Data-flow graph 로부터 Systolic Array 에의 변환방법

*박 명 순, **진 주 식

*포항공대 전기전자공학과, **서울대 전산기공학과

A Mapping Method of Data-flow graphs into Systolic Arrays

Myong-Soon Park, C. S. Jhon

Dept. of Elec. Eng., POSTECH , Dept. of Computer Eng., SNU

ABSTRACT

Previous methods to map from a FORTRAN-like specification into a systolic array were difficult to find data dependencies because the specification was expressed and executed sequentially. Data-flow graph(DFG)s show data dependencies explicitly. In this paper we show a mapping tool from a DFG specification into a systolic array. We introduce the concept of a Systolic Pattern Stream(SPS) and use that concept to derive a systolic array.

I. Introduction

To have very high speed, many computer systems use multiprocessor system. But the communication problem between processors becomes the bottleneck of the muitiprocessor system.

Systolic array alleviates that kind of bottleneck by allowing only the local communication between neighboring processors. But the design of a systolic array for a specific algorithm is very difficult. Many researchers have tried to find a systematic method to design a systolic array from a high level specification. Most of them try to find the method from a FORTRAN-like specification. They try to find dependencies between data and use them. But it is very difficult to find the dependencies between data because the specification was expressed and executed sequentially.

In this paper, we present a tool which is useful in the design and analysis of the systolic array. Our treatment for the above problem is to use the DFG notation as a high level specification. DFGs express data dependencies explicitly. Therefore we can solve the problem of concurrency easily.

In general, a given data-flow graph provides functional representation of many circuits. In fact, some restrictions imposed on the graph can achieve a practical one-to-one mapping to the circuit realization. The restriction could be a function evaluation scheme which implies an evaluation sequence and the amount of computational resources involved, etc. Our compilation heavily relies upon such a restriction. We introduce the concept of the Systolic Pattern Stream(SPS). By using SPS we transform a DFG into a systolic array. To get the target system or improve the performance, we often need to do preprocessing and/or postprocessing. As a preprocessing, we mean the structural change in a DFG level and as a postprocessing, we mean the structural change in a systolic array level. But these subjects are not dealt with in this paper.

This paper consists of three sections, in addition to this section. Section II gives the characterization of the data-flow graph. Section III discusses the mapping of DFG's into systolic arrays. Finally, Section IV concludes with some results concerning the properties of systolic arrays.

II. Characteristics of data-flow graphs.

Data-flow language is suitable for some form of parallel computers like data-flow machine. The basic idea behind a data-flow computing model is that activities should be initiated asynchronously by the arrival(or availability) of the necessary information required to perform that activity.

A program in a high-level data-flow language can be directly translated into a directed graph at the machine language level with little analysis. The graph consists of nodes and arcs. The nodes represent functions and arcs represent data dependencies between functions. In this graph values are represented as tokens on the arcs. The data simply flows to the instruction nodes. The arcs between nodes can be viewed as FIFO storage pipes which queue the data items until the node is ready to remove them for node execution. When all the operands are present (each queue has transported a data item to the node), the node removes a datum item from the head of each queue , computes the result, and passes the resulting data value along the output arc from the node. Due to the queueing behavior of the arcs, the data-flow model shows the parallelism (temporal and spatial) provided that sufficient data arrives to drive the concerning nodes. Because datum is used as a value rather than as a storage location, DFGs have no side effects. And because the order of statement execution is dependent upon functional relationships, it shows the data depedencies explicitly. This property with no side effects property makes the program verification easy.

We shall present a simple framework to specify

algorithms. We shall employ simply types of stream-based recursions using primitive functions defined below:

_FIRST evaluates to the first element of the input stream. For example, FIRST:(1^2^3^....) = 1.
_REST evaluates to the stream derived by removing the first element from the input stream. For example, REST:(1^2^3^...) = 2^3^... .
_FBY (denoted by ^) forms a stream out of a value and a stream, so that FIRST:(FBY(a,X)) = a and REST:(FBY(a,X)) = X. For example, FBY (1,2^3^4^...) = 1^2^3^4^... .

The nodes defined above doesn't exhaust the primitive functions we can define. We can define more popular primitive functions and can make other nodes by combining some nodes defined.

A FIR filter, which we'll use as an example, can be specified by the nodes described above as in Fig.1. In that figure, the dotted boxes denote the blocks, which are the interconnections of nodes.

III. Mapping of DFGs into systolic arrays.

Our compilation scenario tolerates the DFG whose characterstecs are given as follows.

i) The DFG consists of several types of blocks, but the interconnection patterns between blocks are indentical except at the boundary.
ii) No successor of a block can be both a non-immediate successor and an immediate successor. When block A has a path to block B, block B is called a "successor" of block A. If that path does not include any block between A and B, then the block B is called an "immediate successor".
iii) The DFG can include block cycles. Here, by a block cycle, we mean a cycle formed at a block level by the interconnection of blocks.
iv) Only one fanout is allowed inside a block.

By the target system, we mean a systolic array which can be synthesized without destroying the interconnecting structure between blocks in a given data-flow graph. Our target system has the characteristics as follows:

a) The systolic array is a synchronous system, where the clock is provided to buffers.
b) The realizations corresponding to indentical blocks have same realization patterns.
c) Buffers can be inserted in the realizations of interconnections only between blocks.
d) The target systolic array operates in a pipelined manner, wherein the cell corresponds to a block. A pipelining buffer is to be realized by flip-flop driven by the global clock which will be a single phase or n-phase clock. As a default, a pipelining buffer is to be realized by master-slave(MS) flip-flop driven by a single-phase global clock.
e) The target systolic array has the minimal number of queueing circuits.

A brief scenario of our systolic compilation is shown below:

1) Derive patterns of function evaluations for a given DFG, as if it were our target systolic array.
2) Realize the DFG by the systolic array, as directed by the function application patterns derived above.

To describe our procedures, we define a "Systolic Pattern Stream(SPS)" which describes the flow of data through arcs of a DFG, as if the corresponding target systolic array behaved.

Definition. A systolic pattern stream X, which is a stream of binary numbers, is associated with any arc of a given DFG. Here 1(0) of the ith element of the stream X denotes that the interconnection in the target systolic array of DFG, which corresponds to the arc, passes (does not pass) a valid datum at the ith clock period when the system works to have fastest outputs.

Now for the purpose of derivation of SPS from the graph alone, we interpret a DFG as if it were a target system.

A) At each node, only all necessary input tokens are needed and consumed provided they are available and the output token being produced by the node can be transmitted to a neighboring block through an arc or by a reachable node (including this node), which does not produce a token, inside that block. Here, by an available token, we mean that the token is present sometime during the clock period, not necessarily only at the beginning of the clock period.
B) Each arc passes at most one token during the same clock period.
C) When the datum pass buffers including MS buffers, it will take a unit time delay to pass.
D) It takes no time delay to process and transmit the data at any node except buffers.
E) MS buffers are located at input ports.
F) Data have an unit speed.

Now we consider how we could derive SPSs for the given DFG in a constructive way.

First of all, we consider the derivation of SPSs for a block. To satisfy the characteristics e), data must arrive at proper clock period and as fast as possible. If the DFG has one output, then it is easy to compute SPSs in a backward direction. If contradictions of partially derived SPSs are found, then the beginning of this procedure is resumed with modification of hypothetical SPSs for the output.

If a block has more than 2 outputs, it is difficult to compute SPSs. In this case, derivation of SPSs needs to be started with hypothetical SPSs for those outputs. In general, whether we can derive SPSs depends upon how we assume the SPSs for those outputs.

Let's consider a DFG with bi-directional interconnections. As we see in Fig.2, the REST node can be replaced by FBY node because the input sequences at points A and B of the left DFG are same as those of the right DFG, respectively. In this case, the direction of data flow is reversed. By using that, we can change the structure in Fig.1. The left-most REST node is deleted. The changed DFG is shown in Fig.3, where the blocks are reconstructed. This DFG can be used as a specification of a FIR filter from the beginning.

Before we compute SPSs for this DFG, let's consider the interval between valid data. In Fig.3, the second datum at P1, which is the first

datum at P2, must meet the second datum at P2. Because we have propagation delays to propagate a block in systolic arrays we can know that interval between data in systolic system must be same as the time to pass the block cycle, In this example, the interval between data must be 2. The SPSs for blocks shown in Fig.1 and Fig.3 are shown in Fig.4(a) and Fig.5(a) respectively.

To constructively derive SPSs for an overall DFG from SPSs for a block we connect new block to partially constructed DFG and adjust SPSs and so on until we construct the source DFG. To describe the condition needed to meet when a new block is connected in terms of SPS, we define a notion of "accumulatively greater than or equal to" as follows.

Definition. For two infinite streams P and Q, P is said to be accmumlatively greater than or equal to Q if and only if the sum of first i elements of P is greater than or equal to the corresponding sum for Q for every positive integer i.

Whenever a new block is connected, the following two conditions need to be satisfied during the computation of SPS's to meet e) and D) in target systems. First one is that at an interconnecting arc between blocks, SPS at one end must be accumulatively equal to SPS at the other end. Second one is that at other interconnecting arcs SPSs at beginning end must be accumulatively greater than or equal to those at the other end respectively. The SPSs need be adjusted such that these two conditions are satisfied. We continue this until we construct the source DFG. The procedure to derive SPSs for an overall DFG is given below:

Procedure I

1) Compute SPSs for a block in a given DFG. The data interval must be same as the length of block cycle.
2) Connect new block and adjust SPSs to meet above two conditions.
3) Repeat the step 2) until we construct the source DFG.

By applying Procedure I, we computed SPSs for the given DFGs, which are shown in Fig.4(b) and Fig.5(b). From these, we can realize systolic arrays directly. To realize a DFG in a systolic array, follow the Procedure II.

Procedure II

1) Inside a block, all nodes except apply-to-all nodes such as adder and multiplier will be removed. The correct operation can be provided by sampling I/O at proper times.
2) If interconnecting arcs between blocks have different SPSs, then insert the buffers. The number of buffers is same as the delay between the first valid data at two arcs.
3) Connect the clock to every buffer only.

If we apply procedure II to Fig.4(b) and Fig.5(b), then we can get the realizations shown in Fig.6 and Fig.7, respectively.

Here the weighting factors are not changed in time. So they can be stored registers. And the first input to FBY node can be stored in advance. In Fig.7, the initial conditions are used for the implementation of the first input element to FBY node.

IV. CONCLUSION

We proposed a new method to realize the systolic array from a given DFG. We proposed a systolic pattern stream to compute a correct sampling time from a given DFG as if that DFG were a target systolic array. As a future work, we have to define more primitive nodes to describe a wide class of DFGs. And we have to devise a way to reconstruct DFG such that it have regular structure. Also developing more structured method to derive SPSs dor the DFG with more I/O ports.

REFERENCES

[1] A. L. Davis and R. Keller, "Data flow programs", Computer, Feb.1982, pp.26-41.

[2] C. S. Jhon and R. Keller, "Deadlock analysis in the design of data-flow circuits", 21st Design Automation Conference, June 1984, pp.705-707.

[3] H. T. Kung, "Why Systolic Architecture?", Computer, Jan.1982, pp.37-46.

[4] G.-J. Li and B. W. Wah, "The Design of Optimal Systolic Arrays", IEEE Trans. on Computers, Vol.C-34 No.1, Jan.1985, pp.66-77.

[5] D. I. Moldovan, "On the Design of Algorithms for VLSI Systems", Proc. of the IEEE, Jan.1983, pp.113-120.

[6] P. Quinton, "The Systematic Design of Systolic Arrays", IRISA Research Report No.193, April 1983.

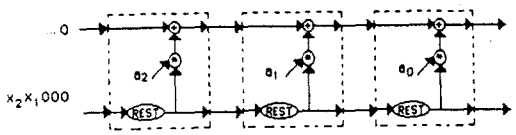


Fig. A DFG for a FIR filter

Fig.2. An example of structural change

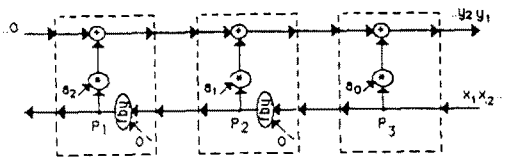


Fig.3. Another DFG for a FIR filter

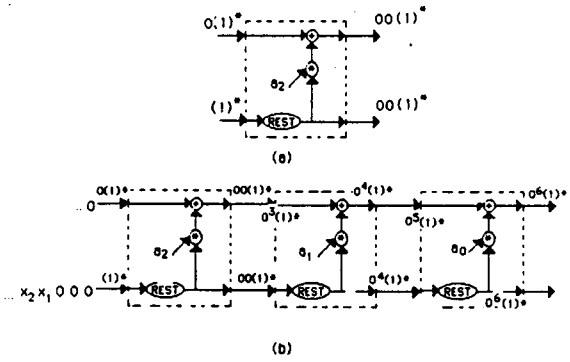


Fig.4. SPSs for a DFG shown in Fig.1
(a) SPSs for a block
(b) SPSs for overall DFG

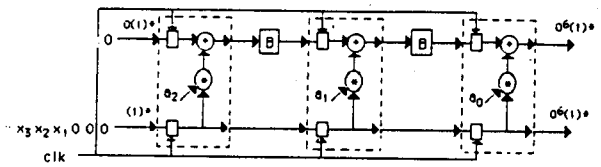


Fig.6. Realization of a DFG shown in Fig.1

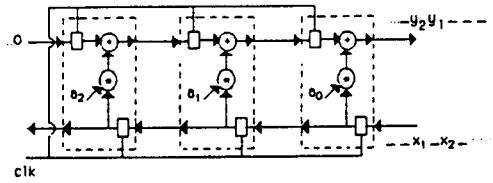
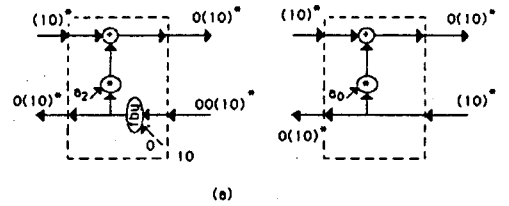


Fig.7. Realization of a DFg shown in Fig.3



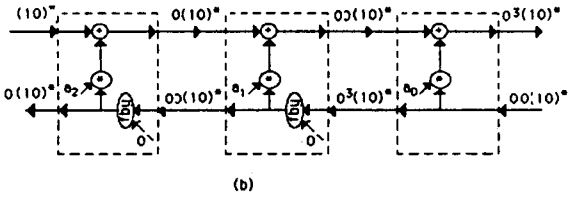


Fig.5 SPSs for a DFG shown in Fig.3
(a) SPSs for a block
(b) SPSs for overall DFG