A DATAFLOW SYSTEM FOR SIMULATION OF DYNAMIC OBJECTS[1]

Vladimir V. Vlasov, Alexander V. Kraynikov, Boris A. Kurdikov, and Dmitry V. Puzankov
Computer Engineering Department, Leningrad Institute of Electrical Engineering, Leningrad, USSR
May 1991

Abstract. This paper presents a dataflow machine VSPD-1, an application specific system, developed and implemented at the Computer Engineering Department, Institute of Electrical Engineering (LETI), Leningrad, USSR. An implemented VSPD-1 prototype consists of a host computer with the PDP-11 architecture and five Qbus-based processing nodes connected via bus adapters.

In VSPD-1, dataflow computation is realized at the level of labeled Pascal code fragments called *actors*. Data flow is followed by token flow. A *token* is a signal which notifies that a datum has arrived at a destination actor. The token is generated by a source actor after the actor sends data to the destination. The amount of token traffic can not be less that the amount of data flow. An actor is ready for execution when it collects tokens (data) on all its inputs. Data and token flows are controlled by a distributed run-time system (called monitor) with hardware support.

An application for VSPD-1 must be written in a specific dataflow style where actors are programmed in Pascal with macros in Macro-11 assembler. The macros specify data dependences and can be either hand-written, or automatically generated by a SDL compiler from data-token flow schemes written in Scheme Definition Language (SDL). A dataflow application is statically partitioned and is distributed among processing nodes when the application is loaded to VSPD-1.

A complex dataflow application was performed and evaluated on VSPD-1. The application is numeric simulation of a dynamic object modeled by a 20-order system of differential equations solved with the Runge-Kutta method. The paper also presents a technique used for partitioning the dataflow application in VSPD-1 and the performance of the application.

1 The VSPD-1 Dataflow Machine

Dataflow computing is a subject of considerable interest, since this class of computer architectures exposes high level of parallelism. The VSPD-1 dataflow machine is a research prototype of a static dataflow architecture according to the classification of Arthur H. Veen[2]. A run-time environment of VSPD-1 supports static linking, partitioning and distribution of actors among processing units. The VSPD-1 machine was developed to be used as an accelerator for a computer with the PDP-11 architecture for numeric simulation of dynamic objects in real-time. In simulation experiment reported in the paper, the requirement of real time scale means that a simulation time must “run” faster than real time in the simulated object.

Although VSPD-1 was developed to include 16 processing nodes, a research prototype of the VSPD-1 machine built at LETI was constructed of five processing nodes (PN1-PN5) and a host computer (PN0), as depicted in Fig.1. Each VSPD node consists of a microcomputer MS11200.5 with the PDP-11 architecture 1MIPS processor and 32KB of RAM, a token-flow unit (TFU), ROM for a loader ad a kernel of the dataflow run-time system, and communication Qbus-Qbus adapters (CA).

---

1. This is an extended version of the article Vlassov, V.V., A.V.Kraynikov, B.A.Kurdikov, and D.V.Puzankov, A Dataflow System for Simulation of Dynamic Objects. In: Add. Proc. SCS’91 European Simulation Multiconference, 17-19 June, 1991, Copenhagen, Denmark, pp. 31-44.
   This version was published in Russian as a part of a technical report at Dept. Comp. Eng., State Electrotechnical University (former Leningrad Electrotechnical Institute, LETI), St.Peresburg, Russia, May 1991.
   The English translation presented here was done by Vladimir Vlassov.
2. A.H.Veen, Dataflow machine architecture, ACM Computing Surveys, Vol. 18, No. 4, 1986, pp. 365-396.

2 Organization of Computation Process in VSPD-1

In VSPD-1, dataflow computation is realized at the level of labeled Pascal code fragments called *actors*. An actor is enabled for execution when it collects all required data (tokens). When execution completes, result is sent to destination actors.

The system provides explicit and implicit data transfer from a source actor to destination actors. Explicit data transfer between actors located on different processing nodes is accompanied with transmission of tokens. A token is a synchronization signal that indicates that data has arrived to a destination actor. Implicit data transfer between actors located on the same processing node, is done through shared variables and, if necessary, can be followed by tokens. An actor is ready when it collects all needed tokens.

The token format is depicted in Fig.2. A token contains two 1-bit tags *E* and *T*, and a destination address (fields *N*, *S*, and *I*). The data type tag *E* indicates whether data followed by this token, is a vector or a scalar. The flow type tag *T* indicates whether this token follows data (dataflow) or not (control flow). The destination address carried by token, specifies three components: (i) the destination actor *N*, (ii) a segment *S* where the destination actor is located, (iii) an actor input *I* to which the token is directed.

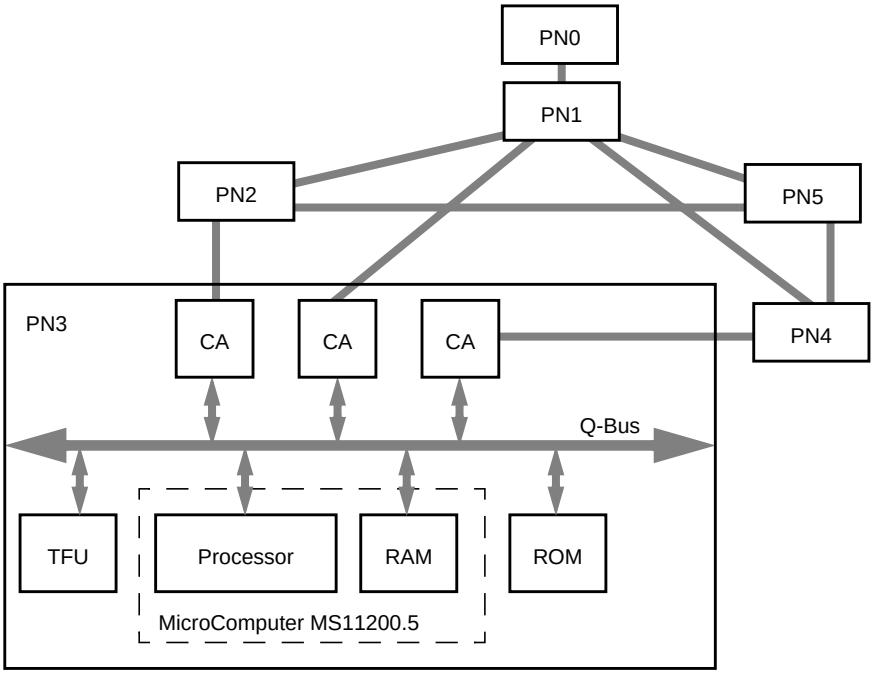

Figure 1. 5-node configuration of the VSPD-1 research prototype

Token transmission constitutes control flow since a token indicates an actor that (if ready) can be executed next to the source actor. The amount of token traffic can not be less than that of data flow. If data is directed to actors located on the same remote node, data and all tokens are passed in one transfer.

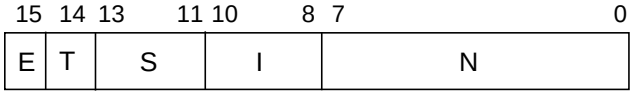

Figure 2. The VSPD token format

Token and data flow is manages by a distributed run-time system (monitors) with hardware support. On each node, a hardware Token Flow Unit (TFU) controls token flow and a queue of ready actors. The major functionality of TFU is to receive tokens, to maintain state information on actors allocated to the processing node (ready-bit vectors), and to supply the monitor with a number of next ready actor to execute. Data flow and execution of actors are controlled by software monitors. Although hardware TFU has not been completely operational, functionality of TFU has been implemented in software as a part of the monitor.

Each actor of a dataflow program is supplied with a number that is unique for a processing node where the actor is located. An actor number is used to look up directory entries for the actor in the following tables (directories):

- SAT: starting address table stored in RAM, that holds starting addresses of actors;

- RT: the table of ready vectors stored in the memory of the Token Flow Unit. A ready vector is a bit vector of *n* ready bits which indicate for each of the *n* actor inputs whether a token has arrived to the input.

A list of destination addresses (DAL) for an actor is used to send results and tokens. The list is stored in the memory of a processing node. In fact, an actor may have more than one list of destinations.

2.1 The TokenFlow Unit

As already mentioned, TFU receives tokens, maintains ready flag vectors, checks if an actor is ready, manages the queue of ready actors and supplies the monitor with a number of a ready actor. TFU consists of the blocks depicted in Figure 3.

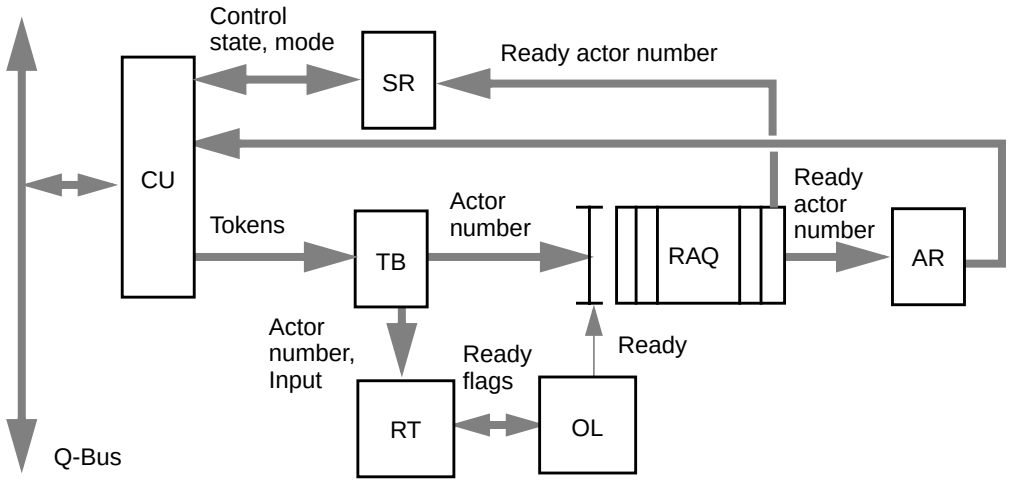

Figure 3. Block-diagram of the Token Flow Unit (TFU)

A Token FIFO Buffer (TB) can accept up to six incoming tokens.

A Ready flag Table is a 4KB SRAM for 265 16-bit-wide entries that stores readiness flag vectors and constant masks for actors allocated on the processing node. We assume, that an actor cannot have more than eight inputs, therefore a readiness flag vector is 8 bit wide. Each actor allocated on the processing node, obtains an entry in RT. Each record in RT is 16 bit wide, where the low byte contains eight 1-bit readiness flags (readiness vector), and the higher byte contains an “always-ready” mask (constant flags) which indicates for each of the 8 inputs whether the corresponding input is always ready (i.e., either a constant or not in use). When TFU receives a token, it extracts an actor number and an input number from the token (Fig.2), and updates the readiness information for the destination actor in RT: the readiness vector is changed to set the bit corresponding to the receiving input of the actor. As VSPD-1 is a static dataflow machine, it does not allow multiple tokens (data) on the same actor input. A token collision is signalled as an error if a token arrives at the input which is already occupied, i.e. appropriate readiness flag has been already set before. On token collision, TFU inserts a staring address of an appropriate error handler in the head of the ready actor queue (RAQ). TFU also checks if the actor that receives a token, is ready for execution. The actor is ready when all readiness flags in the low byte of the RT record are set. This indicates that all actor inputs are provided, and

the actor can be executed. The number of the actor is inserted to the ready actor queue (RAQ). On a request from the monitor, TFU supplies an actor number from the head of the queue. TFU is notified by the monitor when execution of the actor completes. When TFU receives such notification, it resets readiness flags (the low byte) masked with the constant flags (the higher byte) in the RT record for the actor.

The Ready Actor FIFO-Queue (RAQ) with 256 8-bit-wide entries is used to store numbers of ready actors waiting to be executed. If an actor is initially ready according to the initial marking of a dataflow scheme, its number is inserted into the queue before the program execution starts. When an actor receives a token, it may become ready, and its number is placed into RAQ. In the case of token collision, a number of a special error handler is inserted in the head of the queue. The monitor extracts the handler number and invokes the handler.

The Actor Register (AR) contains the number of an actor executed in the processing node. On request from the monitor, a number is extracted from the head of RAQ, passed to the requesting monitor, and placed to AR. If there are no ready actors in RAQ, then the AR is loaded with a reserved actor number (the “no-ready-actors” actor).

The Operational Logic (OL) operates on readiness flags, performs tests for token collisions and for readiness of actors. A TFU State Register (SR) stores information related to the state of TFU such as the TFU mode (loading or execution) and the “no-ready-actor” flag.

The Control Unit (CU) realizes a communication protocol on the Q-bus when the TFU is accessed by the monitor to fetch a ready actor, or to store a token.

2.2 The DataFlow Monitor

In each processing node, the monitor starts with a initialization phase that is performed only once to initiate state and control structures. The initialization phase consists of the following steps.

1. The monitor creates connection tables with communication information about neighboring processing nodes. Note that the monitor of a processing node communicates via memory mapped registers of Q-bus adapters.
2. The monitor waits for a special signal called R marker, and when it receives the marker, it sends a local address of a virtual shared memory segment to neighboring nodes. This address is used to access the shared memory segment via DMA.
3. If the local node is not the last in a chain of nodes, the monitor passes the R marker to the next neighboring node and goes to Step 4. In this way, the R marker is passed through all nodes. Each node exchanges addresses of a shared memory segments with its neighbors. If the local node is the last in the chain of nodes, its monitor passes a W marker back to the neighboring node from which it has received the R marker, and proceeds to Step 5.
4. The monitor waits for a W marker. When the marker arrives, the monitor sends it to a neighboring processing node and proceeds to Step 5.
5. The monitor starts execution of a dataflow program with an actor labeled MARK (“initial marking” actor) that finds a first ready actor (if any) to execute.

After the initialization phase, the monitor executes a monitor cycle until a zero token is encountered in a destination address list (DAL) of an actor. Zero token indicates the end of computation (end of dataflow). The monitor cycle consists of the following steps:

1. Fetch an actor. The monitor fetches a number of a ready actor from TFU. If there is no ready actor to execute, TFU returns the “no-ready-actor” status code, and the monitor retries. If TFU returns a ready actor number, it locks the Token Buffer for this actor, i.e. it stores the actor number and sets the input lock in the State Register (SR). Locking allows to prevent data and token collisions on inputs of the executed actor. Two locking mechanisms for actor inputs are described later in this section.

2. Execute the actor. The monitor looks up the Starting Address Table (SAT) for a starting address of the actor and executes a jump to the address.

3. Reset ready flags. When actor completes, the monitor informs TFU (resets an appropriate control bit in SR). TFU resets readiness flags of the actor in RT, unlock the lock on inputs of the completed actor, and loads the Actor Register (AR) with a number of the next ready actor (if any) from the head of the Ready Actor Queue.

4. Distribution of results and tokens to destinations. The monitor performs transfer of data and result tokens to destination actors listed in one of the lists “attached” to the actor. If the monitor finds a zero token in the list, it aborts the monitor loop (It transfers control to the Init system monitor, located in PROM).

As mentioned above, in order to avoid races that may cause data/token collisions on inputs of an actor, the actor inputs are locked when the actor is executed, and unlocked when the execution is done. The input lock is dynamically created when the actor is executed. The lock can be created in the State Register of the node where the actor is executed (as described above). The number of the locked actor is stored in SR. The register is inspected by a remote node when it intends to send data (and/or tokens) to the given node. If the actor number in SR matches to a destination actor number, the remote node must wait until execution of the destination actor completes, and only then attempt data and token transfer.

Another way to implement locking is to allocate locks for a given processing node on neighboring nodes. Before starting the execution of an actor, the node stores the number of the actor to the locks on neighbors and unlocks the locks after execution is finished.

3 Dataflow programs for VSPD-1

A dataflow application for VSPD-1 must be developed in a dataflow style as a collection of dataflow program modules to be loaded on processing nodes. Actors are programmed in Pascal and a dataflow scheme that specifies data dependences can be hand-written in macros in assembler Macro-11. To facilitate programming, we have developed a first version of a DataFlow Scheme definition Language (DFSL) which allows definition of dataflow schemes and automatic generation of assembler code for dataflow modules.

A dataflow module is a Pascal program where assembler code is included as control comments in Macro-11. Each module should include type definitions, a data segment, an actor code segment and a control segment for a dataflow scheme. We require, that a statically declared virtual shared memory (communication variables) used for data communication must be identical for those programming modules which communicate with each other. The communication variables must be specified at the beginning of a data segment. This is necessary because each monitor informs neighbors about an address of the static data segment in the dataflow module loaded to a processing node. These addresses are used for data communication via DMA. The monitor assumes that commutation variables are located at the top of the data segment.

An actor is a Pascal code fragment labeled with a unique assembler label, which is used as an actor entry point (starting address). All entry points are stored in a Starting Address Table (SAT). A dataflow scheme specifies data dependencies between actors and initial marking of tokens. In a program module, the scheme is represented by destination address lists (DALs) and a table of readiness-flags and constant masks (RT). The scheme can be hand-coded in MACRO-11 or generated from DFSL definition.

The Starting Address Table contains starting addresses of actors in order of their appearance in the module (to ensure a correct addressing of actors).

Each actor has a collection of destination address lists (DALs) associated with it. Each DAL is a block of 16-bit words labeled by a unique label. The first word of a DAL specifies a length of the list in words, every other word specifies a destination address (a token, see Fig.2). A zero token in a DAL indicates the end of execution. When the dataflow monitor fetches the zero token from the DAL, it stops execution of the dataflow module and breaks the monitor cycle (See 2.2).

A Readiness flag Table (RT) contains a 16-bits vector of ready and constant flags for each actor. In RT, the first 16-bit word stores the number of actors in the module, and next words contains readiness and constant flags of actor's inputs. RT should be loaded into the hardware TokenFlow Unit of the processing node with TFU (See 2.1).

A result of actor execution can be a value of a communication shared variable located in the communication area of the module. If a destination list is not empty, the value must be sent to destination actors. A code of each actor ends with a control assembler code (a distribute macro) that loads monitor's variables ADR, VL and DALA, with an address of data to be sent, a length of data in words, and an address of DAL, respectively. The code then transfers control to the monitor at point MOND, and the monitor distributes the result (specified in ADR and VL), and result tokens according to the list specified in DALA. If the distribute macro is placed in a Pascal conditional operator, then distribution of result and tokens is performed conditionally. With conditional distribution, an actor can have several destination lists. Conditional distribution allows programming of cyclic dataflow computation and conditional dataflow branching.

As dataflow modules are statically distributed among processing nodes, in order to build an executable module to be loaded to a given node, it is necessary to link the module with the following objects:

1. The VSPD-1 configuration module;

2. The VSPD Loader configured to be loaded into the given processing node (The Loader configuration depends on the VSPD-1 configuration and an order of loading of VSPD-1);
3. The dataflow monitor.

4 Simulation of a Dynamic Object on VSPD-1

To evaluate application performance on the VSPD-1, we have performed real-time numeric simulation of a complex dynamic object represented by a 20th-order differential equation system[1].

To be executed on VSPD-1, an application must be decomposed into actors which must be then assigned to modules, and, finally, the modules must be statically distributed among processing nodes. Efficient decomposition of an application into actors, and mapping of a dataflow program to processing nodes is an important and challenging task. Static mapping using a priori knowledge of characteristic of an application, in particular, a dataflow scheme, and a VSPD-1 configuration, must be determined before the execution begins. We have developed a suitable method for static mapping of dafaflow modules to VSPD processing nodes. The main goal to be achieved in the mapping method illustrated below is to minimize inter-node communication and to balance the workload among processing nodes.

To solve the differential equation system, we use the fourth order Runge-Kutta method. On each iteration (integration step), the method involves four evaluations of the right-hand sides. Values of left-hand sides constitute a vector of values of state parameters of the object (a state vector). We assume that for each equation, there is an actor assigned to the equation that evaluates the right-hand side of the equation and computes a state parameter value in the left-hand side of the equation. On each iteration, the computed value is transferred from the actor to other actors that need the value of the state parameter to compute right-hand sides of equations assigned to them. The state parameter value is also sent to an actor that collects all state parameters into a state vector for output.

To group actors into modules and to map modules to VSPD-1 processing nodes, we have calculated for each actor a normalized weight proportional to the computation time of the right-hand side assigned to the actor. To calculate the weights, we have used execution time estimates for floating point operations reported for the microprocessor used on VSPD-1.

Consider a occurrence matrix where rows correspond to equations (right-hand sides) and columns correspond to state variables (left-hand sides). An element in the i-th row and j-th column indicates whether the j-th state variable is used in computation of the right hand side of the i-th equation, i.e., a value of the variable computed by the j-th actor must be transferred to the i-th actor. In order to minimize communication between processing nodes, the occurrence matrix must be as similar as possible to a block-diagonal matrix, where ones are grouped near diagonal. For example, an ideal 3-block-diagonal matrix 6 by 6 with three blocks of size 1x1, 2x2 and 3x3 located on the diagonal looks like this:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

In our mapping method, actors (equations) that correspond to rows where one or more blocks of ones are located can be grouped into one module to be mapped to a processing node. Actors (equations) in a group use variables that correspond to columns where the blocks are located. Ideally, actors in the group should communicate mostly within a group. The group computes a part of an object state vector, that can be passed to an actor collecting all state parameters at the end of an integration step. As multiple hardware contexts are not supported in a node of VSPD-1, the actors computing right hand sides and mapped to the same node can be joint together and implemented as one actor computing right hand sides of several equations assigned to it.

1. In fact, the VSPD-1 was developed as a specialized computer for numeric simulation of complex dynamic objects.

An occurrence matrix that indicates the usage of variables in the right hand sides of equations that represent the studied dynamic object, is shown inTable 1. We compute a sum (by rows and columns) of average distances between ones and the diagonal of the matrix:

$$S = \sum_{i=1}^{n}\left(\frac{1}{l_i}\cdot\sum_{j=1}^{n} a_{i,j}\cdot|j-i|\right)+\sum_{j=1}^{n}\left(\frac{1}{l_j}\cdot\sum_{i=1}^{n} a_{i,j}\cdot|i-j|\right) \quad (1)$$

where - $a_{i,j}\in\{0,1\}$, $(i, j=\overline{1,n})$ is an element of the occurrence matrix; l_i is the number of ones in the i-th row; l_j is the number of ones in the j-th column.

To make a matrix as similar as possible to a block-diagonal one, the sum S must be minimized. A swap of a row i_1 with a row i_2 and a simultaneous swap of a column i_1 with a column i_2 do not change occurrence of variables in equations as specified in the initial matrix, but such two simultaneous swaps may cause a change in a value of the sum S (1) for the new matrix. To choose pairs of rows and columns to be swapped, we use weights calculated for a row i (or a column j) as average distances between ones and a position $k\in\overline{1,n}$ in the row (column):

$$\begin{cases} W_i^{(k)} = \dfrac{1}{l_i}\cdot\displaystyle\sum_{j=1}^{n} a_{i,j}\cdot|j-k| & i=\overline{1,n} \\ W_j^{(k)} = \dfrac{1}{l_j}\cdot\displaystyle\sum_{i=1}^{n} a_{i,j}\cdot|i-k| & j=\overline{1,n} \end{cases} \quad (2)$$

A pair of swaps (as described above) should decrease weights of swapped rows and columns.

We have developed a software tool MIM that allows transformation of an initial occurrence matrix to a matrix that is similar to a block-diagonal one. MIM was used to obtain a mapping of actors (equations) to five processing nodes of the VSPD-1 prototype. A result occurrence matrix and a mapping of equations to processing nodes (PN) are shown in Table 2. The table also shows weights T proportional to computation time of the right-hand sides. The weights T helped to balance the workload among processing nodes. The processing node PN_0 was used for loading dataflow modules to other processing nodes, and for collecting simulation results for output.

Figures 4-7 show four dataflow schemes of the application. In figures, actors are depicted as rectangles, inputs and outputs are numbered. Different outputs correspond to different destination address lists (DALs). Each actors has an assigned number where a subscript indicates the processing node in which the actor is located. An entire dataflow scheme can be obtained as a theoretical-set union of schemes in Fig.4-7.

The boot scheme is show in Fig.4. The IO actor located in PN_0 is initially ready, as it has tokens on its both inputs. The actor splits the initial 20-elements state vector (shown as a token on the first input) into five vectors of size 6, 1, 6, 2, and 5 elements according to the occurrence matrix and the mapping shown in Table 2. These vectors are transferred through TR (“TRansit”) actors to the FL (“FLight”) actors that will compute right hand sides. Tokens that follow the vectors are also collected at the B (“Barrier”) actor with number 8 that provides synchronous start of computation in FL actors. When the B actor collects all five tokens it puts enabling tokens on control inputs of FL actors, and integration begins.

Figure 5 shows the dataflow scheme for the first three stages of an integration step that involves three evaluations of the right hand sides. In this scheme, each FL actor computes the right hand sides of the equations assigned to it and exchanges results with other FL actors as specified in the scheme. The results (vectors EV_1-EV_5) are transferred via first outputs (DALs) of the FL actors.

Figure 6 shows the dataflow scheme for the fourth stage of the integration step in the fourth order Runge-Kutta method. The FL actors exchange result vectors (EV_1-EV_5) with each other, and also pass the vectors to the ME (MErge) actor. The vectors are passed from second outputs of the FL actors. The ME actor concatenates all five vectors EV_1-EV_5 into one state vector EV that is passed to the IO actor for output. The ME actor also synchronizes the FL actors by passing an enabling tokens to their control inputs. When an actor receives the synchronization token, it can start the next iteration specified by the scheme in Fig.5. Such synchronization between iterations is conservative. Nevertheless, it is convenient because it allows to simplify a check for the end of simulation after each iteration and prevents token collisions on the IO actor.

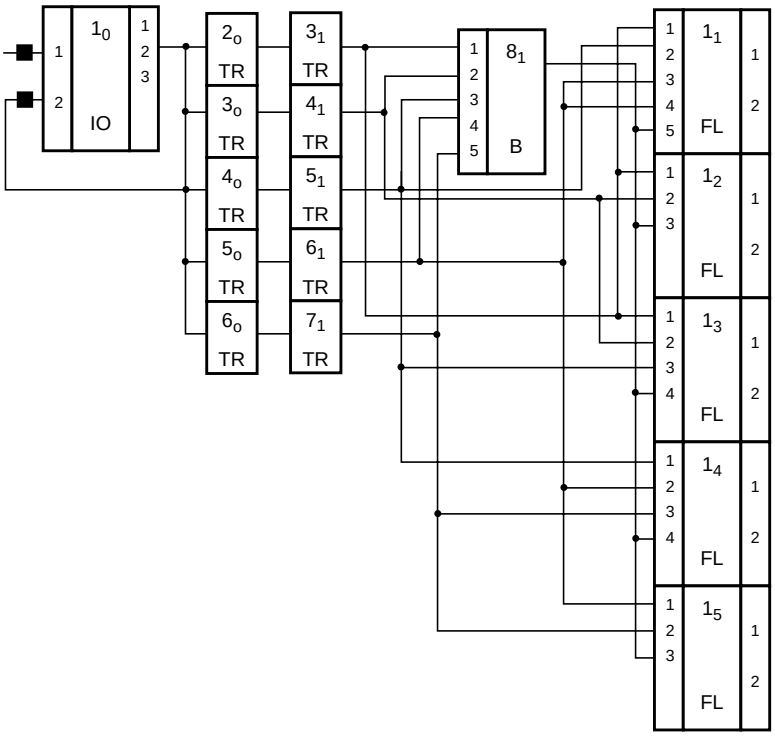

Figure 4. The boot dataflow scheme.

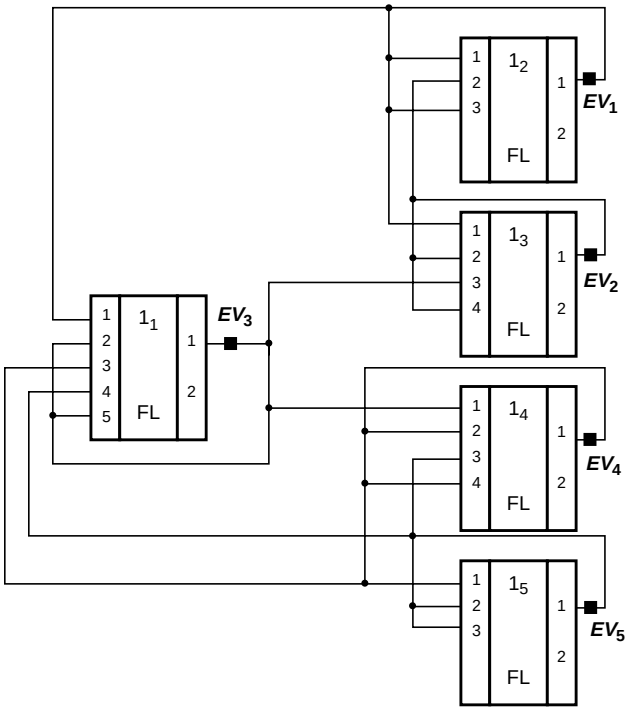

Figure 5. The dataflow scheme of a stage of the integration step (evaluation of right hand sides)

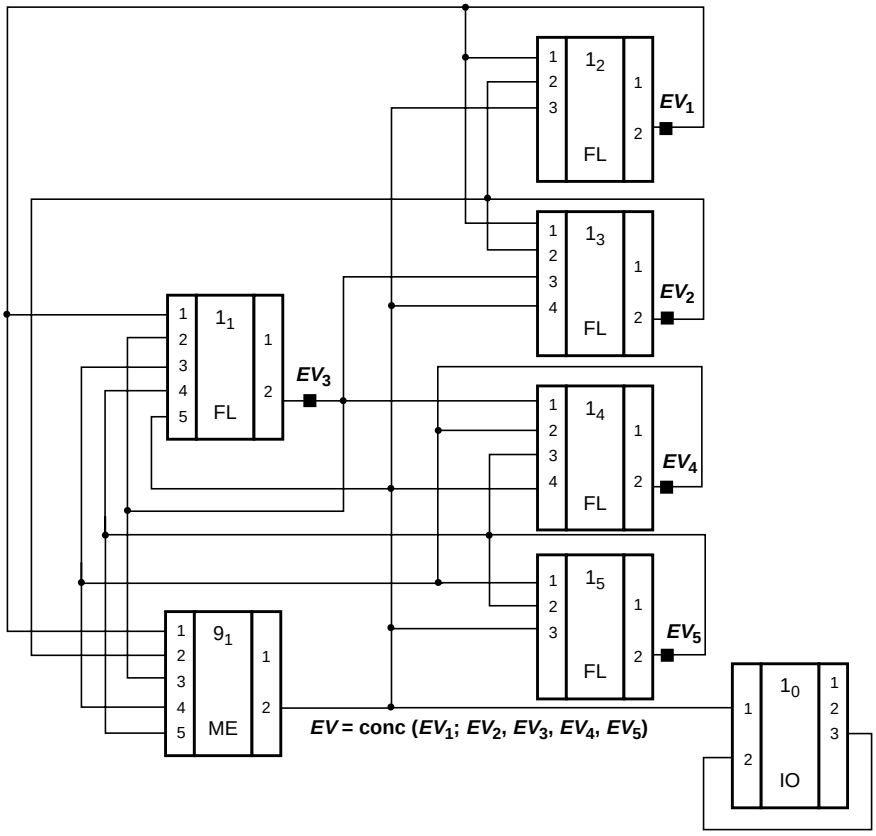

Figure 6. The dataflow scheme of the fourth stage of the integration step

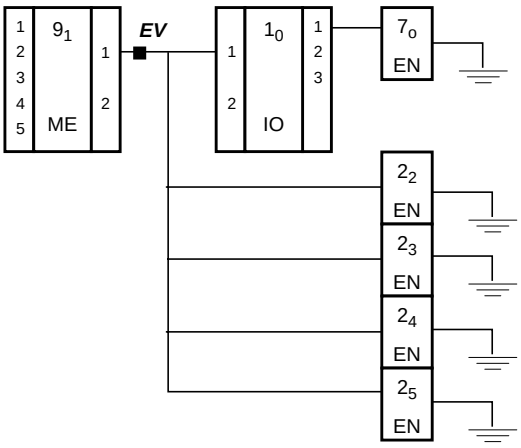

Figure 7. The dataflow scheme of the end of simulation

A dataflow scheme that implements the end of simulation is shown in Fig.7. Simulation stops when a specified condition is true, e.g. a specified state parameter either gets to zero or changes its sign. The condition is checked by the ME actor when it gets the state vectors. If the condition is true, the actor sends a token to the EN (“ENd”) actors in all processing nodes that generate zero tokens. As described in 2.2, a zero token forces the monitor to stop execution of the dataflow program.

Figure 8 shows execution time of the dynamic object simulation on the VSPD-1 for different number of processing nodes. As mentioned above, the performance requirement for this application was that the simulation time must "run" faster than real time. Figure 8 presents performance results of simulation experiments for 70 seconds of object time.

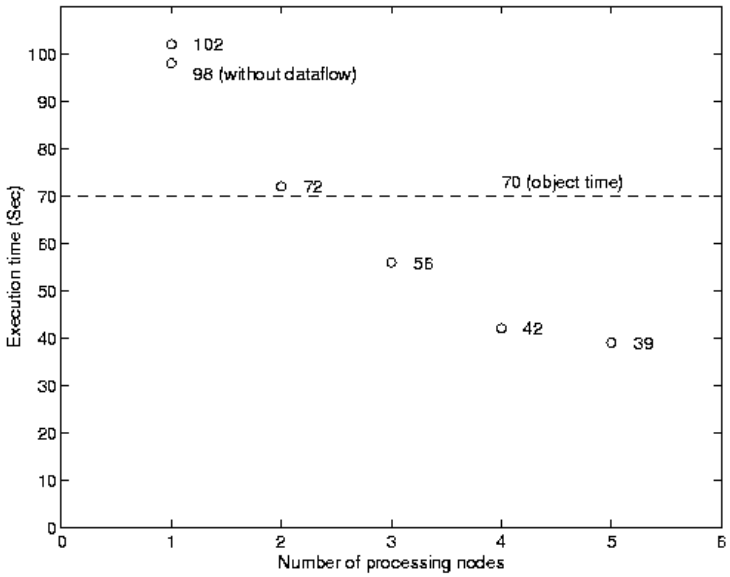

Figure 8. Execution time of dynamic object simulation (70 Seconds of object time) versus the number of PE

VSPD-1 is a research prototype of a static dataflow architecture that illustrates how a inexpensive dataflow machine can be realized. The VSPD-1 machine can be used, in particular, as a functional accelerator for mini computers. Our future work includes developing of a convenient programming environment for the VSPD-1.

Table 1: The Initial Occurrence Matrix

Equations (Actors)	State Variables																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1		1	1																
2	1	1	1	1	1	1	1	1	1											
3		1																		
4	1		1	1		1														
5		1			1															
6				1		1														
7	1		1																	
8								1		1	1									
9								1												
10								1			1									
11											1	1		1	1					
12										1	1	1	1	1				1	1	1
13										1	1	1	1	1				1	1	1
14													1							
15											1	1		1	1	1	1			
16											1					1				
17											1	1		1	1		1			
18										1										1
19										1									1	1
20								1			1	1								

Table 2: Mapping of Equations to Processing Nodes (FL Actors)

Equations	State Variables																				*T*	PN
	1	7	3	4	6	5	2	9	8	10	19	18	20	12	13	14	11	17	15	16		
1	1		1	1																	24	2
7	1		1																		24	
3							1														1	
4	1		1	1	1																48	
6				1	1																12	
5						1	1														12	
2	1	1	1	1	1	1	1	1	1												163	3
9									1												1	1
8		1							1	1							1				36	
10									1								1				17	
19										1	1		1								24	
18										1			1								12	
20									1					1			1				36	
12										1	1	1	1	1	1	1	1				91	4
13										1	1	1	1	1	1	1	1				91	
14															1						0	5
11														1		1	1		1		43	
17														1		1	1	1	1		67	
15														1		1	1	1	1	1	120	
16																	1			1	12	