JET PROPULSION LABORATORY

INTEROFFICE MEMORANDUM
SFB: 363-89-001
31 May 1989

TO: Time Warp Folks
FROM: Steven Bellenot

The New GVT Algorithm

The new GVT algorithm is better distributed (scales better) and sends fewer messages than the old GVT algorithm but it is more complex and slightly less flexible. Initial performance data also seems to favor a change to the new GVT algorithm. Let N be the number of executing nodes (or processors). The new GVT algorithm has a run time of O(1) in each node and overall run time of O(log N) vs a run time of O(N) for the old GVT algorithm (on node 0 and overall). The new GVT algorithm sends less than 4N messages vs the old GVT algorithm which sends 5N. The new GVT algorithm is more complex in the sense that a message routing graph is constructed at run time. Each node configures itself at gvtinit time (before the simulation starts) and determines who it sends messages to and who it receives messages from. (The configuration algorithm has a run time of O(log N) on each node.) The new GVT algorithm is less flexible in the sense that it requires the nodes to be (logically) numbered from 0 to N - 1 and only node 0 can start a GVT calculation. (Although not required by the old GVT algorithm, these conditions have always been true for our Time Warp implementation.)

The Message Routing Graph

The message routing graph is different for different values of N. To aid the reader we have included both graphical representations of these graphs for values of N less than 18 (Figure 1) and the code each node executes to configure itself (Figure 2). The message routing graph is basically two binary trees and some connecting arcs. There are three cases to consider in constructing the message graph. We consider the easiest case first. Suppose N = 2M so that N is even, then the basic construction goes as follows. Nodes 0 through M - 1 form a binary tree with 0 at the root and the arcs pointed away from the root. Nodes M through N - 1 form a binary tree with N - 1 at the root and the arcs pointed toward the root. Finally there is an arc from node i on the first tree to node N - 1 - i on the second tree if node i has no children. (See for example N = 14, 12, 6 in Figure 1.) If N is odd, then the middle node is considered part of both

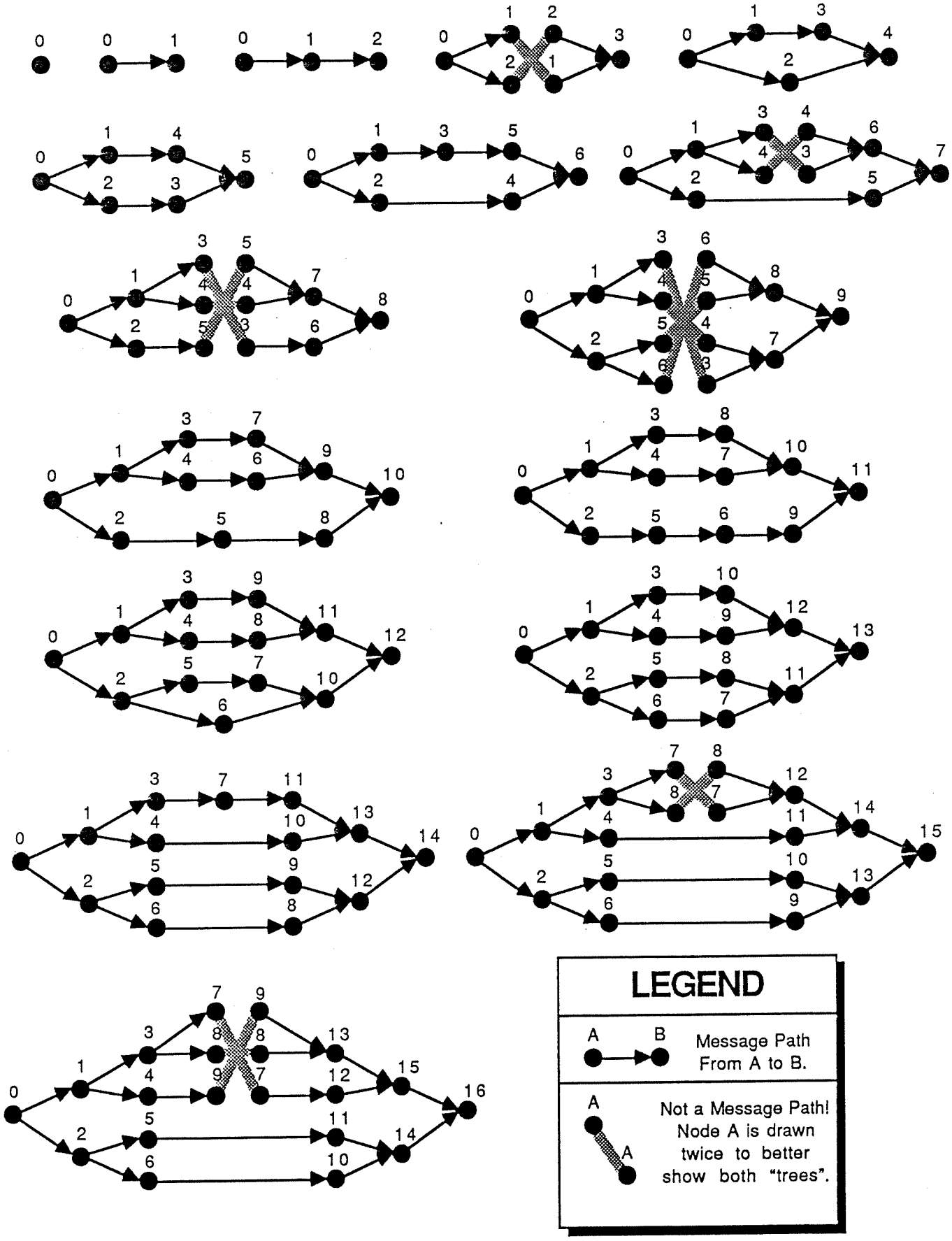

Figure 1. Message Routing Graphs for 17 and fewer nodes.

```
int numFrom, numTo, from[2], to[2],
int Out0, Out1, numArrive;

/* my node number */
int myNum;

/* total number of nodes */
int numNodes;

FUNCTION gvtcfg()
{
    int Mid0, Mid1, myInv, OutFrom;
    int In0, In1, InTo, pen0, pen1;

    Mid0 = ( numNodes - 1 )/2;
    Mid1 = ( numNodes & 0x01 )?
        Mid0 : Mid0 + 1;

/* matching node on other tree */
    myInv = numNodes - myNum - 1;

/* for binary tree with root 0 */
    /* left child number */
    Out0 = 2 * myNum + 1;
    /* right child number */
    Out1 = Out0 + 1;
    /* parent number */
    OutFrom = ( myNum - 1 )/2;

/* for binary tree with root
    numNodes - 1 */
    /* left child number */
    In0 = 2 * myNum - numNodes;
    /* right child number */
    In1 = In0 - 1;
    /* parent number */
    InTo = (numNodes + myNum + 1)/2;

    for ( pen0 = 1; pen0 < Mid0 + 1;
        pen0 = 2 * pen0 + 1 )
    {
/* this is the only O(log N) part,
everything else is O(1) */
      ;
    }
    pen0 = pen0/2 - 1;
    pen1 = numNodes - pen0 -1;

    if ( pen1 <= 2 * pen0 + 3 )
/* the two trees can share the
bottom row */
    {
      Mid0 = pen1 - 1;
      Mid1 = pen0 + 1;
    }

    if ( myNum == 0 )
    {
      numFrom = 0;
    }
    else if ( myNum <= Mid0 )
    {
      numFrom = 1;
      from[0] = OutFrom;
    }
    else if ( Mid1 <= In1 )
    {
      numFrom = 2;
      from[0] = In0;
      from[1] = In1;
    }
    else if ( Mid1 == In0 )
    {
      numFrom = 1;
      from[0] = In0;
    }
    else
    {
      numFrom = 1;
      from[0] = myInv;
    }

    if ( myNum == numNodes - 1 )
    {
      numTo = 0;
    }
    else if ( myNum >= Mid1 )
    {
      numTo = 1;
      to[0] = InTo;
    }
    else if ( Mid0 >= Out1 )
    {
      numTo = 2;
      to[0] = Out0;
      to[1] = Out1;
    }
    else if ( Mid0 == Out0 )
    {
      numTo = 1;
      to[0] = Out0;
    }
    else
    {
      numTo = 1;
      to[0] = myInv;
    }

}
```

Figure 2. The configuration code to make the local part of the message routing graph.

trees. (See for example N = 15, 13, 11, 7, 5 in Figure 1.) Finally if the number of nodes on the "bottom row" of each tree is less than half the number which could fit on the bottom row, then both trees share all the bottom row nodes. (See for example N = 17, 16, 10, 9, 8 in Figure 1.)

The local information needed from the message routing graph is stored in the variables numFrom, numTo, from[2] and to[2] and a variable numArrive is used to count the number of received messages. (The variables Out0 and Out1 are used in the update routine.) We note for later use that we always have numTo + numFrom <= 3.

The Algorithm

A GVT calculation starts on node 0 with the interrupt routine gvtinterrupt. Gvtinterrupt just calls gvtstart and then calls dispatch. All nodes receive a call to gvtstart, all but node 0 by the receipt of a GVTSTART message.

The gvtstart routine first checks to see if this node has received all its GVTSTART messages. (If numFrom equals two it needs both messages, otherwise one is enough.) Next the gvtstart routine starts the minimizing of the virtual times of messages in transit. Then if numTo is one or two, gvtstart sends GVTSTART messages to the node to[0] and, if needed, to the node to[1]. Finally, if the node is number N - 1, then numTo is zero and at this point all of the nodes have started minimizing virtual times of messages in transit, so gvtstart calls the routine gvtlvt with the virtual time parameter of POSINF + 1.

The gvtlvt routine uses the message routing graph in the opposite direction. All nodes but N - 1 call gvtlvt with the receipt of a GVTLVT message. If numArrive is zero, the gvtlvt routine stops the message minimizing on this node. Then gvtlvt takes the minimum of its old value of lvt with the parameter it is called with. Next gvtlvt checks to see if has received all its GVTLVT messages. (If numTo equals two it needs both messages, otherwise one is enough.) Then if numFrom is one or two, gvtlvt sends GVTLVT messages (with its estimate of GVT) to the nodes from[0] and from[1] if needed. Finally, if the node is number 0, then numFrom is zero and at this point node 0 has the new estimate of GVT, so gvtlvt calls gvtupdate with the new GVT.

The gvtupdate routine sends GVTUPDATE messages with the new GVT to the nodes Out0 and Out1 if these numbers are strictly less than N. (This is a binary tree message routing graph using all the nodes with node

0 at the root and the arcs pointing away from the root.) Then gvtupdate does those things that a node must do after receiving a new GVT.

Mathematical Analysis

Since the message routing graph uses all the arcs once in each direction and the degree of each node is <= 3 and some are < 3, the number of GVTSTART plus the number of GVTLVT messages is < 3N. Also there are N - 1 GVTUPDATE messages. Thus the new GVT algorithm uses < 4N messages. Because of the binary trees in the message routing graph, the longest path from node 0 to node N - 1 is O(log N).

Code Location

The code for the new GVT algorithm is in gvt2.c, which could replace gvt.c with version 2.1. (It is currently included in libtw.a in the directory /usr/local/src/bbn_2.1.) The code has been tested on all number of nodes from one to twenty.

Initial Performance Data

The performance data was done using various Time Warps all near version 2.1. In each comparison, the only difference is that between gvt.c and gvt2.c. However, Time Warp 2.1 evolved during this time and other comparisons are suspect. The time difference between the interrupt on node 0 till the time node 0 has the new GVT will be called a *GVT calculation time*. Note that GVT calculation time is more a measure of message delays than the run time of gvt code. Also note that the time to spread the new GVT from node 0 to the rest of the nodes is not included in the GVT calculation time. This interval also has nothing to do with the timval command in the configuration file.

1. Stb88 16 nodes before message priority fix. (The run times were roughly the same.) The configuration file was stb88.cfg.16.

GVT calculation times				
file	average	best 90%	worst 10%	worst times
gvt.c	.423 sec	.272 sec	1.76 sec	7.14 sec 5.84 sec
gvt2.c	.394 sec	.200 sec	2.14 sec	9.87 sec 5.92 sec

2. Stb88 4 nodes before message priority fix. The configuration file was stb88.cfg.16.

The usual stats					
file	run num	run time	neg msgs	rev msgs	
gvt.c	1	4123.79	76087	163806	
gvt.c	2	3703.50	64632	130773	
gvt2.c	1	1870.99	4752	42	
gvt2.c	2	1869.64	4816	11	
GVT calculation times					
file	run num	average	best 90%	worst 10%	worst time
gvt.c	1	2.89 sec	2.43 sec	7.00 sec	8.13 sec
gvt.c	2	2.48 sec	1.99 sec	6.90 sec	7.90 sec
gvt2.c	1	.127 sec	.077 sec	.579 sec	8.41 sec
gvt2.c	2	.127 sec	.077 sec	.576 sec	7.73 sec

3. Stb88 4 nodes after message priority fix. The configuration file was stb88.cfg.16.

The usual stats				
file	run time	neg msgs	rev msgs	
gvt.c	2074.65	14813	8856	
gvt2.c	1843.26	4814	0	
GVT calculation times				
file	average	best 90%	worst 10%	worst times
gvt.c	.134 sec	.106 sec	1.23 sec	1.58 sec 1.51 sec
gvt2.c	.085 sec	.065 sec	.273 sec	1.03 sec 1.00 sec

4 Stb88 16 nodes after message priority fix. (The run times were near the same.) The configuration file was stb88.cfg.16. (Figure 3 shows a histogram of these values.)

GVT calculation times					
file	run num	average	best 90%	worst 10%	worst time
gvt.c	1	.248 sec	.189 sec	.776 sec	3.35 sec
gvt.c	2	.263 sec	.188 sec	.957 sec	3.10 sec
gvt2.c	1	.174 sec	.118 sec	.671 sec	2.09 sec
gvt2.c	2	.174 sec	.117 sec	.678 sec	1.66 sec

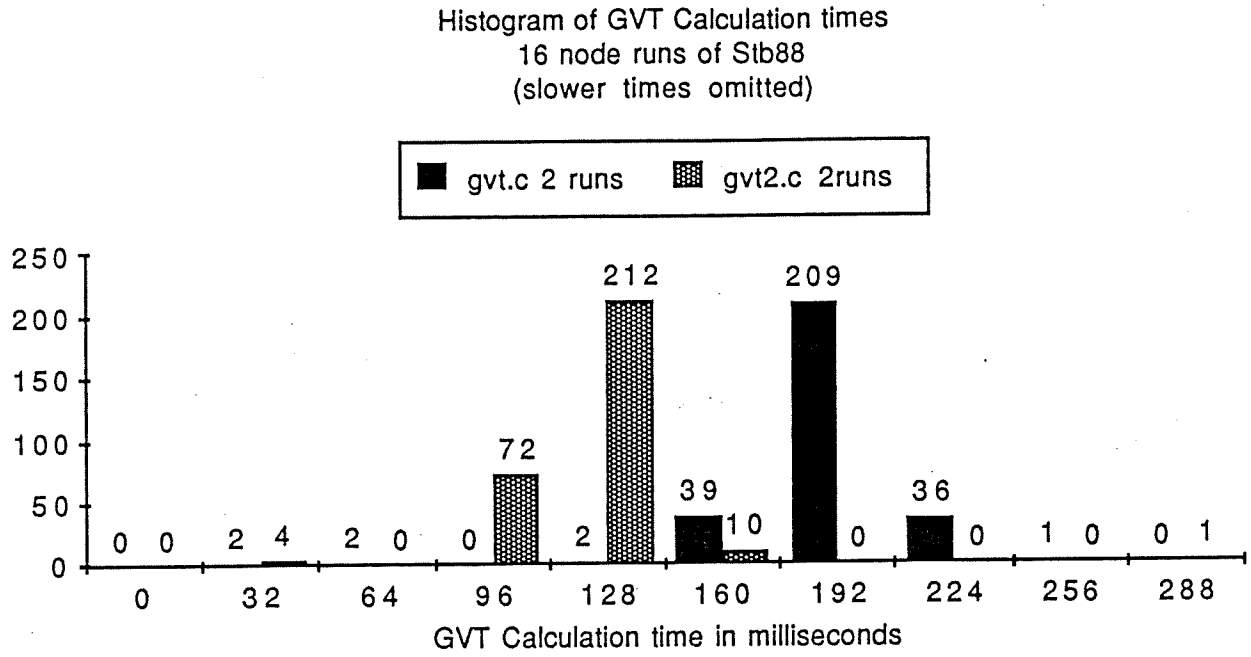

Figure 3. Histgram of GVT Calculation times.

Conclusions

The new GVT algorithm seems to perform better on as few as four nodes, and since it scales better, it should do at least as well as the old GVT algorithm on all number of nodes. However, this doesn't mean the new GVT will speed up the run time of all current simulations in all configurations. But it should not slow any of them down.

TIME WARP LIST

B. BECKMAN
L. BLUME
M. DILORETO
A. FEINBERG
J. GIESELMAN
L. HAWLEY
P. HONTALAS
B. JACOBSON
D. JEFFERSON
B. MILLER
B. MOORE
G. PAINE
M. PRESLEY
P. REIHER
J. RUFFLES
J. TUPMAN
V. WARREN
J. WEDEL
F. WIELAND
H. YOUNGER

TIME WARP DISTRIBUTION

BECKMAN
BELLENOT
BLUME
DILORETO
FEINBERG
FUJIMOTO
GIESELMAN
HAWLEY
HONTALAS
JACOBSON
JEFFERSON
MILLER
MOORE
PAINE
PRESLEY
REIHER
RUFFLES
TUPMAN
WARREN
WEDEL
WIELAND
YOUNGER