Seminar on

Architecture and Design methods for Embedded Systems

Comparison of different Data Flow Graph models

Submitted by:
Sameer Gupta, 2241236
Under guidance of Mr. Dominik Luecke

INDEX

1. Abstract

This paper studies different Data Flow Graph Models currently available and used by researchers. The paper emphasizes on the extended models used, without substantially proving their properties. The properties of these models often contradict with the properties of the model used by other researchers, and hence undermine the previously proved results about the Data Flow Graphs. This has essentially led to many definitions of Data Flow Graph models. Here in this paper we study the basic features of the Data Flow Graphs and compare the extended Data Flow Graph Models.

2. Introduction

Data Flow Graphs have always been extensively studied and used because of their ability to specify algorithms exhibiting parallelism and / or asynchronous activity. For this reason Data Flow Graphs have been used successfully in the simulation of hardware like computer systems. This paper also takes a look at the various extensions of the Data Flow Graphs currently under use and try to find a minimum subset of the features which are common to all of these models. A model based on these common features can be used as the starting point of all other Data Flow Graph extensions.

We start this paper with a brief introduction and background of the Data Flow Graphs in section 3. There we discuss the reasons and advantages of Data Flow Graphs. In section 4, we talk about the various Extended Data Flow Graph Models available. Though the list presented here in this paper does not cover all the models of the research world, but I have tried to discuss the most commonly used ones. Later in section 5, we compare the properties of the models discussed in section 4. We also find the features which are common to all the models and hence develop the basic Data Flow Graph Model. Finally, we conclude this paper in section 6, with a brief conclusion.

3. Background of Data Flow Graphs.

3.1 Reasons for existence of Data Flow Graph

Increasing demand for higher computation speed has generated considerable interest in the field of parallel computation, concurrent operations within computer systems representing parallelism in hardware, parallelism in algorithm and new programming languages for parallel computers.

Data Flow Graph have been and are being extensively studied and used because of their respective ability to specify algorithms which exhibit high degree of parallel and / or asynchronous activity. For this reason Data Flow Graphs have been used successfully in the simulation of hardware like computer systems.

3.2 Advantages of Data Flow Graphs

The chief advantage of Data Flow Graphs over other models of parallel processing is their compactness and general amenability to direct interpretation.

Also, the algorithms expressed in a Data Flow Graph Model are controlled by the arrival of data (token) at the transformational actors (nodes). This can be contrasted to control flow environments where the locus of execution is based on an instruction pointer which identifies the operation to be performed at any point in time.

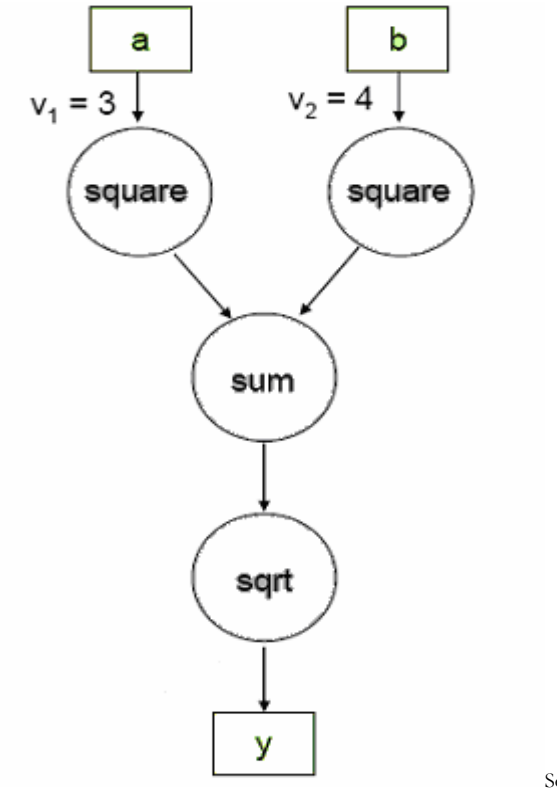
4 Data Flow Graphs Models.

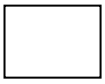
There are several extensions of Data Flow Graphs; these have been proposed because they provide useful capabilities for the researchers and interesting usage in various applications. These extensions often pose greater challenges to the Data Flow Simulator System implementers. Also these extensions often make the previously proved properties of Data Flow Models inapplicable. In this section we would have a brief look at some of these extensions of the Data Flow Graph Models. These models includes:-

- Data Flow Graph Model without intermediate nodes to store results.
- Data Flow Graph Model with intermediate nodes to store results.
- Data Flow Graph with named tokens.
- Data Flow Graph Model with stream of tokens.
- Data Flow Graph Model for Re-configurable systems.

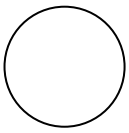
4.1 Data Flow Model without intermediate result storage [2].

This Data Flow Graph Model doesn't have any intermediate result storage nodes. The data from one operation unit is fed to another operation unit. This means that the edges act as the placeholders for the intermediate results. Formally this Data Flow Model is described as:





A square represents external input and output.



A circle represents an operation performed on the input token and output tokens being produced.

Source: Lecture notes M.S.S. WM05/06

Figure 1

DFG: **<N, A, V, v^0, f>**

- **N** = $\{n_1, n_2 \ldots n_m\}$: Set of nodes.
- **A** $\subseteq N \times N = \{a_1, a_2 \ldots a_n\}$: $n \in N$
  Set of Arcs between the nodes.
- **V** $\subseteq V_1 \times V_2 \times \ldots \times V_l = (v_1, v_2, \ldots, v_n)$:
  Vector of values associated to arcs,
  $v_i \in V$ is value at arc a_i
- $\mathbf{v^0} \in V$ is initial value vector.
- $\mathbf{f} = \{f_{n_i} : \prod_{a_j \in I(n_i)} V_j \rightarrow \prod_{a_k \in O(n_i)} V_k \mid n_i \in N\}$:

Function performed by each node $n_i \in N$, $I(n_i)$ and $O(n_i)$ are set of incoming and outgoing arcs for each node $n_i \in N$.

In the figure 1, nodes without incoming arcs are called input nodes, while the nodes without outgoing arcs are called output nodes.

This Data Flow Graph Model has the following properties: Here nodes receive only one data (token) per incoming arc. Also nodes generate exactly one data value (token) per outgoing arc. And there is no feedback possible.

But these properties lead to limitations, such as some computations may require more data (more then one token) for input or it may produce varying amount of data as output. Example, as often in up sampling or down sampling the amount of data at input or output varies.

4.2 Data Flow Model with intermediate result storage [3].

This Data Flow Graph has intermediate result storage capacity. This leads to insertion of nodes which hold the results from the previous operation. These nodes remove the necessity of the edges to hold intermediate results. Formally, this Data Flow Graph is defined as,

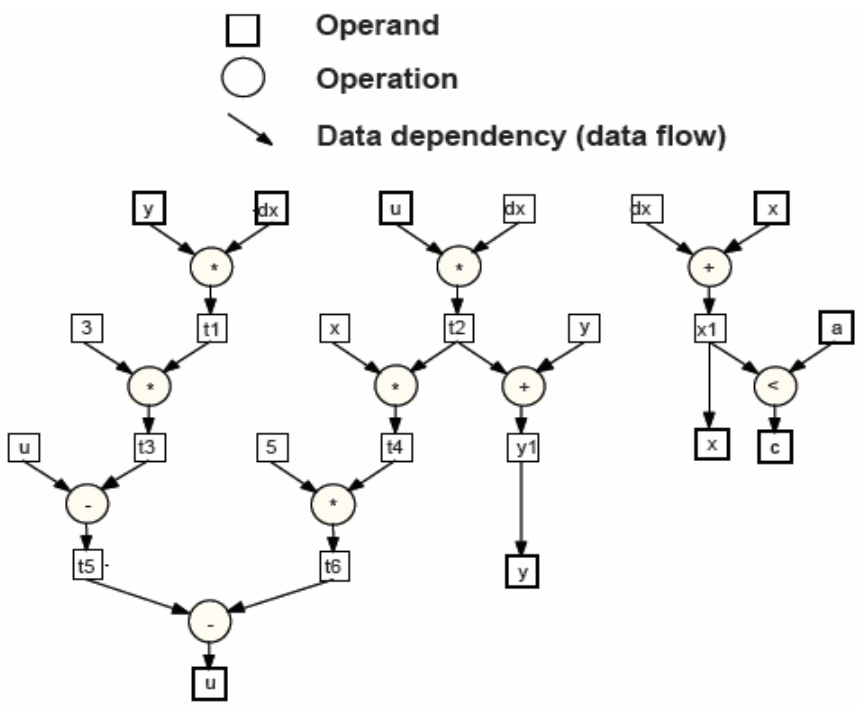


Figure 2

Source: Lecture Notes ESE SM06

DFG = **{O ∪ V, D}**

- **O** = Set of operations of the system. These are represented by the circles.
- **V** = Set of operands of the system. These are represented by the squares.
- **D** = Set of edges, data dependencies between operands and operators. These are shown by the directed lines between operands and operations.

The model shown in figure 2 is similar to the model in the previous section (figure 1) except for the fact that now there are nodes between two operations which store the intermediate results. In this model the results are no longer stored on the arcs, but they are stored in these specially added nodes.

In the example shown in figure 2, the values 'y' and 'dx' are multiplied and the result is stored in the node 't1', which is a temporary storage for the intermediate results. Often these nodes are mapped to the registers in the hardware. But many nodes may often share the same register; depending on how do we need to store the intermediate results. Thus the total number of intermediate result storage nodes doesn't give a count of the total number of registers required in the hardware.

This Data Flow Graph Model has the similar advantages and disadvantages as the model discussed before. In fact this model represents the same concept / design in a different way. Unlike the previous model, this model has special nodes to store the intermediate results.

4.3 Data Flow Model with Named tokens.

This extension generalizes how the nodes are specified. A specified node may contain the "name" of the operation that is to be performed. The corresponding operation is "applied" by the "apply node" (OP1, OP2…) to the incoming tokens with other constraints of Data Flow Graph Model retained. This results in nodes which are useful in many algorithms particularly where a given function will be selected from a set of function based upon the instance of invocation. This realization of named node is straight forward when the resulting operation specifies a primitive node. However, if the target operation is itself a dataflow procedure, the routing of procedures across the procedure boundaries at procedure invocation and termination must be accommodated. The major difficulty is in maintaining the proper association of input and output tokens at the procedure boundaries when two or more procedures originate from the same apply node.

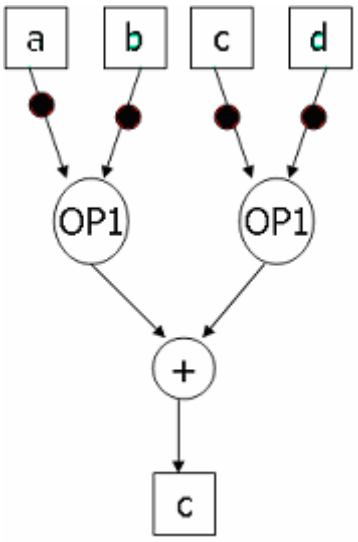


Figure 3

In the example shown in figure 3, we have OP1, which represents a set of operations. In this example we assume, OP1 represents the set of arithmetic operations. Each token, along with the data value has another value (name / label) which represents the operation to be performed out of a given set of operations. Hence in this example, the tokens from external nodes 'a' and 'b' might be added while the tokens from node 'c' and 'd' might be subtracted or

multiplied, depending on the name / label value of the token. The composite structure of a token containing the name of the operation and the data value is shown in the figure 4.

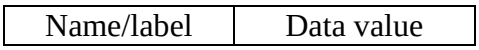


Figure 4

Also we might have these "apply nodes" present in multiple hierarchical levels, and in this situation we would need to embed this composite structure shown in the figure 4 recursively so that there is a name value associated with each apply node.

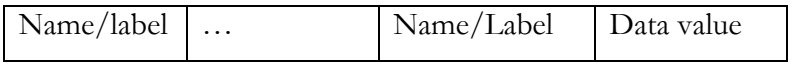


Figure 5

Another problem which may arise in this model is that, we might have different names associated with the tokens which act as an input to the same apply node. Here I would like to suggest two solutions. First and the simpler solution would be that the model designer makes sure that this situation never arises. But an interesting solution can be that we associate a priority table which can be used to resolve this conflict. In figure 3, the tokens from nodes 'a' and 'b' might have labels 'add' and 'subtract' respectively, and assuming that we gave higher priority to 'add' operation, we would chose 'add' as the applied operation and the result would be 'a+b' and not 'a - b'.

4.4 Data Flow Model with Stream of tokens

Another Data Flow Graph Model extension is the introduction of streams of tokens [4, 5]. Stream provides an effective way to reference several homogeneous data tokens as a group. The support for streams of simple tokens can be easily introduced by providing several stream manipulation nodes and a data type for end of stream identification. The generalization of this model to include streams of stream of tokens requires extended support. However there are no clear specifications for such systems containing stream of tokens.

This model looks similar to the models discussed in the previous sub sections, with the only difference that instead of a single input, we give it a stream of input data.

A valid example for a Data Flow Graph with stream of tokens is a hardware which performs audio/video processing. In video processing, often we need to apply Fast Fourier Transform algorithm (FFT) on the data. The input data required here is not a single value (token) but a stream of values (tokens) depending on the length of FFT. Example an FFT of length 128 would require a set of 128 values. Hence these systems are modelled using Data Flow Graphs with stream of tokens.

4.5 Data Flow Model for Re-configurable systems

The basic data flow model has been extended so that it can be used in designing and modeling reconfigurable systems. This extension allows non-directed or partially directed graphs. This property is a major source of differentiation from the other models. The models discussed above always had a directed graph with directed arcs, but this model might have non-directed or partially directed arcs.

This Data Flow Graph Model has special kind of nodes (configuration specifying nodes), and special kind of arcs (snoop arcs). The arcs represented with solid lines are the normal arcs which have similar usage as the arcs in the previous models. The arcs shown by the dotted lines represent the snoop arcs. Snoop arcs are used for gathering information about the states of the nodes and communicating these values to the configuration specifying node. The special type of nodes, also known as Configuration specifying nodes are used to gather information from the snoop arcs and based on this information they calculate the current state of the system. Once the current state of the system is calculated, it sends back this information to other nodes via the snoop arcs. This information is used to reconfigure the system dynamically at the run time.

Figure 6 shows a simplified example of this model. We do not discuss the functionality of this model, but this figure is used to give readers a general idea. In this figure, the circular node "CT" is the configuration specifying node, while the dotted lines represent the snoop arcs for transmission of information. The solid arcs are the normal arcs as in the previous models.

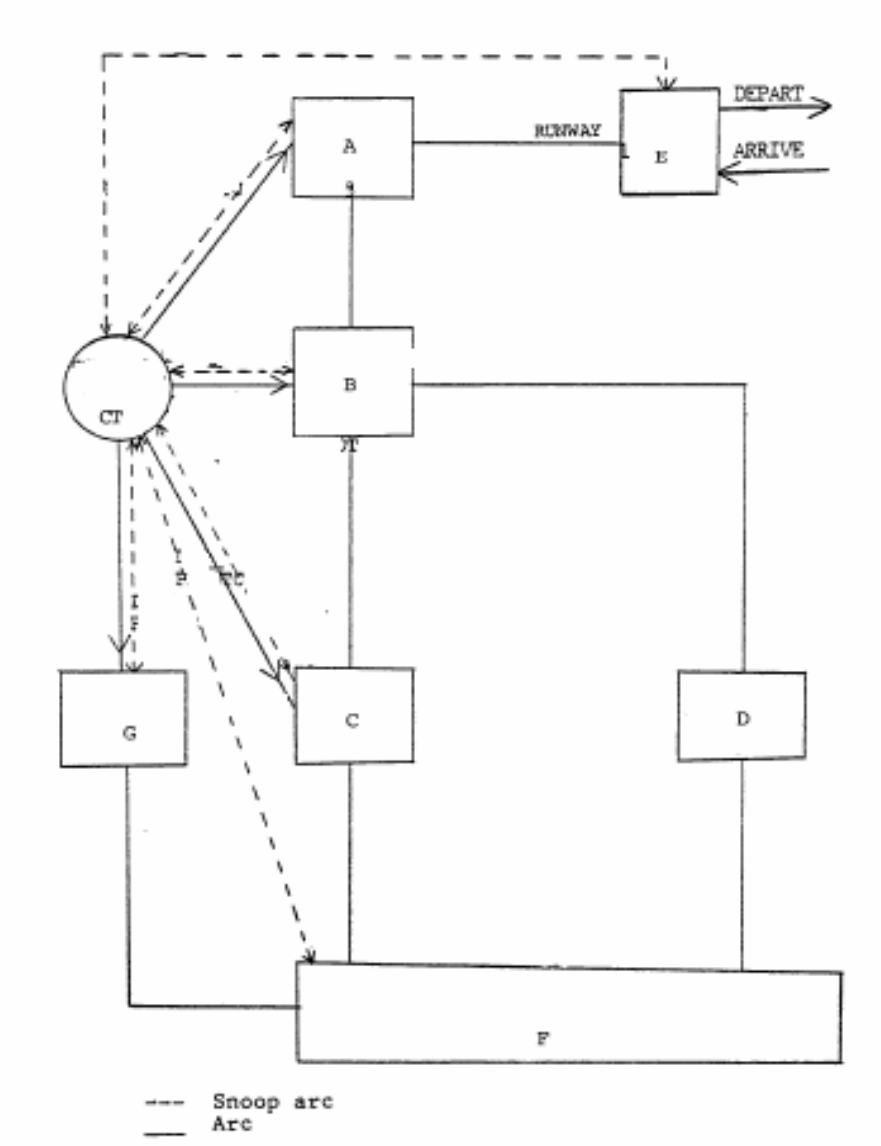


Figure 6

Each node can be assigned a macro operation or a micro operation. The computations in the operations can be nondeterministic; that is, random choice of actions at one or more points in the computation. This is due to the fact that there are configuration specifying nodes which change the configuration of the nodes, thus changing the operation being performed.

An extended abstract data flow graph is specified as a 4-tuple **(A, N, S, K),** in which

1. **A** is a set of arcs with attributes representing data paths.
2. **N** is a set of nodes with attributes representing activities.

3. **S** is a set of special arcs called snoop arcs for gathering information about states of nodes and communicating these values to the configuration specifying node.
4. **K** is a coloring scheme for the tokens on arcs to identify tokens belonging to different invocations of the graph.

5 Comparison of different Data Flow Graph Models

Much of the research in Data Flow programming has dealt with defining the functionality, designing instruction level architectures, or specifying programming methodologies. But not much effort has been put in the formalization of the Data Flow Graph model itself. This has essentially led to many definitions of Data Flow models and customized extensions to the model as per the researcher's likings. This often results in incompatible models used in various researches leading to non comparable results.

5.1 Common Features of Data Flow Graph

A Data Flow Graph is expressed as a directed graph where the arcs represent the data path and the nodes represent the operations to be performed on the data tokens arriving on the incoming arcs. A Data Flow Graph is shown in figure 7.

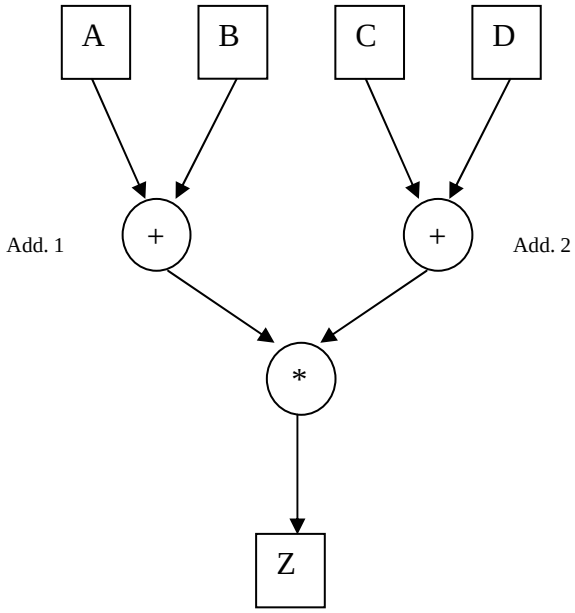


Figure 7

In the Data Flow Graph shown above, we can see that there are 4 basic inputs to the graph which are marked by 'A', 'B', 'C' and 'D' (external input nodes). These are always written in square boxes which act as the place holder for the values. The output is stored in 'Z' (external output node). The arcs are represented by the directed arrows, which also show the direction of the flow of data (tokens) in the Data Flow Graph. The intermediate results are stored on the arcs. That means the value calculated after the first addition operation i.e. "Add 1" is stored on the arc from this operation to the multiply " * " operation. Hence the arcs store all the intermediate results between two successive operations.

The availability of the input tokens and the availability of the output arc(s) to receive data are the only conditions which must be satisfied for any operation to

take place. The act of performing the operation is called firing the node and results in the consumption of the input tokens and production of output tokens.

In the Data Flow Graph shown in figure 7, 'A', 'B', 'C' and 'D' are the external inputs nodes. When the tokens from nodes 'A' and 'B' are available, the operation "Add 1" can fire which consumes these token and generates an intermediate result say token "IR1". Also as soon as tokens from nodes 'C' and 'D' are available, the operation "Add2" can also fire consuming the input tokens and producing "IR2" as the output token. As operations "Add1" and "Add2" have no data dependency, there's no restriction on the order of execution of these operations. They are fired as soon as the input tokens are available and there's some space on the output arc. If the input tokens became accessible at the same time then these two operations can be fired simultaneously. The intermediate tokens "IR1" and "IR2" act as the input to the multiply operation. Multiply operation cannot fire as long as both the tokens "IR1" and "IR2" are available. Once the multiply operation is fired we get an output token which is finally stored in the external output node 'Z'.

So from the above explanation of the Data Flow Graphs it can be easily seen that, since only the availability of data on input arcs and the ability of the output arcs to take more data is required for the node to fire, parallelism in algorithms is exhibited naturally. The order of execution is only restricted by the data dependencies among various operations.

6 Conclusion

We have studied many Data Flow Graph Models, and each of these models extends the functionality of the basic Data Flow Graph Model. Starting with the model where there are no intermediate nodes to store the results in between the operations we moved to the model where we have the special intermediate nodes to hold the results. A detailed study of these two models showed that they represent the same thing, but they are modelled in different ways. While in the first model it's the edges which stores the intermediate results, in the second model there are special nodes for this purpose. Then we moved to the next model with named tokens. Here the name of the token suggested the operation to be performed by the apply node, which would select an operation out of a set of operations depending on the name of the token. This provides a convenient way to do data dependent processing of data.

Next we studied the model with stream of tokens. The tokens in this model are represented as a stream of values. This extension is very helpful where the input to the operator is not a single value/token but a stream of values which have to be processed together or in a similar context. This model however needs a special marker to mark the end of the stream. The last model studied was for a re-configurable system where the graph itself is undirected or partially directed.

Later in the section 5, we striped all the extensions of the Data Flow Graph Models and we were left with the bare-minimum model. This model has features common to all the models and can be used as the base system for all other models to be built upon.

7 Literature

1. J. Backus, "Can programs be liberated from Von Neumann style? A functional style and its algebra of programs," CACM, pp 613-641, Aug 1978
2. M. Radetzki, "Modelling, Simulation and Specification", Lecture notes, Winter Semester 2005/2006.
3. M. Radetzki, "Embedded Systems Engineering", Lecture notes, Summer Semester 2006.
4. Arvind, Kim P. Gostelow and Wil Plouffe, "An Asynchronous Programming Language and Computing Machine", Department of Information and Computer Science, University of California, Irvine, California, Dec. 8, 1978.
5. King Song Weng, "Stream Oriented Computation in Recursive Data Flow Schema", Masters Thesis, Laboratory for Computer Science, Massachusetts Institute if Technology, 1975.
6. J.B. Dennis and D.P. Misunas, "A preliminary architecture for a basic data flow processor", Proceedings of the 2nd annual symposium on Computer Architecture, 1975.
7. J.E. Rumbaugh, "Data Flow Multiprocessor", IEEE TC, C-26, February 1977.
8. A.L. Davis, "The architecture and system method of DDM1: A recursively structured data driven machine", Proceedings of the 5th Annual Symposium on Computer Architecture, 1978.