JET PROPULSION LABORATORY

INTEROFFICE MEMORANDUM
SFB: 366-90-01
August 7, 1990

To: Time Warp Fans
From: Steve Bellenot
Re: Mach and Time Warp

Althought Time Warp 2.4 supports the BBN Butterfly GP1000 running Mach, there are a number of ways which this Mach support can be improved. Some of these improvements are currently available in the files BF_MACH_Hg2.c and BF_MACHrun2.c, other improvements would require implementation and finally there is the paging problem which has not been solved under Mach.

1. BF_MACHrun.c variables:

The Mach version of Time Warp incorporated the "run program" into the Time Warp executable. Before Mach, a separate run program was used to start up Time Warp on the other nodes and to collect certain data messages from the simulation, for examples msglog, qlog, flowlog and is_log. By linking the run program with Time Warp, every Time Warp node got a copy of the variables in BF_MACHrun.c, even though the Time Warp nodes have no use for these variables. When the run program was running on its own node, it had lots of available memory and hence the data space for the run program had grown to a huge size. By replacing a large array variable requiring B bytes for storage with pointer and then mallocing the needed array removes B - 4 bytes from the data space, (4 bytes are needed for the pointer).

After adding this change to BF_MACHrun.c, the data segment of Time Warp was reduced by 32 pages which is a quarter of a megabyte per node. (Let's see ... we have 84 nodes, so that is 21 MB total, and at the going rates a MB is about $100, so we found about two thousand dollars.) Memory is very dear under Mach. The main Time Warp heap is 2.75 MB in the Chrysalis 2.4 version and this heap was 1.75 MB for the 2.4 benchmark of the Mach version. (We think we know where another 0.5 MB went under Mach. Unix has a constant called LOTSFREE. If there is LOTSFREE amount of free memory, then the pager is not run under Unix. Mach also has this constant LOTSFREE which is set to 0.5 MB. We think Mach works like Unix in this case.)

Two additional improvements were made while implementing "malloced run program variables". First an addition 5 pages of data space

was removed from pucks. The circles package had large arrays used for testing circles which was not used for pucks. (The data segment for pucks went from 50 pages to 13 pages with all these improvements.) Second bugs in the way the run program collected the data were discovered. A data collection with zero items of data would crash the program (and this happened after the simulation part of the run had been completed.) One of the loging collection was found to hang. These bugs were fixed.

These improvements can be added to Time Warp 2.4.1 without those listed in 2 below.

2. The Uniform System:

The Mach version of Time Warp 2.4 uses the Uniform System. The Uniform System is a collection of routines which are available under both Chrysalis and Mach. In either case, there is a library of routines which in turn will call the native operating system on your programs behalf. Since the Chrysalis version of Time Warp does not used the Uniform System, and the Mach version cannot run under Chrysalis, we do not gain any portability by using the Uniform System. There are some advantages to removing the Uniform System from Time Warp. There are modest improvements in time and space, but the main improvement is the simplifiction of removing a software layer.

In the Mach version of 2.4, there are five different node numberings. Time Warp node N is on Uniform System node N + 1 is on Cluster Logical node N + 1 is on some Mach Logical node M which is some Physical node P. (Making the Uniform System nodes match the Cluster Logical nodes was a last minute addition to 2.4.) In removing the Uniform System, Time Warp node N now is Cluster Logical node N. (Also the XL_STATS file now contains a list of which Mach Logical and which Physical node each Time Warp node was running on.) This simplifies the send and receive routines, the node numbers are no longer one off.

The Uniform System on Mach isn't designed for Time Warp. It forks processes to all the nodes before Time Warp can allocate its shared variables (queues and message buffers), hence all the calls to "Share" to update these variables. Without the Uniform System the shared areas are allocated before the fork so that each process automatically gets the correct data at fork time. (The fork command creates a new process with an identical data segment.) The Uniform System also manages to sometimes kill the parent process before its children have exited. This isn't an error so much it is an annoyance, when a parentless process exits an error line is printed to the screen. A 60 node run would sometimes get 60 identical error messages, scrolling the screen pass information you might want to see. This is fixed in BF_MACHrun2.c

The removal of the Uniform System required adding some delay fuctions. The routine wait_a_millisecond() has advantages over the UsWait() which makes calls to getrtc(). Since wait_a_millisecond() just decrements a register, the CPU is not making any memory references once the instruction loop is cached. Calls to getrtc() go over the switch, increasing switch traffic. The other delay function is in lock() and is timed to be about 40 micro-seconds.

Two additional improvements were made while implementing these delay functions. First two "divisions" were replaced with "compares" from the enqueue and dequeue routines, speeding up the time between the lock and unlock calls. Second, since only one process dequeues from any particular queue, dequeuing doesn't need to done with locks, dequeuing can go on in parallel with enqueuing.

Other improvements in BF_MACH_Hg2.c include better feedback when initializing Time Warp. While creating processes (and exiting processes) a period is printed as each process is started (ended). Time Warp 2.4 claims to get the number of message buffers off the command line, however this didn't work and the number of message buffers was always 64. This is fixed in BF_MACHrun2.c and BF_MACH_Hg2.c which allows the user to set the number of buffers between 16 and 64. Also code to prevent max_acks from getting larger than the number of buffers needed to be added. BF_MACH_Hg2.c is set up so that it is easy to wire down the message buffers, unfortunately this currently causes the butterfly to crash when it is releasing these wired pages.

By not including the Uniform System code, the size of the text segment (code) is reduced by about 18 KB which is a couple of pages. For a 60 node pucks run with 2.0 MB of memory the run times averaged 2% to 3% faster (3 to 4 seconds) using BF_MACHrun2.c and BF_MACH_Hg2.c over ones that used the Uniform System. BF_MACHrun2.c includes the fixes in 1 above.

Both BF_MACHrun2.c and BF_MACH_Hg2.c are ready to be merged into Time Warp 2.4.1.

3. Mach Threads:

The file BF_MACH_Hg.c get its name by making the Mach send and receive routines look like the Mercury send and receive routines to the software above it. BF_MACH_Hg.c actually uses Mach to act like Chrysalis did in BBN_Hg.c (now BF_PLUS_Hg.c) by implementing "DualQueues". It seems possible to use Mach's send and receive primitives to directly implement the Mercury routines. Moreover, it seems reasonable to use Mach threads to obtain an interrupt routine for arriving messages. Unfortunately, the Butterfly's Mach isn't that kind of Mach. All of these

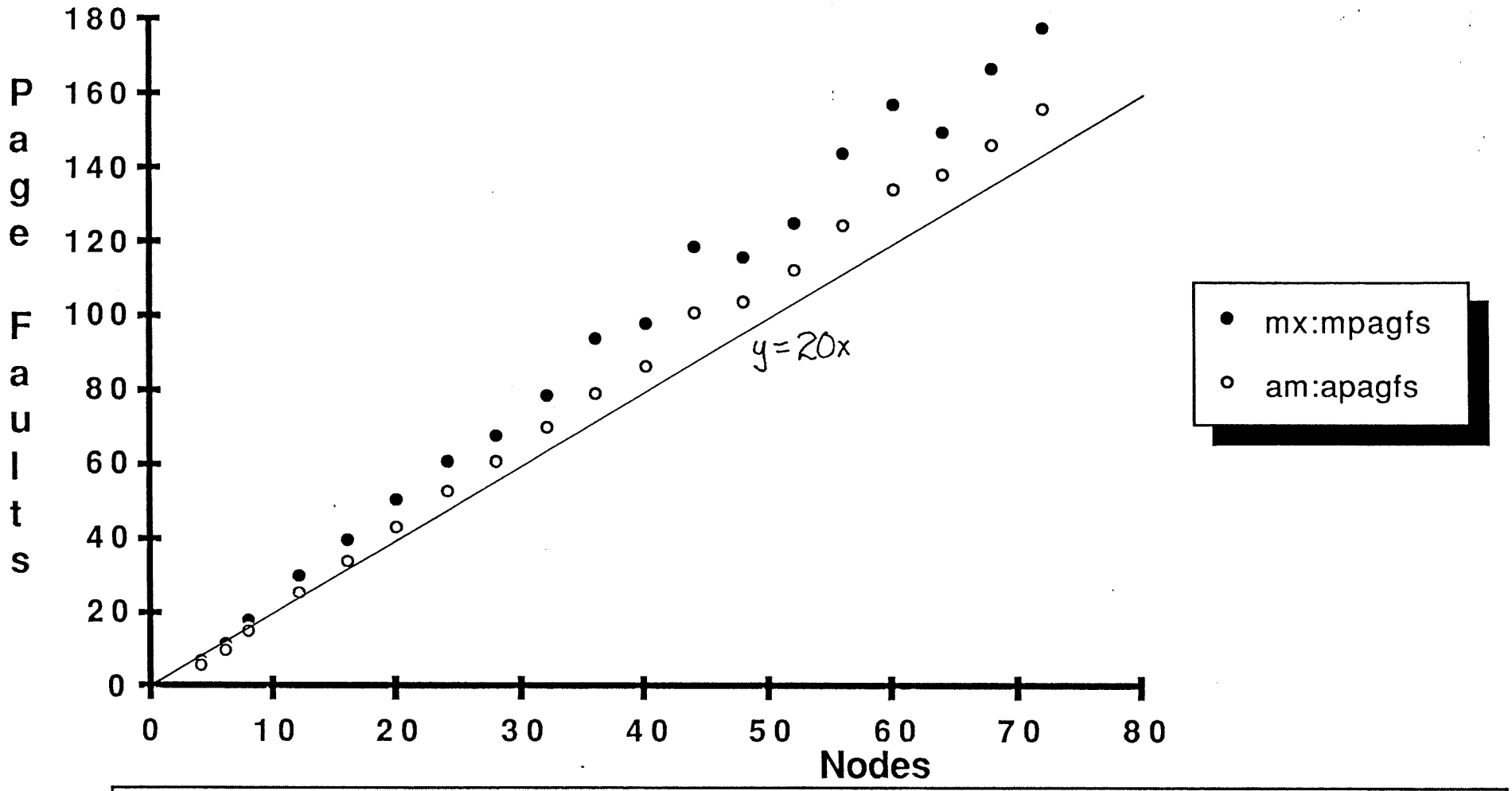
routines are missing. Mach is not Unix and the Butterfly Mach isn't even Mach.

4. Page Faults:

The page fault problem has not been solved. The enclosed graph (from the 2.4 benchmark memo) shows pages faults per node versus number of nodes. The average page faults points is slightly above the line faults = 20 * nodes. Futhermore these page faults cause more harm to the higher number of nodes case. (Shorter run times make initial faults easier to see.) Graphs of message logs (using mplot) show this slow start for messages. Interestingly enough these slow times are also true for gvt messages, and these are the second round of gvt messages. (There is some evidence that having too many message buffers is what is causing gvt messages to be slow. The min_messages routine must touch each message buffer, and most of them have not been touched since the last gvt.)

A number of experiments have been devised to find out where and why we are page faulting so much. One test lead to the discovery of the quarter Meg in the run program. Three others crash the butterfly on a regular basis. Other tests, which work on Unix, seem not to work for Mach. Although Mach has many of the same kernel variables as Unix, it seems to use them differently.

Pucks Average and Max Fault Curve



TW 2.4 Benchmark/Pucks Page Faults/4 runs per point/Butterfly (MACH)/EB/7-11-90