Paralelization of the Sequantial Threads in DF Computers

Liberios Vokorokos, Norbert Ádám, Anton Baláž

Department of Computers and Informatics, Technical University of Košice
Letná 9, 042 00 Košice, Slovakia
E-mail: Liberios.Vokorokos@tuke.sk, Norbert.Adam@tuke.sk,
Anton.Balaz@tuke.sk

Abstract: One of the possible solutions of how to achieve higher performance of computer systems is represented by the concept of architecture of high-performance parallel computer systems. Traditional techniques of parallelism cannot fully utilize the characteristics of parallel architectures because of typical problems of modular and parallel decompositions of programs. In concept, research presented in this article represents the unification into paradigms of control-flow and data-flow computing. The research assumes the parallelization of the sequential threads, where forking and joining are used.

Keywords: dataflow, architecture, multipipeline processing, sequential thread, fork, join

1 Introduction

One of the possible solutions of how to achieve higher performance of computer systems is represented by the concept of architecture of high-performance parallel computer systems.

Monoprocessor systems based on the principle of a Von Neumann-type computer are trying to meet these demands by increasing the speed of individual parts of the computer. Possibilities of increasing the speed in this way, however, are determined by technological possibilities [4]. In conventional control-flow computers the processing of instructions is determined by the existence of control and synchronization signals. Instructions in these architectures are executed serially, or quasi-concurrently.

In most cases, programs contain those instructions, whose order of execution does not influence the correct interpretation of the whole program. The option of taking advantage of these independent instructions brought about a new concept of processing an instruction flow – processing instructions on the basis of data flow. The basis of this new semantic is the data-flow principle, according to which it is

possible to execute every instruction asynchronously, as soon as all of it operands are available. Computers controlled by data flow (Data Flow Computers) allow taking advantage of natural parallelism of the program, and so to shorten the time needed for the realization of a calculation. The advantage of the Data Flow (DF) architecture rests in the application of the principle of data control of the computing process in multiple-processor systems. It eliminates the problem of dead points, which can originate in Control Flow (CP) architectures.

Traditional techniques of parallelism cannot fully utilize the characteristics of parallel architectures because of typical problems of modular and parallel decompositions of programs.

This contribution deals with the DF KPI architecture model with the multipipeline execution unit, which is a research subject at the Department of Computers and Informatics at the Technical University of Košice [2].

2 Data Flow Architectures

Conventional Von Neumann computers are based on the CF computing model, in which the control of the computing process is realized through interpretation of the serial data flow of the program [1,4]. In frame of different directions of development of new generation computers with extremely high performance, presently, attention is paid to a special class of parallel computers based on the DF computing model, in which the control of the computing process is driven by the flow of operands (data) prepared to execute the program instructions [3,5].

A characteristic feature of DF computers is that instructions of a DF program are passively waiting for the arrival of a specific combination of its arguments, the access f which is being organized as a data control stream, in the sense of being data-driven. The instruction's interval of waiting for the arrival of operands represents its selection faze, in which allocation of computing sources takes place. Basis for the DF computing model is task mapping (Fig. 2) on processor elements (CEs – Computing Elements). In general, the task needs to be decomposed into smaller communicating processes, which are represented by the Data Flow program.

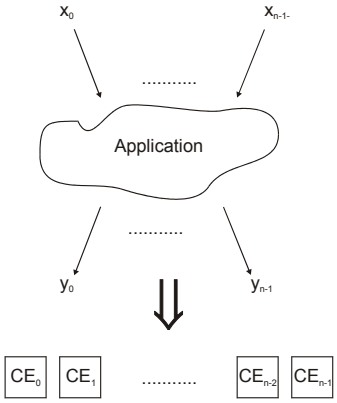

Figure 2
Mapping of tasks on CEs

A Data Flow program, utilizing the DF computing model, is represented by its machine representation, called dataflow graph (DFG). The DF computing model (DF program) makes it possible to detect parallelism at the lowest level, i.e. at the level of machine instructions (ILP – Instruction-Level Parallel).

Implementation of a DF computer architecture depends on the mode of instruction execution of the DF program, which runs as a process of receiving, processing and sending of activation tokens (Data Token - DT), representing data and flags on DFG edges. Depending on the way of processing activation tokens in DFG, or depending on the extent of architectonic support of its execution, two types of direct DF architectures are distinguished:

- Static models (Fig. 3).
- Dynamic models (Fig. 4).

In the static dataflow model (Fig. 3) the operator in the form of a node is executable when tokens (values) are presented in all input edges and a token is not presented in the output [3]. This model can take advantage of structural parallelism and pipelined parallelism. The static dataflow model found use in applications with frequent numeric computing structures.

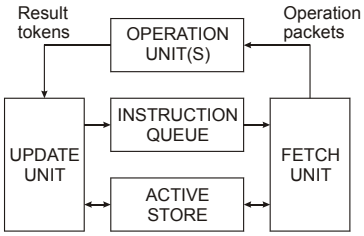

Figure 3
The basic organization of the static dataflow model

In the dynamic dataflow model (Fig. 4) the operator connected to the node is executable when all input edges contain tokens, whose marks are identical [3]. In this model, every edge can contain more than one signed token. When the node is executed, tokens belonging together are removed from input edges and on the output a token with a responding mark is generated. The dynamic dataflow model so uses loop parallelism, as well as recursive parallelism, which dynamically appear during the execution of the program. This kind of architecture must support the process of connecting operands.

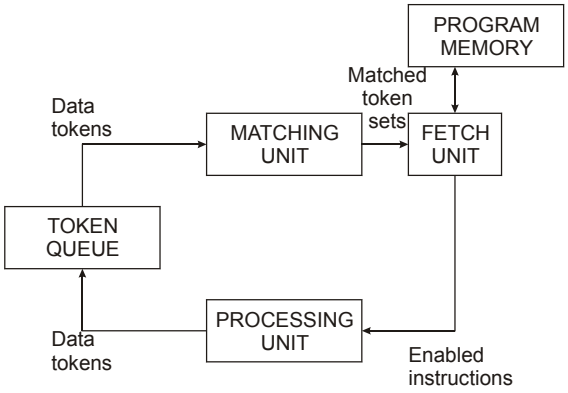

Figure 4
The basic organization of the dynamic dataflow model

In many dynamic dataflow architectures an associative memory is needed to achieve a connection of suitable data tokens (DT) (operands). At the moment, DF architectures are favored, in which associative concatenation of tokens is eliminated by explicit and direct memory addressing – using direct concatenation of operands (Monsoon, EM-4, Epsilon-2).

4 Multipipeline Processing

Multipipeline processing of DF Graph (DFG) is introduced on Fig. 5, where PS_k is the k-th computational stream of the DFG (k=1, 2, ..., n); AZ_l, Az_p are correspondent input operands (left, right) at the DFG operator O_m in k-th computational stream PS_k.

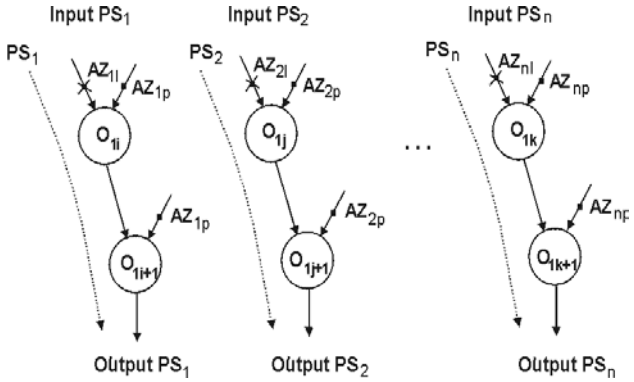

Figure 5
DFG of n-instructional streams

Any computational stream is executed by the pipeline organization of the coordinating processor CP, meaning, by the main components of the proposed architecture. CP consists of a linear structure of the pipeline stages SG, which are controlled by independent segment microprograms. On the structure level, the segments represent the pipeline stages LOAD, FETCH, OPERATE, MATCHING and COPY. Detailed structure organization of the DFKPI model is discussed in [2].

Synchronization, control and state signals are distributed by the Segment Control Part (SGC) of the pipeline stages. Concurrent execution of defined phases of the pipeline processing in the instruction streams of the DFG give possibilities for parallel implementation of the computing process at the instruction level.

In a direct matching, scheme, storage (called an activation frame) is dynamically allocated for all the tokens generated by a code-block. The actual usage of locations within a code-block is determined at compile-time; however, the actual allocation of activation frames is determined during run-time. In a direct matching scheme, any computation is completely described by a pointer to an instruction (IP) and a pointer to an activation frame (FP). The pair of pointers, <FP, IF>, is called a continuation and corresponds to the tag part of a token. A typical instruction pointed to by an IP specifies an opcode, an offset in the activation frame where the match will take place, and one or more displacements that define the destination instructions that will receive the result token(s). Each destination is also accompanied by an input port (left/right) indicator that specifies the appropriate input arc for a destination actor.

5 Multithreading

The notion of a thread in the context of multithreaded processors differs from the notion of software threads in multithreaded operating systems. In the case of a multithreaded processors a thread is always viewed as a hardware-supported thread which can be - depending on the specific form of multithreaded processors – a full program (single-threaded UNIX process), a light-weight process (e.g. a POSIX thread) or a compiler – or hardware-generated thread (subordinate microthread, microthread, nanothread, etc.).

In this point let's assume the DF contains many sequential threads of control. Each thread can be thought of as an independent instruction stream, which is executed in the Von Neumann control-flow way. Each thread has an independent set of registers. Each PE has a fixed limit as to the number of threads it can actually process simultaneously [2]. The state of a thread is contained in an execution interpreter EI (Fig. 6). The EI has five registers, which simply hold data associated with a particular thread: the execution register E, the value register V (the result of ALU operation is stored there), and three temporary registers R1, R2, R3.

The execution register E defines context, in which the EI thread is executed. IT contains a pair of pointers – the instruction pointer IP into instruction memory (points to the next instruction to be executed), the frame pointer FP into frame memory (base address of activation frame for procedure calling).

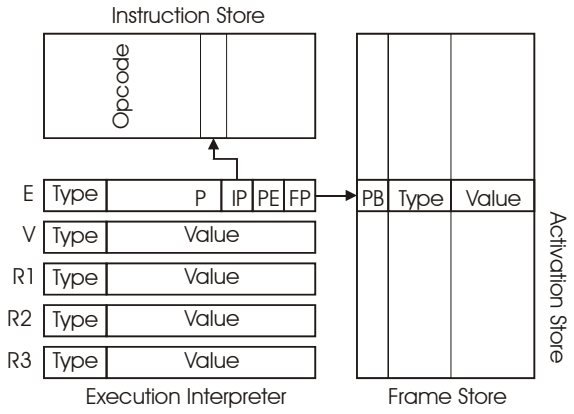

Figure 6
Execution Interpreter

Using frame addressing, one program block can have more active calls. The pair IP and FP has to have the same vertex number, and moreover, this node is PE, which will execute the thread. Register E also has a field called PORT P (used only during the Join operation).

Concerning the resident EI threads, every PE has a part of the global address space of the memory under control. We distinguish two types of address spaces:

instruction memory and data address space. Both have two parts: the node number and the offset into a part of address space under control of this node.

The PE fields of thread's E register indicates where it executes, and so references to the activation frame can be thought of as fetch instructions in the sense of local references [2,6].

Parallel driving operators, similar to sequential driving signal-operators in the Von Neumann model, have the task of controlling the instruction stream processing. While in the sequential driving operators's case, the operators are able to intervene only into the sequence of instruction execution, parallel driving operators are able to form and cancel new computing threads and so influence the number of active threads.

The basic instructions for manipulating the E and V registers – to create and synchronize threads, and to make global memory references presented from a multithread perspective are described in the following.

Assumed a program fragment:

R := (A - B) x (B + 1) (1)

Where R, A, B are local variables held in the activation frame. The example shows the FP-relative addressing mode for accessing these variables, which can only make local references, i.e. references to locations on the same node as is executing the construction. All of these instructions increment the IP field of E to move to the next instruction.

Command (1) can be compilated into a sequential thread, where every instruction is assigned into a sequential instruction memory address.

```
mov     V, [FP+A]           ;V := A
sub     V,B                 ;V := A – B
mov     [FP+R1],V           ;R1 := V
mov     V,[FP+B]            ;V := B
add     V,[LITERAL_1]       ;V := B + 1
mov     [FP+R2],V           ;R2 := V
mul     [FP+R1]             ;V := (A - B)x(B+1)
mov     [FP+R],V            ;R := V
```

Fork and Join. To establish a new thread into the computing process, serves the fork mechanism. Instruction fork is a combination of the jump instruction and establishing an instruction into a stream of processing instructions. Executing the „fork label" instruction has two effects. First, the present thread calls the following instruction (field IP is incremented) and second, a new thread is established into the system, whose registers E and V are identical to the present EI, while it is expected, that field IP in E is the name.

The Fork instruction may set the port in addition to computing a new IP. When a thread executes a non-joining instruction, its E register specifies the left port.

Two threads executed on the same PE can synchronize each other using the Join mechanism. For example join-subtract could be described as:

label [FP+0]: sub vL, vR (2)

Join mechanism in expression (2) is indicated by the presence of the memory operand in front of the colon, which divides the expression.

The idea behind a Join is that two threads will fetch a join-modified instruction at different times, but only the second one (in time) actually performs the indicated operation and continues. The first thread to execute merely saves the context of its V register, and then dies without finishing the join-modified instruction or proceeding to the next. When the second thread executes the join-modified instruction, it retrieves the saved value, so that both V registers are available as its operands. Two threads participating in a Join have different PORT bits in their E registers, thus left and right operands can be distinguished regardless of the order in which the two threads happen to execute. The example using FORK and JOIN to cause the multiplications in expression (1) to execute it in separate threads is shown as follows:

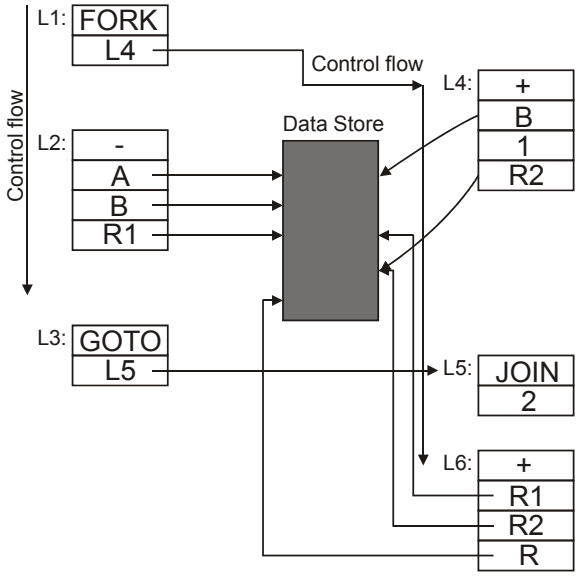

Figure 7
Parallelized sequential threads

Conclusion

The research of DF architectures was motivated by the need to achieve higher performance. Research doesn't eliminate the Von Neumann architecture while it is able to effectively process dataflow tokens and effectively execute the sequence code.

In concept, research presented in this article represents the unification into paradigms of control-flow and data-flow computing. New technologies and algorithms were presented at a theoretical level.

From a different point of view, multithread processing is a confirmation of the fact, that parallelism at an instruction level is well taken advantage of in the way of sequential thread executions in streams.

This addition could lead to further research of DFKPI dataflow architecture, which provides the utilizing of natural parallelism.

References

[1] Jelšina, M.: *Architectures of Computer Systems (in Slovak)*. Elfa s.r.o., Košice, 2002, ISBN 80-89066-40-2

[2] Jelšina, M. a kol.: *Design of Computer System Data Flow KPI (in Slovak)*. Elfa s.r.o., Košice, 2004, ISBN 80-89066-86-0

[3] Sima D., Fountain T, Kacsuk P.: *Advanced Computer Architectures – A design Space approach (in Hungarian)*. Szak Kiadó Kft., Bicske, 1998. ISBN 963 9131 09 1

[4] Tanenbaum, A. S.: *Structured Computer Organization* (in Hungarian). Panem Könyvkiadó Kft., Budapest, 2001. ISBN 963 545 282 9

[5] Vokorokos, L.: *Data Flow Computer Principles (in Slovak)*. Monograph. Copycenter, spol. s.r.o., Košice, 2002. ISBN 80-7099-824-5

[6] Papadopoulos, G.M., Traub, K.R.: *Multithreading: A Revisionist View of Data Flow Architectures.* In Proc. of the 18th Int. Symp. on Comp. Architecture, 1991, Toronto, pp. 344-350

Supported by VEGA project No. 1/1064/04