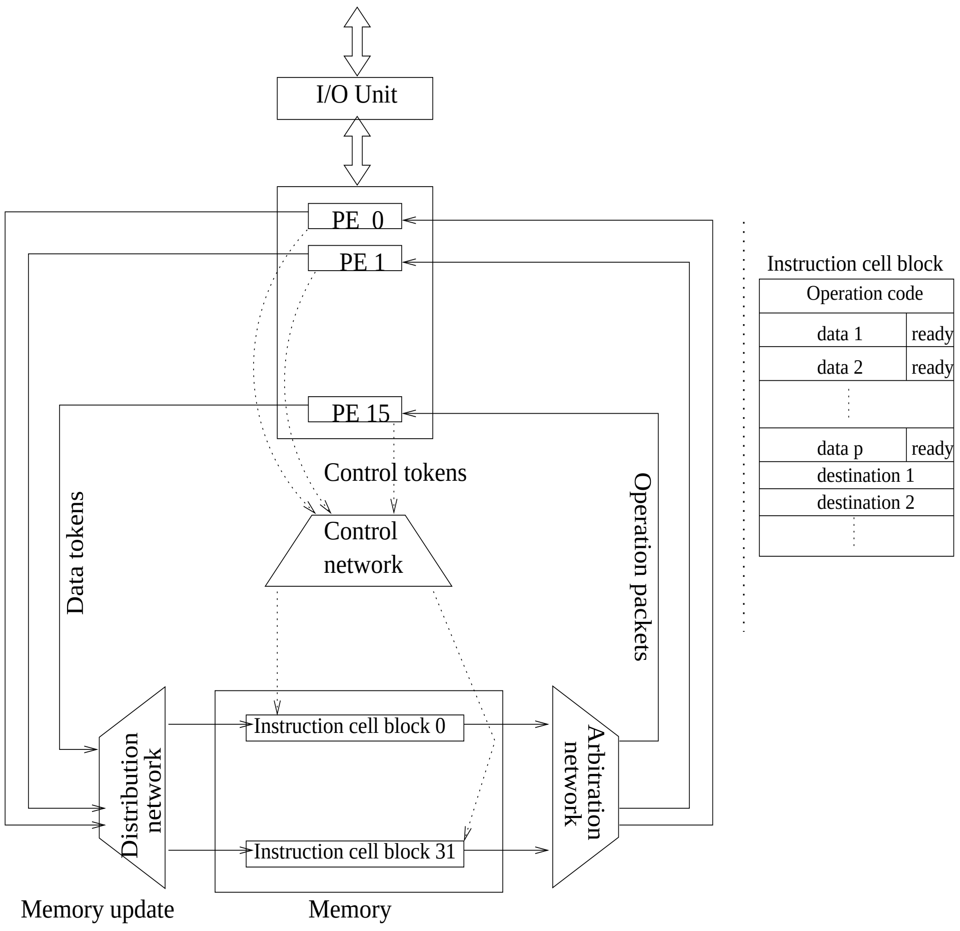

Figure 62: Basic data flow computer architecture MODIFY

Instruction unit (memory) stores the operations that are ready to execute.

Data hazards due to true dependences and control hazards can be avoided by DFM.

The prototype DF computer designed so far is classified as the static dataflow, dynamic dataflow and explicit token store architecture.

0.10.1 Static dataflow

In static model, a node (actor) fires when all its inputs carry token and there is no token at the outputs of the node.

Such a model requires an acknowledgment (Figure 63) from the subsequent nodes which accept the outputs of the node as their inputs.

A subsequent node informs whether it has consumed the input token.

Each node is then receives a control tag, as one of its input, from the subsequent node indicating that the subsequent node has consumed the data on the output arc.

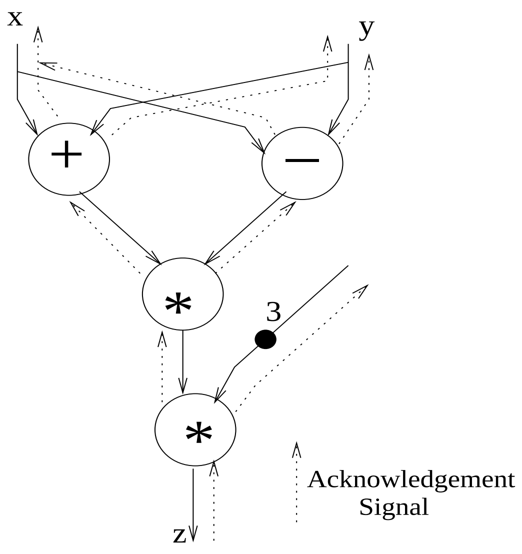

Figure 63: Static data flow graph

0.10.2 Dynamic dataflow

In dynamic dataflow, a node/actor of DFG fires iff all input arcs received the tokens.

Absence of output token in output arc is not mandatory.

Dynamic DFM allows more than one token in an arc - ack is not required.

To associate a token on the arc to appropriate cell block (data set), it needs a tagging scheme called token matching while the processor pool section generates a token.

Processor operates on the operands that have similar tags and the memory update unit uses the tags to update cells belonging to the same data set.

Architecture of such a machine computer is shown in Figure 65.

Cell structures in static and dynamic data model are same.

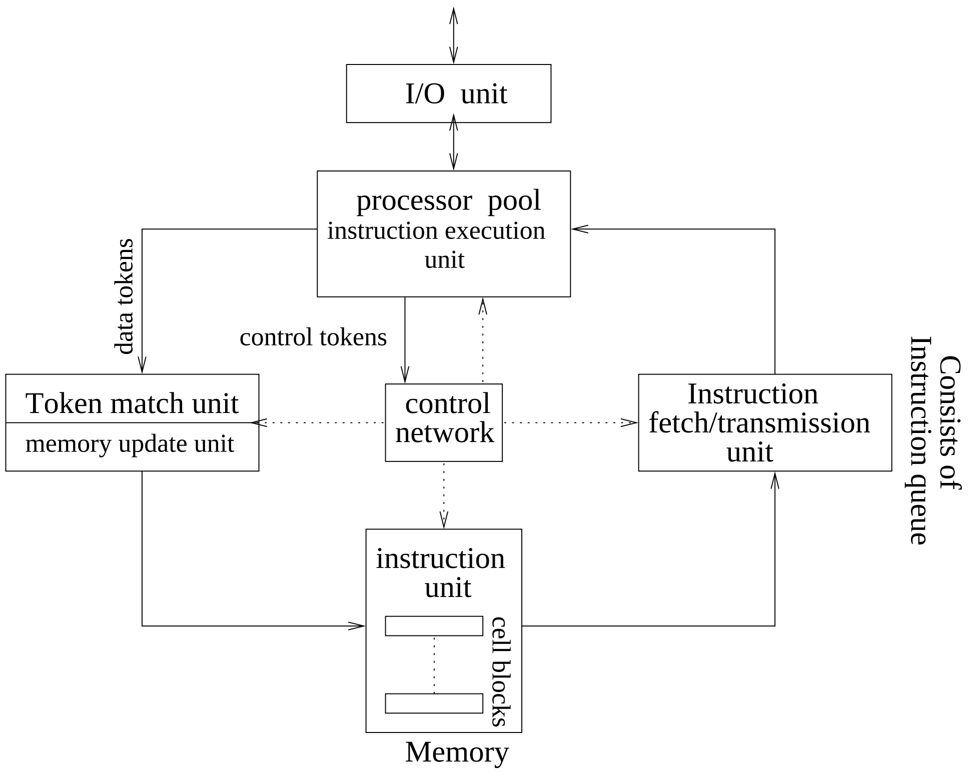

Figure 65: Dynamic data flow computer architecture

Quiz