THE DINING PHILOSOPHERS BENCHMARK

The "dining philosophers problem" is a well known example from the theory of concurrency. As a simulation to be run using Time Warp, the dining philosophers can be used as a benchmark to obtain estimates on the communication costs in several ways. Brian Beckman (JPL IOM 335.1-295, 13 Feb 1985) suggested that the dining philosophers would be a good candidate for Time Warp. However, our objects are somewhat different from those he suggested.

The dining philosophers simulation has a "natural mode" which is perfectly load balanced (at least at first glance). That is, any two objects of the same object type, either fork type or philosopher type, have uniform CPU time requirements. Thus perfect load balance is obtained by giving each node the same number of instances of both fork and philosopher objects. Hence simulation costs that are caused by an imbalanced load are theoretically missing in the "natural mode".

BENCHMARK TESTS:

1. Varying the communications / computation ratio: The philosopher objects have a delay parameter (which is passed with the initialization message in the cfg file) which determines the amount of real time needed to complete a cycle. Since "doing nothing" is very concurrent, if these delays are large, then we should see linear speed-up. On the other hand, if the delays are near zero, then essentially the costs are all in communications and speed-up should not be anywhere as good. Perhaps an overload threshold of communication / computation can be found, where simulations whose ratio is greater than this threshold will have poor speed-ups.

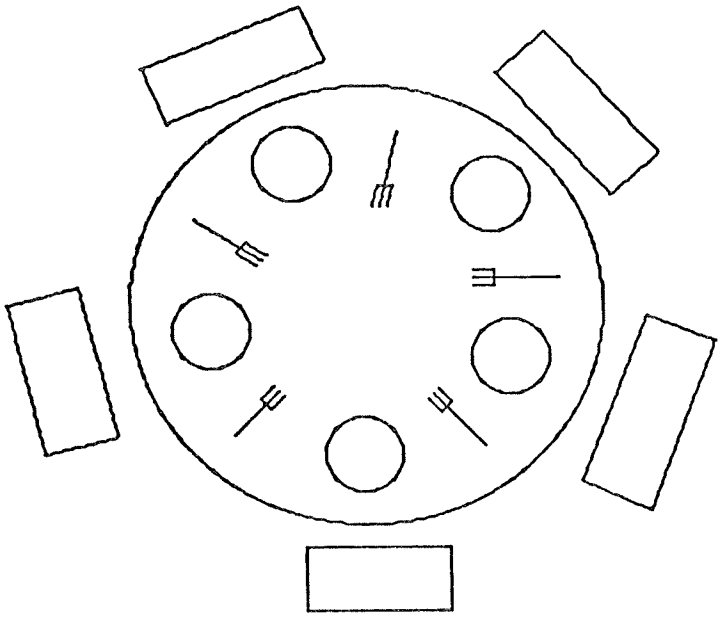
2. Varying message length: The messages used by the simulation are in the separate file "dinphmsg.h". As long as the first letter is the same as given, the message length can be arbitrary increased (all messages must be the same length). Both of the objects' code must be recompiled to effect such changes. Thus a measure of the effect of message size on the simulation run time can be made.

3. Varying the internode distance that messages travel within the hypercube: The program configme.c will interactively construct a cfg file where objects that communicate are, on the average, a given number of hypercube jumps apart. This will estimate the importance of putting two objects which talk to each other near to each other in the hypercube. (The program configme.c isn't written yet but its construction is outlined below.)

4. Unnatural mode costs: Although we haven't automated it, one can obtain some estimates on the costs of unbalanced loads with the dining philosophers as well. Making philosophers on one half of the table run twice as fast as the other philosophers should increase the amount of rollbacks and might run a good deal slower than if all the philosphers were running in the slower mode.

Statement of the dining philosophers problem:

There are several (say five) philosophers sitting around a table. Philosophers alternate between eating and thinking. To eat, a philosopher must get two forks, one on each side of him. Since there is only one fork between neighboring philosophers, when philosopher A is eating, neither of his neighbors can eat. See the diagram below. (In some statements of the problem, the forks are replaced with chopsticks.) A solution to this problem must be deadlock-free (i.e. all the philosophers can't have one fork while waiting for the other) and treat each fork as a critical region (i.e. if two philosophers reach for the same fork at the same time, then one and only one philosopher gets the fork).



THE PHILOSOPHER OBJECT:

The philosopher object has six "states" as shown in the diagram below. Except for the initial message which is received in the UNINIT state, the philosopher never reads his messages. The philosopher goes into the DEAD state after MAX_NUM_OF_THOUGHTS cycles through the loop: THINK, HUN0 (hungry with zero forks), HUN1 (hungry with one fork), EAT.

UNINIT: Reads the initial message to find the object names of his two forks and the delay factor, then changes to THINK state.

THINK: (Assumes the event message is a MSG_WAKE_UP.) He increments num_of_thoughts, requests the fork on one side and changes to HUN0 state.

HUN0: (Assumes the event message is a MSG_GRANTED from the fork.) The object requests the other fork and changes to HUN1.

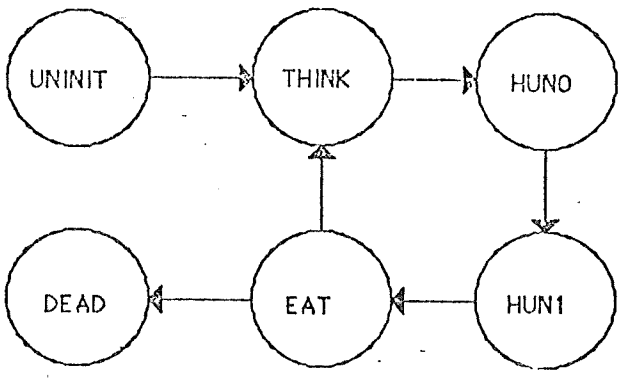
HUN1: (Assumes the event message is a MSG_GRANTED from the second fork.) The object has both forks and changes to EAT, but first sends a MSG_WAKE_UP to himself so he will know when to stop eating.

EAT: (Assumes the event message is a MSG_WAKE_UP.) The object releases both forks, changes to DEAD if num_of_thoughts >= MAX_NUM_OF_THOUGHTS, else changes to THINK and sends a MSG_WAKE_UP to himself so he will know when to stop thinking.

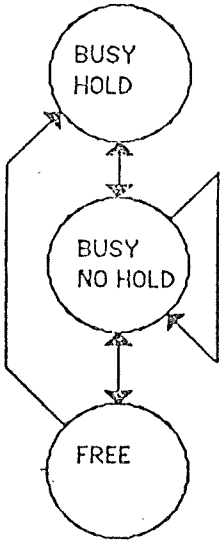
DEAD: Do nothing. (The object gets no messages in this state.)

Some predefined constants:

```
10              /* Reach Time */
THINK_TIME
EAT_LENGTH
START_TIME
```



Finite state diagram for the philosophers



Finite state diagram for the forks

THE FORK OBJECT:

The fork object acts like a binary semaphore. However its wait queue only has room for one philosopher. (All that is needed for this problem.) The finite state diagram for the fork is given above. The fork starts out in the FREE state. Note that the fork can receive two event messages at the same time and they both could be requests (MSG_WAIT). Care has been taken so that the same philosopher will win this race no matter which order Time Warp uses for these messages.

FREE: Fork currently idle.
BUSY_NO_HOLD: Fork is in use by one philosopher, but the other philosopher hasn't requested the fork.
BUSY_HOLD: Fork is in use and there is a philosopher waiting for it too.

Some predefined constants:

```
1       /* Time it takes for a fork to reply */
```

FORM OF THE INITIAL MESSAGES:

The Fork needs no initial message (in fact such a message could introduce a fatal error). On the otherhand, the Philosopher's initial message is critical. The Philosopher's initial message basically tells him the name of his two forks and optionally, how much to delay while thinking. It is assumed that the name of all Forks start with 'F' and the initial message ends with a '\0'. The general form is either:

"FxxxxxFyyyy" or "dFxxxxxxxFyyy"

where 'd' is the delay factor (given by (int)('d' - 'A') or zero if d is missing), Fxx is the name of fork0 and Fyyyyy is the name of fork1. (Note that a delay of 'F' - 'A' is not possible.)

DEADLOCK PREVENTION:

The order in which the Forks are given in the initial message must take deadlock into account. Since the Philosopher always asks first for fork0 and holds it until he gets fork1, deadlock is a real possibility. A suggested format is that fork0 should always be "even" and fork1 "odd", not only will this prevent deadlock, but it will also smooth the initial behavior of the simulation.

CONFIGME ALGORITHM:

In general, we want to put p Philosophers per node on a d dimensional hypercube with the "average" message travel of j hops (the number of node to node communications between the source and destination of the message). Here there are n = p * 2**d Philosophers and Forks both numbered 0 through n - 1.

Philosophers i*n through (i + 1) * n - 1 go onto node k = gray (i). The function gray can be defined as:

gray(i) int i; { int is; is = i >> 1; return i ^ is; }.

Forks i*n through (i + 1)*n - 1 go onto node k ^ (2**j - 1).

MEMORY COSTS:

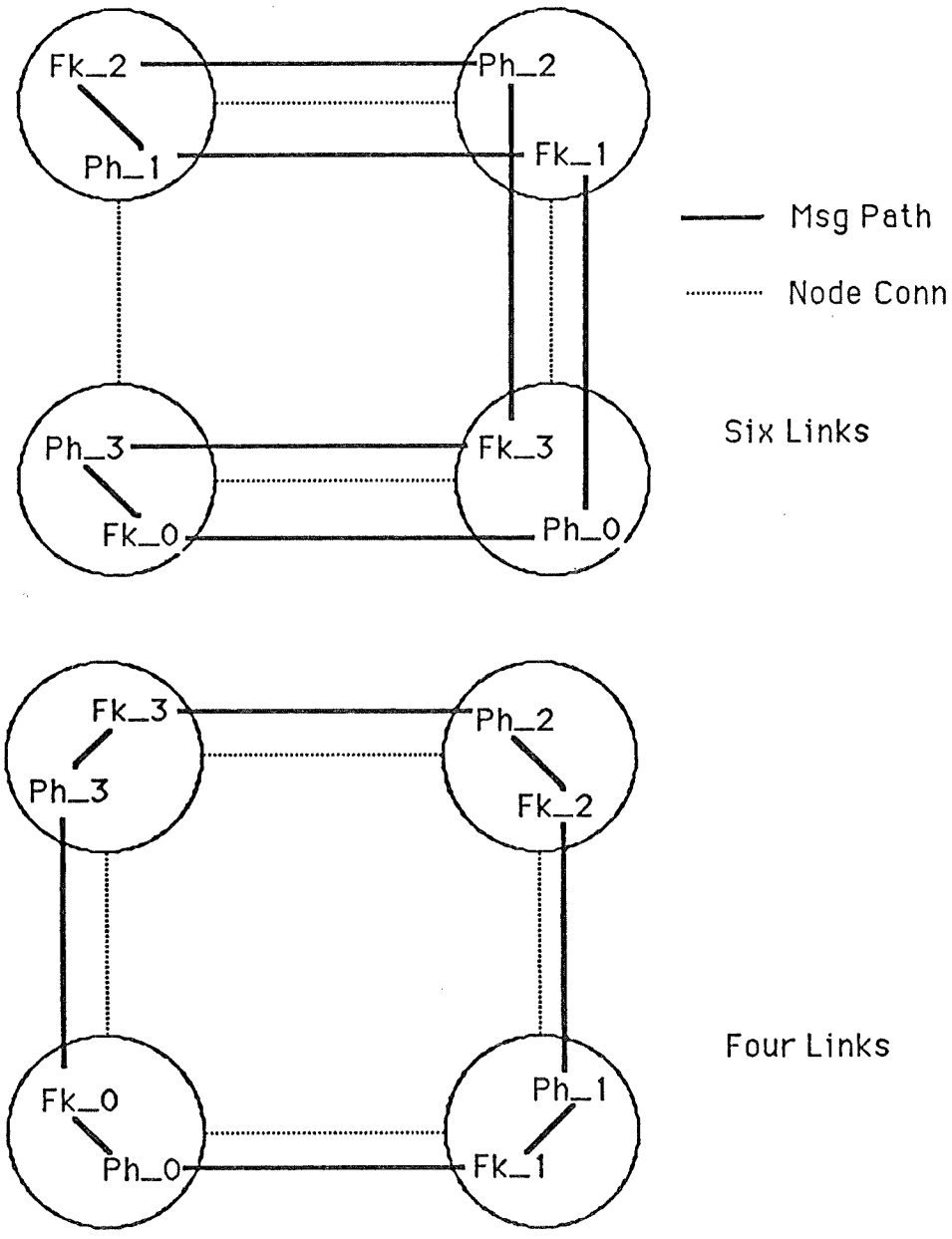
The memory costs of different sized messages has been designed so this cost is just the amount of memory needed to store the message. That is there is only one place in the code for each of the five messages. However, a long message will make the code section longer and thus decrease the memory for messages and states. Perhaps keeping the messages long and just varying the #defined variable MSG_LEN will give a better measure of the cost of long messages on simulation speed.

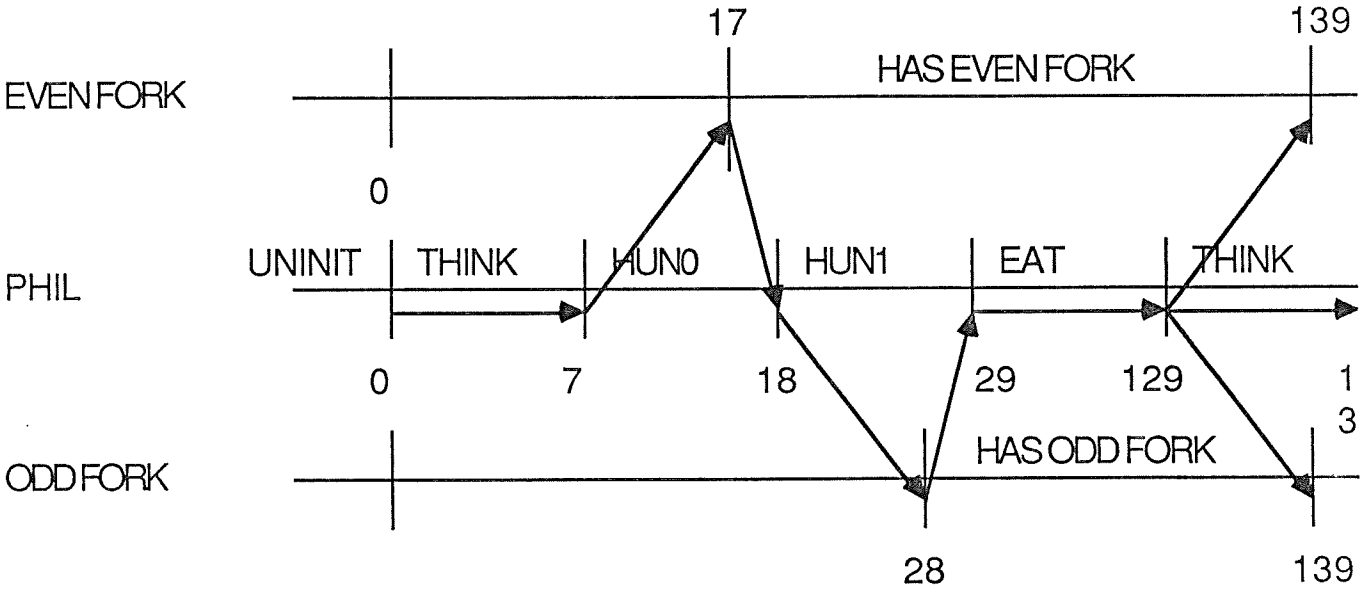
AN EXAMPLE:

The figure on the next page shows two ways of assigning objects to nodes for a Dining Philosophers simulation. The bottom one is obviously the best possible. The top one has the interesting property that interchanging any two objects of the same type will increase the commucation costs. That is, the top assignment is in some sense a local best assignment which isn't the global best assignment.

SOURCE FILES:

The .c and .h files for the Dining Philosopher simulation are in the GaSSVax::mda2:[bellenot.twsim] directory.


Fk_2
Ph_2
Ph_1
Fk_1
Msg Path
Node Conn
Ph_3
Fk_3
Fk_0
Ph_0
Six Links
Fk_3
Ph_2
Ph_3
Fk_2
Fk_0
Ph_1
Ph_0
Fk_1
Four Links



GENERAL TIMELINE FOR THE PHILOSOPHER OBJECT.
This assumes there is no competition for the forks. A busy fork would not return the message until the fork was free.
The numbers 131? and 329? depend on the constant THINK_TIME which was 200 in the simulater tests and 2 for the mike tests.
This pattern is repeated MAX_NUM_OF_THOUGHTS times (either 5, 10, 15 or 20). However the first "THINK" is for START_TIME = 7, and isn't part of the repeating pattern.