English translation of the paper:
D.I.Batalov, V. V. Vlassov, V.N. Emelin, S.V.Ivanov, and B.A. Kraynikov, Simulation of a Vector-Flow Computer System. In: Izvestiya LETI (Proceedings of Leningrad Electrotechn.Inst.), St.Petersburg, Vol. 423, 1990, pp. 60-65. (in Russian).

UDK 681.325.025

D.I.BATALOV, V.V. VLASOV, V.N. EMELIN, S.V. IVANOV, A.V. KRAYNIKOV

SIMULATION OF A VECTOR-FLOW COMPUTER SYSTEM

Today, one of efficient approaches in development of high performance computer systems is to combine the dataflow model of execution, the RISC-architecture and pipelined hardware.

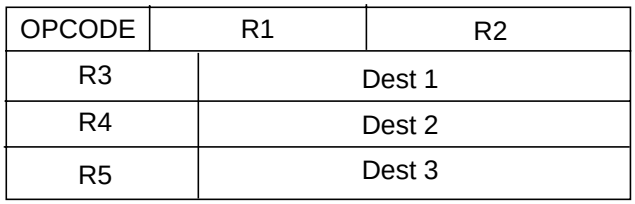
A vector-flow computer system (VPVS) presented in this paper, is based on the following principles [1] :

1. A two-level computational model;
2. Dataflow execution on the instruction level;
3. RISC architecture;
4. Functionally distributed specialized hardware;
5. Pipelined execution of instructions.

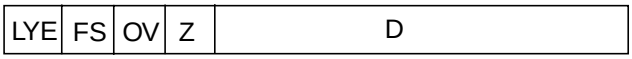
In the two-level computational model, a dynamic recursive parallel computation model[1] is realized on the level of procedures (programming modules), while dataflow computation model is used on the level of instructions [2] .

In fact, an instruction set for VPVS has been inspired by the CRAY vector processor architecture. To design the VPVS instruction, we have looked at and analyzed programming and execution of a collection of the "Livermore loops" [3] with the results that the instruction set was extended with additional instructions such as constant set/reset, conditional and unconditional distribution of tokens to multiple destinations.

Figure 1 shows the format used by a VPVS instruction and the data format.



(a)



(b)

Fig.1.

1. The model was developed at the Computer Engineering Department of LETI.

Instructions (Fig.1.a) have at most two operands specified by R1 and R2, and up to three destinations for the data result and result tokens. Each operand field selects one of the 24 scalar registers or one of the six 32-element vector registers. The instruction set also contains special vector load and store memory instructions.

As data flow is followed by token flow, each of the three possible destinations is specified by two components: the register (e.g. R3) as the destination of the result of the operation, and an address that defines the destination instruction that will receive the result token (e.g. Dest1). The destination address is used as a token that is passed to the processing element where that destination instruction is located.

In order to reduce the instruction set, data are tagged (Fig.1.b). Each scalar and a vector element value has four 1-bit tags associated with it: zero (Z), overflow (OV), representation, i.e. fixed- or floating-point (FS), and last vector element (LVE).

The structure of the VPVS vector-flow processing node is presented in Fig.2.

Each vector-flow processing node consists of four processing elements (PE). Each PE contains operating unit (OU), instruction processor (IP), and four banks of local memory (LM). Each memory bank is associated with a processing element and is logically split into four segments to store vectors (V), scalars (S), instructions and ready tags (I), and token queues (Q).

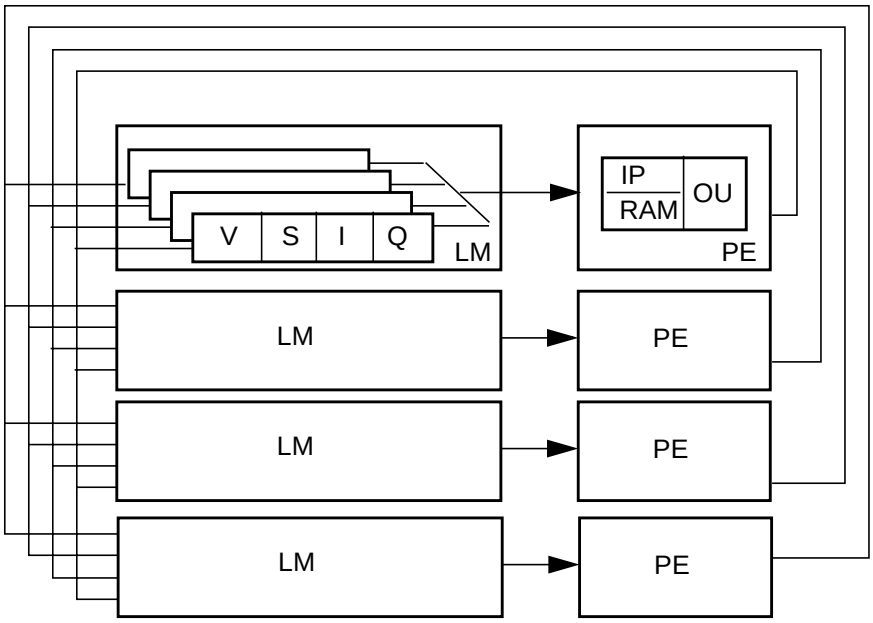


Fig.2.

In order to simulate the vector-flow processing node and to verify a dataflow run-time system and dataflow applications, we have developed a simulator DFSL realized in FORTAN-4. The sim-

ulator is based on the SMPL[1] (Simulation and Modeling Programming Language) [4] , it is a simple discrete-event simulation package. The package includes a library of routines that operate on event lists, facilities and queues. The package has been modified and extended with routines that provide a convenient user interface to work in the SVM (System of Virtual Machines) on ES EVM.

An input file with simulation parameters and a dataflow application typically contains the following sections:

- Simulation parameters of the vector-flow processing node;
- Code segments of the data-flow application (actors);
- The initial marking segment that specifies initial marking of a dataflow scheme for the given application;
- Data segments to be loaded to RAM of processing elements.

Simulation parameters include the size of instruction memory and the size of RAM of a processing element; the maximum number of procedures (actors) that can be performed concurrently; the number of processing elements; the number of scalar registers and the number of vector registers in LM; the maximum length of vectors; the maximum length of token queues; simulation time.

A dataflow application is specified in the form of code segments (actors) to be loaded to the instruction memory of a processing element. A code segment is written in autocode (in a textual form). An autocode instruction specifies operation code, addresses of operands and result destinations. For example, vector addition instruction located in the I-th processing element can be specified as

```
K     ADD  V1   V2   V3   J    L
```

where `K` is instruction address; `ADD` is opcode; `V1` and `V2` are operands; `V3`, `J`, `L` are result destinations, where the pair (`J`, `L`) forms a result token that specifies a PE number and the address of destination instruction in the `J`-th PE, respectively; the result vector register `V3` is located in the J-th local memory bank (LM) of the `I`-th PE.

A COPY instruction copies a value in a first scalar register to three destination registers if a value (acknowledgment) has arrived to a second operand S5:

```
K     COPY S1   S5   S10  J    P
      S2   L    M    S7   N    O
```

where `S1` is a source operand; `S5` selects a scalar register for acknowledgment; (`S10`, `J`, `P`), (`S2`, `L`, `M`), and (`S7`, `N`, `O`) are first, second and third destinations. Pairs (`J`, `P`), (`L`, `M`), and (`N`, `O`) form result tokens.

Logically the simulator consists of three parts: a monitor, a processing element model and a simulation subsystem.

---

1. A.k.a. SIMPL. - *Note of Vlad Vlassov*

The monitor manages input of simulation directives and initiates execution of directives. Major simulation directives are

1. BEG -- set initial state of the simulator.
2. AUT -- start simulation in auto mode.
   The following reasons cause simulation in auto mode to stop:
   (a) execution of “end of module” instruction;
   (b) execution of a simulation directive read from the input file;
   (c) break at a specified breakpoint (break if specified condition is true);
   (d) an exception that occurs during simulation (token collision, incompatible data types, incompatible vector dimensions, etc.
   On stop, breakpoint, or exception the monitor is invoked.
3. STEP -- step by step simulation until an event occurs. The step-by-step mode allows observing simulation in event order.
4. OUT -- output a simulation report.
5. ACT -- set a breakpoint at a specified instruction (actor) on a specified event;
6. T -- set a time to break. When the simulation time reaches the specified time, the simulation continues in STEP mode.
7. LC -- set a control (monitor) level. The amount of information on events, collected and displayed by DFSL, depends on the specified monitor level.
8. DM -- dump (inspect) the state of RAM (data memory) of a specified processing element.
9. MP -- dump (inspect) the state of the ready tag memory (I).
10. S -- dump (inspect) the scalar registers in the local memory (LM) of a specified PE.
11. V -- dump (inspect) the vector registers in the local memory (LM) of a specified PE.
12. END -- end of simulation.

A model of a processing element contains operating unit, that is considered as a facility (resource) in SMPL, and a token queue. The operating unit model includes an interpreter, that emulates execution of ready instructions.

The simulation subsystem represents a event-driven model of an instruction processor. It includes a module that tests instruction ready tags and a module that manages transfer of data and result tokens to destinations. In the simulation subsystem, events (and appropriate actions) of following three types are planned and appropriate actions are taken:

1. Check a input token queue for incoming tokens and set of ready tags for destination instructions. If a destination instruction is ready (it has collected all required tokens), an event of the 2-nd type is planned for that instruction. If the destination instruction is not ready, an event of the 1-st type is planned for that instruction.
2. Execution of a ready instruction. If OU is available, then the instruction is scheduled, executed, and an event of the 3-rd type is planned for this instruction. If OU is not available, an event of 2-nd type is planned for the instruction.
3. Distribution of data and result tokens to destinations. When the distribution is done, an event of the 1-st type is planned.

A main event attribute is event completion time. Execution time of an instruction depends on its opcode and types (vector/scalar) of operands. Result distribution time mainly depends on length of data (length of vector).

A simulation report typically contains statistics on token queues, operating units and on the instruction set, timing estimates, and utilization of the vector-flow processing node (VFPN).

References

[1] V.V. Vlasov, A.I. Vodyakho, S.I. Ivanov, et al. Vector-flow RISC processors. In Proc of the Scient. Conf. of Electrical Faculty VST, Technical University of Kosice, Aug 1989, Kosice, Slovakia, pp. 124-129.

[2] A.I. Vodyakho, V.P. Emelin, V.U. Plusnin, D.V. Puzankov. Vector-flow super-computers, *University News. Engineering*, No. 10, 1987, pp. 35-40.

[3] A.I. Vodyakho, V.P. Emelin, S.I. Ivanov, et al. Realization of vector-flow systems, *University News. Engineering*, No. 9, 1989, pp. 27-33.

[4] ???