Introduction

- Reading
  - Today MPI & OpenMP papers
  - Tuesday “Commutativity Analysis” & HPF

Programming Assignment Notes

- Assume that memory is limited
  - don’t replicate the board on all nodes
- Need to provide load balancing
  - goal is to speed computation
  - must trade off
    - communication costs of load balancing
    - computation costs of making choices
    - benefit of having similar amounts of work for each processor
- Consider “back of the envelop” calculations
  - how fast can pvm move data?
  - what is the update time for local cells?
  - how big does the board need to be to see speedups?

PVM Group Operations

- Group is the unit of communication
  - a collection of one or more processes
  - processes join group with pvm_joingroup(“<group name>“)
  - each process in the group has a unique id
    - pvm_gettid(“<group name>“)
- Barrier
  - can involve a subset of the processes in the group
  - pvm_barrier(“<group name>“, count)
- Reduction Operations
  - pvm_reduce(void (*func)(), void *data, int count, int datatype, int msgtag, char *group, int rootinst)
    - result is returned to rootinst node
    - does not block
  - pre-defined funcs: PvmMin, PvmMax,PvmSum,PvmProduct

PVM Performance Issues

- Messages have to go through PVMD
  - can use direct route option to prevent this problem
- Packing messages
  - semantics imply a copy
  - extra function call to pack messages
- Heterogenous Support
  - information is sent in machine independent format
  - has a short circuit option for known homogenous comm.
    - passes data in native format then

Sample PVM Program

```
int main(int argc, char **argv) {
    int myGroupNum;
    int friendTid;
    int mytid;
    int tids[2];
    int message[MESSAGESIZE];
    int c,i,okSpawn;

    /* Initialize process and spawn if necessary */
    myGroupNum=pvm_joingroup("ping-pong");
    mytid=pvm_mytid();
    if (myGroupNum==0)  { /* I am the first process */
        pvm_catchout(stdout);
        okSpawn=pvm_spawn(MYNAME,argv,0,"",1,&friendTid);
        if (okSpawn!=1) {
            printf("Can't spawn a copy of myself!\n");
            pvm_exit();
            exit(1);
        }
        tids[0]=mytid;
        tids[1]=friendTid;
    } else { /*I am the second process */
        friendTid=pvm_parent();
        tids[0]=friendTid;
        tids[1]=mytid;
    }
    pvm_barrier("ping-pong",2);

    /* Main Loop Body */
    if (myGroupNum==0) {
        /* Initialize the message */
        for (i=0 ; i<MESSAGESIZE ; i++) {
            message[i]='1';
        }

        /* Now start passing the message back and forth */
        for (i=0 ; i<ITERATIONS ; i++) {
            pvm_initsend(PvmDataDefault);
            pvm_pkint(message,MESSAGESIZE,1);
            pvm_send(tid,msgid);

            pvm_recv(tid,msgid);
            pvm_upkint(message,MESSAGESIZE,1);
          }
    } else {
            pvm_recv(tid,msgid);
            pvm_upkint(message,MESSAGESIZE,1);
            pvm_initsend(PvmDataDefault);
            pvm_pkint(message,MESSAGESIZE,1);
            pvm_send(tid,msgid);
    }
    pvm_exit();
    exit(0);
}
```

MPI

- Goals:
  - Standardize previous message passing:
    - PVM, P4, NX
  - Support copy free message passing
  - Portable to many platforms
- Features:
  - point-to-point messaging
  - group communications
  - profiling interface: every function has a name shifted version
- Buffering
  - no guarantee that there are buffers
  - possible that send will block until receive is called
- Delivery Order
  - two sends from same process to same dest. will arrive in order
  - no guarantee of fairness between processes on recv.

MPI Communicators

- Provide a named set of processes for communication
- All processes within a communicator can be named
  - numbered from 0…n-1
- Allows libraries to be constructed
  - application creates communicators
  - library uses it
  - prevents problems with posting wildcard receives
    - adds a communicator scope to each receive
- All programs start will MPI_COMM_WORLD

Non-Blocking Functions

- Two Parts
  - post the operation
  - wait for results
- Also includes a poll option
  - checks if the operation has finished
- Semantics
  - must not alter buffer while operation is pending

MPI Misc.

- MPI Types
  - All messages are typed
    - base types are pre-defined:
      - int, double, real, {,unsigned}{short, char, long}
    - can construct user defined types
      - includes non-contiguous data types
- Processor Topologies
  - Allows construction of Cartesian & arbitrary graphs
  - May allow some systems to run faster
- What’s not in MPI-1
  - process creation
  - I/O
  - one sided communication