Data-driven dataflow computer simulation

ECMP 424 Advanced Computer Architecture
Case Western Reserve University
Spring 1990

Davis's Data Driven Nets (DDN's) are a graph computation schema admitting a high degree of exploitable concurrency (both pipelined and horizontal.) As the name suggests, they are governed by a *data-driven firing rule* – a graph cell becomes fireable and is executed when all of its inputs are available.

In this assignment, you must construct a simulator for a machine which has DDN's as its machine language. The simulator must be constructed in a higher-level language such as C or PASCAL, but you should also consider the use of a language such as Prolog or LISP with built-in garbage collection and list management.

The test program for this assignment is a DDN that implements Newton's method for finding the root(s) of a function. The general equation for this approximation is:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

You should solve the following two cases:

$f(x) = 0$	$f'(x)$
$N - x^2$	$-2x$
$(x-1)^2$	$2x - 2$

The function and its derivative should be implemented as separate "subroutines" so that the DDN code for each function can be easily substituted into your program graph.

1 Data Driven Nets

A DDN program consists of one or more operator *cells* interconnected by *arcs*. The arcs carry data values (sometimes called *tokens*) from cell to cell. An output from a cell may be connected to more than one arc. In that case, a copy of any output token is passed along each of the arcs. The arcs act

Cell type	Purpose (activity)
Operator	Arithmetic, relational, Boolean (0/1) logic
Call	Invoke a DDN subgraph (subroutine)
Synch	Subroutine entry and exit, synchronization
Distribute	Route input to one of several outputs
Select	Route one of several inputs to single output
Arbiter	Deterministically merges multiple input streams
Gate	Loop control

Table 1: DDN cell types.

like FIFO queues, that is, tokens may accumulate in order of arrival at the input which is driven by an arc.

A cell has two parts to its description: a *firing rule* and an *activity rule*. The firing rule specifies the conditions under which the cell becomes fireable and may be scheduled for execution. The activity rule describes the action performed on cell inputs and any output tokens produced by the cell. When a cell fires, it removes all of the tokens from its firing set, executes the activity rule and puts tokens on its output arcs.

There are seven different DDN cell types (Table 1.) *Operator* cells are simplest and implement simple arithmetic, relational and logic functions. They have a *conjunctive* firing rule which says that a token must exist on all of the input arcs in order for the cell to fire. See Table 2 for a list of operator functions.

The *Call* cell permits a DDN to invoke a subnet. The subnet is itself a Data Driven Net and is purely input/output functional (i.e., has no side-effects and must be *clean* after execution.) The subnet has a single entry and exit formed by a *Synch* cell at its entry and a Synch cell at its exit. A Synch cell can have an arbitrary number of inputs. Its job is to wait until all of the inputs have a token, fire, and pass a copy of each input to a corresponding output. Thus, the firing rule is conjunctive.

The *Distribute* cell has two inputs (control and input value) and obeys the conjunctive firing rule. The control input must be a positive integer and is used to select a particular cell output. The token at the data input will be routed to the selected output. The outputs are numbered from left to right

Operator	Inputs	Function
NEG	1	Produces the negative of the input value
NOT	1	Produces the logical complement of each Boolean value
ABS	1	Produces the absolute value of the input value
ADD	2	Produces the sum of the two inputs
SUB	2	Produces difference (subtracts right from left)
MUL	2	Produces the product of the two inputs
DIV	2	Produces quotient (divides right into left)
LT	2	Produces true (1) if left is less than right
LE	2	Produces true if less than or equal
GT	2	Produces true if left greater than right
GE	2	Produces true if greater than or equal
EQ	2	Produces true if two inputs are equal
NE	2	Produces true if two inputs are unequal
MIN	2	Produces smaller of left or right input
MAX	2	Produces larger of left or right input

Table 2: Operator cells.

beginning at zero.

The *Select* cell implements a kind of "case expression." It has a single control input and two or more data value inputs. The cell becomes fireable when a non-negative integer value arrives on the control input and a token exists on the select data input. The data value at the selected input is routed to the single output. The inputs are numbered from zero, left to right.

The *Arbiter* cell has two or more data inputs and two outputs: control and output value. When an input token arrives at one of the inputs, the cell fires and puts the index of the input to the control output and the token to the value output. The inputs are number from zero, left to right.

The *Gate* cell is used to control the evaluattion of loops. It is the only cell type which is history sensitive (i.e., contains storage and its present behavior depends upon its previous execution history.) The Gate cell has three inputs (control, input value and feedback) and one output. Initially, the Gate cell waits for an input value. That value is copied to the output. If the control input is true (1), then the next output value is selected from the feedback input. If the control input is false (0) and there are no items on the value input, then the Gate cell goes back to its initial state. If an input value is waiting, it is copied to the output and the Gate proceeds as described.

2 Simulation.

The simulator must address several concerns. First, it must read and maintain an internal representation of the DDN program. Next, it must obtain initial DDN input values from the user (best handled interactively) and display outputs from the top level graph. The interpreter must move tokens (data values) from cell to cell via the arcs, detect and note fireable cells, and then select fireable cells for execution. The simplest mechanism here is to use an *agenda queue* for fireable cells. When the agenda queue is empty, either execution has terminated or the net needs additional inputs. As cells become fireable, they are added to the agenda queue. The simulator should pull fireable cells from the queue, perform the appropriate activity and keep the tokens moving (possibly copying output values to the input of more than one consumer cell.) Conceivably, if several processing elements are available, many cells can be executing simultaneously.