THE REVIEW OF SOME DATA FLOW COMPUTER ARCHITECTURES

INFORMATICA 1/87

Jurij Šilc and Borut Robič
Jožef Stefan Institute
Department of Computer Science and Informatics
Ljubljana

UDK 681.32.02

The article reviews some selected data flow computer arhitectures. All the architectures are designed for VLSI implementation to provide large throughput, low power consumption, and reduced size and weight. While some are in the phase of simulation and VLSI chip floor-plan contruction the others already exhibit real VLSI implementation.

PREGLED IZBRANIH PODATKOVNO PRETOKOVNIH RAČUNALNIŠKIH ARHITEKTUR - V članku podajava pregled izbranih podatkovno pretokovnih računalniških arhitektur. Nekatere arhitekure so bodisi v fazi simuliranja oziroma izdelovanja logičnih načrtov za VLSI vezja, druge pa so že implementirane v VLSI tehnologiji.

Introduction

In spite of the conceptual break with previous computers, the hardware of the fifth generation computers will be based on VLSI of semiconductor components. Yet it is to be expected, that the hardware of each type of the fifth generation computer will be much more closely tailored to the application area than it is the case at present.

For a number of reasons, one of the most promising architectural models is data flow architecture. It is flexible and extensible, it has the potential for very high data throughputs, and it reflects, at hardware level, the inherent parallelism of the processing. Thus, the potential realm of use includes problem solving & inference machine and intelligent interface machine as it was proposed in the JIPDEC project for fifth generation computer systems.

The presence of some real data flow computers indicates that the state of the art in data flow computing has already passed initial, purely academic discussions.

In the article we review some existing data flow computers from the architectural view point. The presentation is not intended to be thorough. Instead, we concentrate on similarities and differences among the selected architectures.

Manchester data flow computer

The machine organization of the **Manchester data flow computer** [Gurd-85] is a packet communication organization with token matching [Robič-86] and is shown in Fig.1. The basic structure is a ring of four modules connected to a host system via an **I/O switch** module. The modules operate independently in a pipelined fashion with packets transferred at a maximum rate of 4.37 M packets/second. Packets destined for the same instruction are paired together in the **matching unit**. This has limited storage capacity, so that an **overflow unit** is required for programs with large data sets. Paired packets and those destined for unary instructions, fetch the appropriate instruction from the **instruction store**, which contains the machine-code for the executing program. The instruction is forwarded together with its input data to the **processing unit**, where it is executed. Output packets are eventually produced and transmitted back to toward to the matching unit to enable subsequent instructions. The return path passes through the I/O switch module, which connects the system to a host processor, and to the **token queue**, which is FIFO buffer for smoothing out an even rates of generation and consumption of packets.

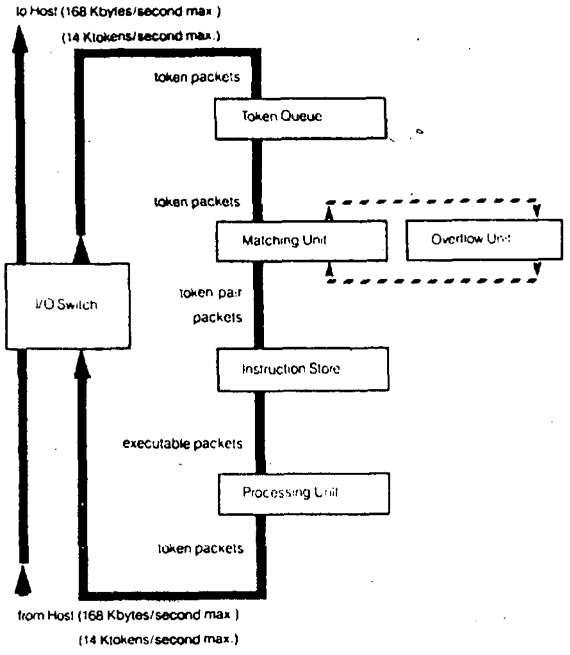


Fig.1. Manchester dataflow system structure.

The I/O switch module gives priority to input from the ring and selects the output route by performing a decode of certain marker bits. It is organized as a simple two-by-two common bus switch.

The token queue comprises three pipeline buffer registers and a circular buffer memory. The later has a capacity of 32K packets, each beeing 96-bit wide.

The matching unit is based on a 1.25 MW pseudoassociative memory with six pipeline registers in the main ring and two buffers interfacing with the overflow unit. The memory is used to store matched packets while awaiting their partners. Its associative operation is achieved by accessing a parallel store using an appropriate hash function. Recall, that packets destined for unary instructions do not need to match with partners; instead, they pass straight through the unit. The overflow unit handles packets that cannot be placed in the parallel hash table becouse they encounter a full hash entry. Overflow packets are stored in linked lists in the overflow unit, which contains a microcoded processor together with data and pointer memories.

The instruction store comprises two pipeline buffer registers, a segment lookup table, and a random-access instruction store to hold the program. The segment field of the incoming packet is used to access the instruction from the store. The instruction is 70 bits wide. The instruction is combined with a destination field and the data field of the incoming packet and is sent to the perocessing unit as a 166-bit executable instruction packet.

The processing unit comprises five pipelined buffer registers, a special purpose preprocessor, and a parallel array of up to 20 homogeneous microcoded function units with local buffer registers and common buses for input and output. A small number of instructions are executed in the preprocessor but the majority are passed into one of the function units via the distribution bus. Each function unit contains a microcoded bit-slice processor with input and output buffering, 51 internal registers, and 4 KW of writable microcode memory. Instructions are execued independently in their allotted function unit, and the output is merged onto the arbitration bus and thence out of the processing unit toward the I/O switch.

In the Manchester architecture a harware hashing scheme is used to simulate the associative memory which turns out to be less exepencive. Unfortunately, this scheme does not produce very good results in terms of waiting time. In order to reduce the waiting time, a multiple matching units scheme is incorporated in the **EXMAN** - EXtended MANchester data flow computer [Patnaik-86].

MIT data flow computer

The **MIT data flow computer** bases on a static concept of data flow architecture [Dennis-80] in which the instructions of machine level program are loaded into specific mermory location in the machine before computation begins, and only one instance of an instruction is active at a time.

Instructions are held in the local memories of the **processing element** PE. Each instruction includes an ooeration code, spaces for operand values, and destination fields that specify where resuls should be sent. Each PE is equipped to recognise which of the instructions it holds have been enabled for execution by arrival of needed operand values. If an enabled instruction calls for a scalar arithmetic operation, the instruction, including its operands, is sent to a **functional unit** FU capable of performing that operation. The **array memory units** AM are provided to hold arrays of data

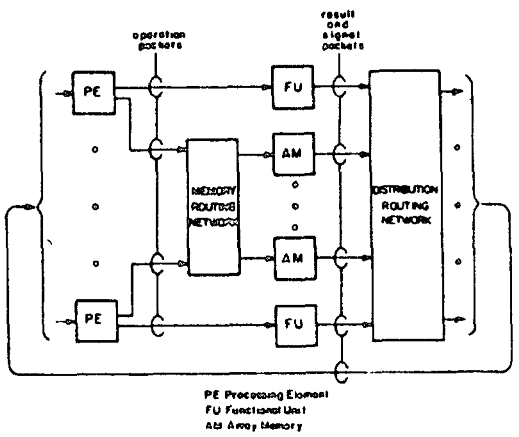


Fig.2. The MIT data flow computer.

making up the data base of computation, and are accessible through the **memory routing network**. Instruction execution in FU or AM yields result packets each of which consists of a data value and a destination field that specifies the target instruction for the result packet. The result packets are sent to PEs that hold the target instruction through the **distribution routing network**. Other instructions, such as those calling for duplicate data values, for boolean operations, and for simple tests, are performed within the PE.

The current status of the MIT data flow project is that hardware for the above computer architecture is under development. For this sake, a data flow engineering model [Dennis-83] consisting of eight processing units coupled by a paket communiation network built of two-by-two routers is designed for emulating the described architecture.

Data flow computer SIGMA-1

SIGMA-1 is a data flow multiprocessor system for scientific computations [Shimada-86]. The configuration of the system is depicted in Fig.3. Four processing elements PE and four structure elements SE are connected by local network and called a group. Groups are connected by global network. The purpose of using hierarchical network is to execute programs efficiently by utilizing principles of locality.

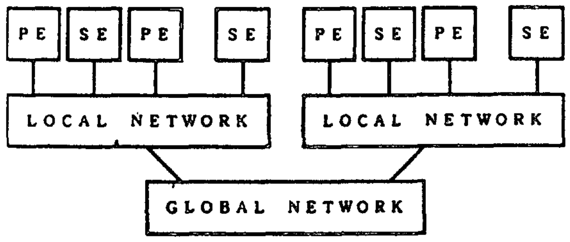


Fig.3. Global configuration of the SIGMA-1.

The **processing element** consists of five units, with the units organized as a two-stage pipeline as shown in Fig.4. PE executes all instructions except those that manipulate structure memory. The **buffer unit** (8 KW of 40 bits) is an interface between the network and the PE. The length of the incoming packet is 88 bits. It is divided into two parts (top

48-bit and bottom 40-bit) and passes through the network as consecutive parts. The most signifficant 8 bits are a network address, next 40 bits are tag, and the remaining 40 bits are data type and value. When there is no waiting packet in the buffer memory and the next units are not dealing with an other packet, the incoming packet bypasses this unit and proceeds to the subsequent units. The **fetch unit** is 16 KW, 40-bit-wide program memory. The link number carried by an incoming packet is used to access the address of an instruction to be fetched. The operation field of the fetched instruction indicates an operation code and is sent to the execution unit. The destination field of the fetched instruction gives addresses of destination instructions (waiting for the result) and is sent to the destination unit. The matching flags from the destination field are sent to the matching unit. The **matching unit** is a 16KW, 80-bit-wide associative memory to find a partner packet of an incoming packet. The matching-flag indicates whether the operation is unary or binary. When it is a unary, the incoming data packet is bypassed to the execution unit. If the instruction is binary operation the incoming packet is stored in the associative memory if it is a first arrived packet of the two operands. Otherwise, the matching unit succeeds to find a partner packet in the matching memory and sends both data of packet pair to the execution unit. The **execution unit** consists of an ALU, a shift unit and a floating point arithmetic unit. The word length is 32 bits. It receives an operation code from the fetch unit and data from matching unit. The **destination unit** makes output packets by combining the destination addresses and results from the execution unit.

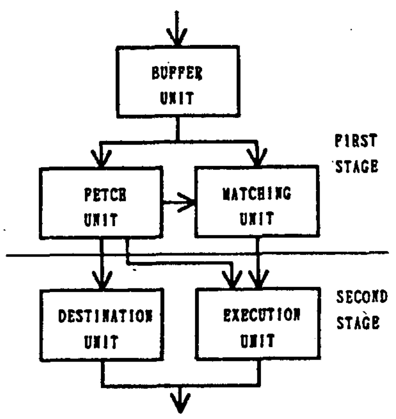


Fig.4. Structure of the processing element.

The **structure element** comprises 64KW, 35-bit-wide memory to store array data and a control unit to manage free memory words and waiting queues. When an array is declared in a program, a contiguous area corresponding to the array size is allocated in the structure memory. Once the word is allocated, the used bit of each word in the area is turned on. Each word has two other special bits. The presence bit means that data has already been written in the word. The waiting bit indicates that at least one read request packet exists in the waiting queue. When data is written in the word the data is sent to the instructions indicated by the waiting packets.

A 10 by 10 crossbar is adopted for a **local network**. This is realized by bit slice chip. The **global network** is organized as a multistage network [Mavriõ-86]. The same crossbar chip is used for the module at each stage of the global network.

Judging from the performance of 1.35 MIPS of the prototype hardware for the benchmark programs, the performance of the next version of a processor with CMOS LSI technology should be about 1.9 MIPS.

µPD7281-based data flow architecture

The **µPD7281** is the first VLSI device on silicon using data flow architecture [NEC-85]. The µPD7281 image pipeline processor is designed to be used as a peripheral processor with a mini- or microcomputer serving as the host. Fig.5 shows a general system configuration example of which four µPD7281s are used connected to the memory in a ring shape with the entire ring interfacing with the host computer via a standard bus.

For the above architecture, NEC is developing a support chip **MAGIC**, Memory Access and General bus Interface Chip. It handles all packet flow between the µPD7281s, the image memory, and the host processor.

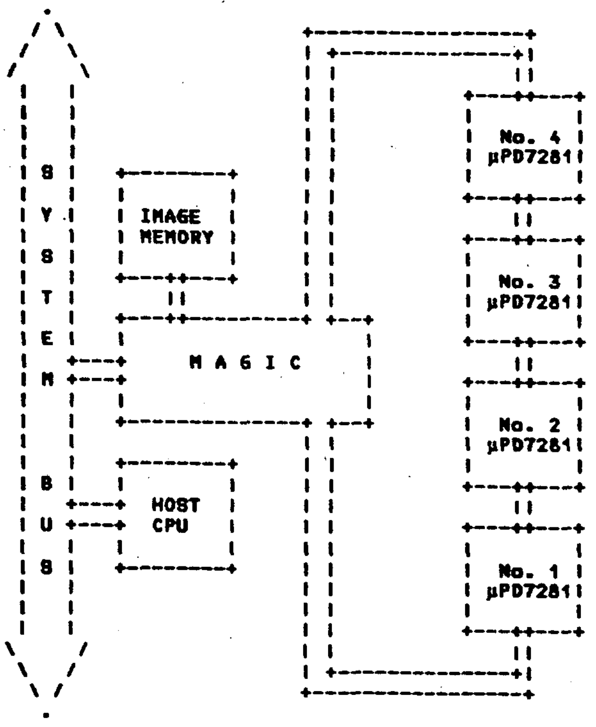


Fig.5. µPD7281-based data flow architecture.

The µPD7281 uses an internal circular pipeline and the powerful instruction set [Silc-86] to allow high end immage processing. A data flow architecture allows the processor to maximize efficiency in a variety of multiprocessing applications. As shown in the block diagram in Fig.6, the µPD7281 is formed by ten functional blocks: the **input controller** IC, the **link table** LT, the **function table** FT, the **address generator and flow controller** AG&FC, the **data memory** DM, the **queue** Q, the **processing unit** PU, the **output queue** OQ, the **output controller** OC, and the **refresh controller** RC.

Before any processing occurs, the host processor down-loads the object code into the LT and FT by using specially formated input packets. The contents of the LT and FT are closely related to a data flow graph. The arcs represent the entries in the LT while the nodes represent the entries in the FT.

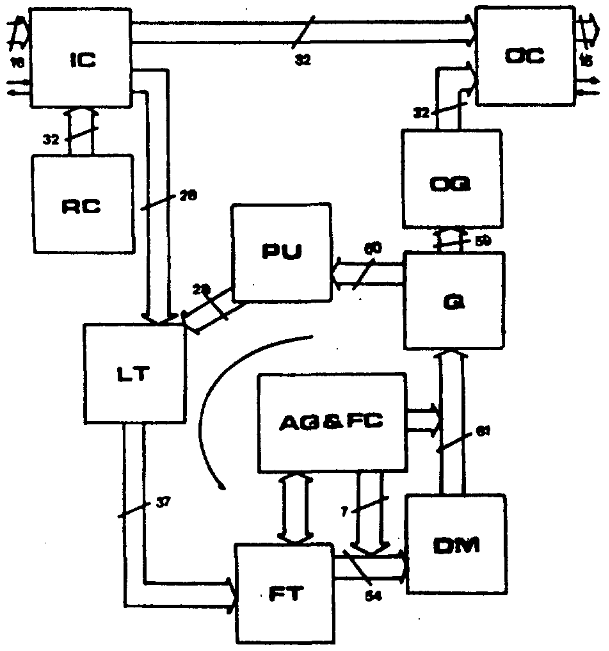


Fig.6. Block diagram of the µPD7281.

When a data packet enters the µPD7281, it fetches from the LT the address of the instruction in FT, waiting for the incoming data. After the destination instruction has been fetched, the AGFC unit determines whether the instruction is unary or binary. If it is unary, the operation packet, consisting of the instruction and the data is composed and sent via Q to the PU. If it is binary, the AGFC stores the incoming data to the DM if it is the first arrived operand for the instruction. Otherwise, it fetches the first operand from the DM and sends it together with the incoming packet and the instruction to PU via Q. The result packet from PU can either be sent out of the µPD7281 (via LT, Q, OQ, and OC) or can be used for further execution of the program graph in the same processor.

The applications of the µPD7281-based data flow architecture include digital image restoration, data compression, and enhancement, pattern recognition, radar and sonar processing, FFTs, digital filtering, speech processing, and numeric proessing.

DFSP - a data flow signal processor architecture

A block diagram of the DFSP architecture [Hartimo-86] is shown in Fig.7. A bank of processing elements constitutes the **execution unit**, which performs the actual digital signal processing computations and I/O operations. Other parts of the architecture form a **control section**, which is essentially a data flow instruction execution pipeline. In orther to increase communication bandwidth, data transfers are separated physically from execution control using a double bus architecture. Signal data is transferred via the shaded buses of the figure. The unshaded buses are used for operation and results packets, which do not contain operand and result values, respectively.

A host computer is required to load the application programs of the DFSP. Programs are coded as separate high level operations, which are copied into the **local memories** of the **processing elements** (PEs). The PEs are allowed to be functionally nonidentical in order to capitalize the existing high speed architectures for fixed signal processing algorithms. Frequently used operations may be executed in dedicated PEs having the appropriate hardware structure. The I/O functions take place in special PEs called I/O processors. This is convenient in signal processing applications, becouse signal sources and sinks tend to introduce specialized requirements.

The control section schedules instructions for the PEs using fixed-format messages. Recall, that initially all the information about the data flow graph of the application program resides in the local memories of the PEs. Each result packet carries the neccessary part of this information to the control section where it is temporarily stored in the activity store until the destination operation may be scheduled for the execution. The execution is performed by sending an operation packet to one of the PEs.

The **activity store** contains the activity templates of those operations which have received at least one of the operands, but which are not scheduled for the execution. Conceptually, the activity store contains a representation of the active part of the data flow graph.

The contains of the result packet are used by the **update unit** for locating the activity template (of the destination operation). It also contains the the address of a block in the

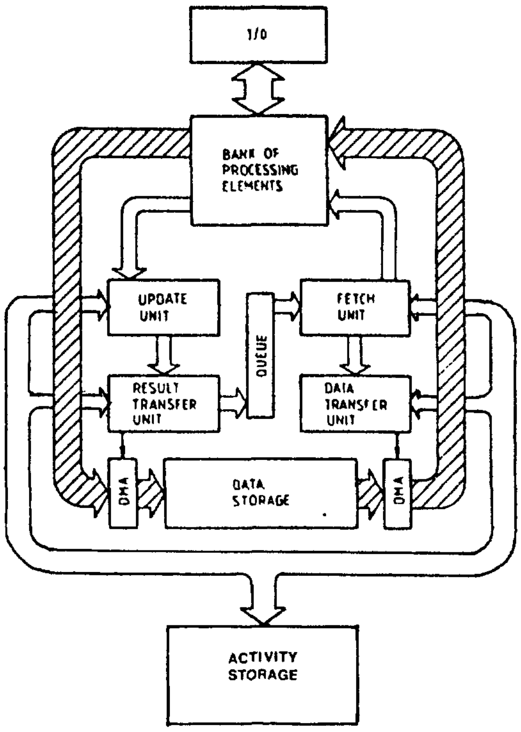


Fig.7. Block diagram of the DFSP architecture.

data storage, where the value of the operand has been stored. If the operand is the first one, the update unit creates a new activity template and stores the result packet into it. Otherwise, the result packet is stored in the located activity template. Finally, it puts a transfer command into the result queue.

After the **result transfer unit** detects the command from the queue it transfers the updated activity template. Each activity template contains a TRIGGER field whose value indicates the number of the arrived operands. The result transfer unit decrements the TRIGGER field of the destination template and checks for the resulting value. If TRIGGER equals zero the template address is put into the queue, since the operation is exectable.

After the **fetch unit** gets a template adress from the queue, sends the operation packet to an idle PE, and puts a data transfer command into the data queue.

Finally, the **data transfer unit** initiates the transfer of the operand data block from the data store to the PE.

The performance of the DFSP architecture was evaluated on a deterministic discrete event simulator. The update unit has been the major bottleneck in the simulated control section. However, considerably uniform utilization of the (four) processors has generally been achieved. Also, a VLSI implementation of the control section is under development.

HDFM - Huges data flow multiprocessor

The **HDFM** is a proposed high performance, foult tolerant, high level language programmable processor targeted for embedded signal and data processing applications [Gaudiot-85]. The HDFM consists of one to hundreds identical **processing elements** PEs connected by global packet-switching network. This network is a three-dimensional bused-cube network as shawn in Fig.8. Packet transmission proceeds via a store-and-forward protocol, which allows any PE to transfer data to any other PE.

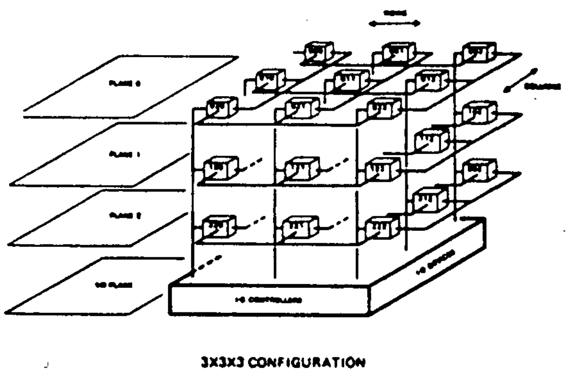


Fig.8. The HDFM cubic bus interconnetion network.

Each PE can execute the instruction set and perform the data flow sequencing and addressing. Each PE has its own local memory for both program and data storage. There is no global memory. The program, which consists of data flow graph nodes, is allocated to the local memories of the PEs at compile time. When nodes are implemented in an architecture they are called templates. A template consists of an opcode, slots for operands and destination pointers, which indicate the nodes to which the results of the operation should be sent. Each PE has a unique 9 bit address corresponding to its position in the cube (plane, column, row). This allows up to 8 PEs per bus for a maximum configuration of 512 PEs. Within each PE are multiple templates, each of which has unique template address. To implement the data flow model, the results of one template are sent to another in the form of packets. Each packet consists of a type field, a destination address, and data. The destination address indicates the PE address and the template address of the template to which the data is to be sent. Packets travelling from one PE

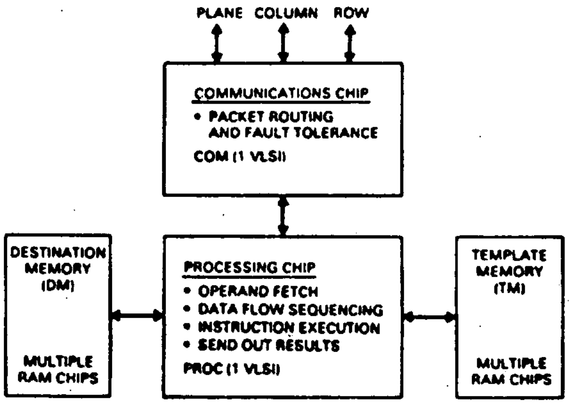


Fig.9. The structure of the PE.

to another allways take the same path. This principle, called single path routing, is requried to preserve the order of packets in time-ordered data groups such as strings. Each PE consists of four parts: **communication chip** COM, **processing chip** PROC, **destination memory** DM, and **template memory** TM. The COM receives packets from the routing network and either forwards them onto other PEs or sends them to its attached PROC. When the PROC receives a packet, it determinates whether the packet has enabled a template to fire. If so, the template opcode and the operands stored in the TM are fetched, combined with the incoming packet, which enabled the template to fire, and sent to ALU. Simultaneously, the destination addresses to which the result should be sent are fetched. The ALU performs the operation and the results are matched with their destination address to form packets. The packets are sent either back to the same PE or out into the routing network. If the template is not ready to fire, the arriving packet is stored in the TM. Since a template result may need to be sent to multiple destinations, there is additional destination overflow storage in the DM to accomodate lists of destinations for a node.

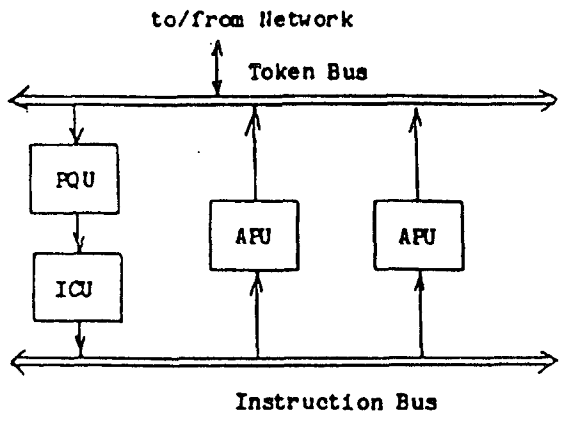
Simulation results demonstrate high-performance operation with high-level language programmability. For example, the results of the deterministic simulation of the machine show that a 64 processing element machine may provide real throughput of 64 MIPS.

PIM-D - the dataflow-based parallel inference machine

The research and development of the parallel inference machine includes the data flow mechanism to rapidly execute inference operations. The data flow model has also similarity to the logic programming languages. Execution of logic programs is performed in a goal driven manner; a clause in the program is initiated when a goal is given and returns the results to the goal. The logic programs are compiled into data flow graphs [Bic-84]. **PIM-D** is an example arhitecture to support parallel version KL1 of the kernal language for ICOT's fifth generation computers [Ito-86].

PIM-D is, similarly to SIGMA-1, constructed from multiple **processing elements** PEs and **structure memories** SMs interconnected by a network.

Each PE has several stages. The packets transferred between these stages include result packets and executable instruction packets. A result packet consists of three fields: identifier, destination and the data. The identifier



PQU: Packet Queue Unit
ICU: Instruction Control Unit
APU: Atomic Processing Unit

Fig.10. Configuration of the processing element.

specifies the invoked procedure instance to which the result packet belongs. The destination specifies the destination instruction address of the result packet. It also specifies whether the destined instruction receives a single operand or two operands. The data field contains the operand data to be sent to the instructuion. Fig.10 depicts the configuration of each PE. **Packet queue unit** PQU is a FIFO queue memory to store the result packets from the token bus. **Instruction control unit** ICU receives the result packets from PQU and checks if the destination instructions are executable or not. An instruction is executable if it receives a single operand, or if the partner operand is already in the **operand memory** in the ICU when it receives two operands. In the later case, the ICU searches in its operand memory whether the partner operand exists or not. If it does, the partner is removed from the memory; otherwise, the result packet is stored in the operand memory. This searching is performed associatively by hardware hash using the identifier and the destination address as the key field. If the instruction is executable, the ICU fetches the instruction code in its **instruction memory** and constructs an executable instruction packet and sends the packet to the next stage, one of **atomic processing units** APUs via the **instruction bus**. The APU interprets the instruction packets and sends result packets to the PQU in its PE or other PEs, or sends structure access comand packets to SMs via the token bus. The SMs are responsible for the structure access commands, perform structure manipulation operations, and return results to the destination specified by the commands.

Actual implementation of the experimental machine is currently beeing developed. The machine includes 8 PEs, 7 SMs, and one host computer used to monitor or debug the system. The APUs and SMs are implemented as microprogram control units using bit-slice microprocessors or special hardware to recognize the data tag. The ICUs are also microprogram controlled to implement hashing hardware. A software simulator for OR-parallel and Concurrent Prolog was developed. Performance evaluation results from the software simulator show that about one million head unifications per second can be achieved by exploiting parallelism.

Conclusions

Since about 1970 there has been a growing and widespread research interest in data flow computer architecture. This interest has culminated in many designs for data-driven computer systems, several of which have been or are in the process of being implemented in hardware.

The major long-term interest in dataflow techniques will be in the construction and performance of multiprocessor systems. It is particulary important to know how dataflow systems should be designed for implementation in VLSI, and to be certain that effictive software techniques are avaiable for utilizing the hardware.

References

[Bic-84] L.Bic, A data-driven model for parallel interpretation of logic programs, Proc. Int'l Conf. Fifth Gen. Comp. Systems, ICOT (1984) 517-523.

[Dennis-80] J.B.Dennis, Data flow supercomputers, Computer **13** (11) (1980) 48-56.

[Dennis-83] J.B.Dennis, W.Y.-P.Lim, and W.B.Ackerman, The MIT data flow engineering model, in: R.E.A.Mason, Ed. Information Processing 83, (Elsevier Science Publishers B.V., North-Holland, 1983) 553-560.

[Gaudiot-85] J.-L.Gaudiot, R.W.Vedder, G.K.Tucker, D.Finn, and M.L.Campbell, A distributed VLSI architecture for efficient signal and data processing, IEEE Trans. Comp. **34** (12) (1985) 1072-1087.

[Gurd-85] J.R.Gurd, C.C.Kirkham, and I.Watson, The Manchester prototype dataflow computer, Comm. ACM **28** (1) (1985) 34-52.

[Hartimo-86] I.Hartimo, K.Kronlöf, O.Simula, and J.Skyttä, DFSP: A data flow signal processor, IEEE Trans. Comp. **35** (1) (1986) 23-33.

[Ito-86] N.Ito, M.Kishi, E.Kuno, and K.Rokusawa, The dataflow-based parallel inference machine to support two basic languages in KL1, in: J.V.Woods, Ed. Fifth Generation Computer Architectures, (Elsevier Science Publishers B.V., North-Holland, 1986) 123-145.

[NEC-85] NEC Electronics, Image pipeline processor μPD72810, Product Description (1985).

[Mavrič-86] S.Mavrič, B.Mihovilovič, and P.Kolbezen, The interconnection network in a multiprocessor system, Informatica **10** (4) (1986) 44-50 (in Slovene).

[Patnaik-86] L.M.Patnaik, R.Govindarajan, and N.S.Ramadoss, Design and performance evaluation of EXMAN: an EXtended MANchester data flow computer, IEEE Trans. Comp. **35** (3) (1986) 229-244.

[Robič-86] B.Robič and J.Šilc, Classification of new generation computer architectures, Informatica **10** (4) (1986) 18-32 (in Slovene).

[Shimada-86] T.Shimada, K.Hiraki, K.Nishida, and S.Sekiguchi, Evaluation of prototype data flow processor of the SIGMA-1 for scientific computation, Proc. 13th Int'l Symp. Comp. Arch., IEEE (1986) 226-234.

[Šilc-86] J.Šilc and B.Robič, Data flow architecture based processor, Informatica **10** (4) (1986) 74-80 (in Slovene).