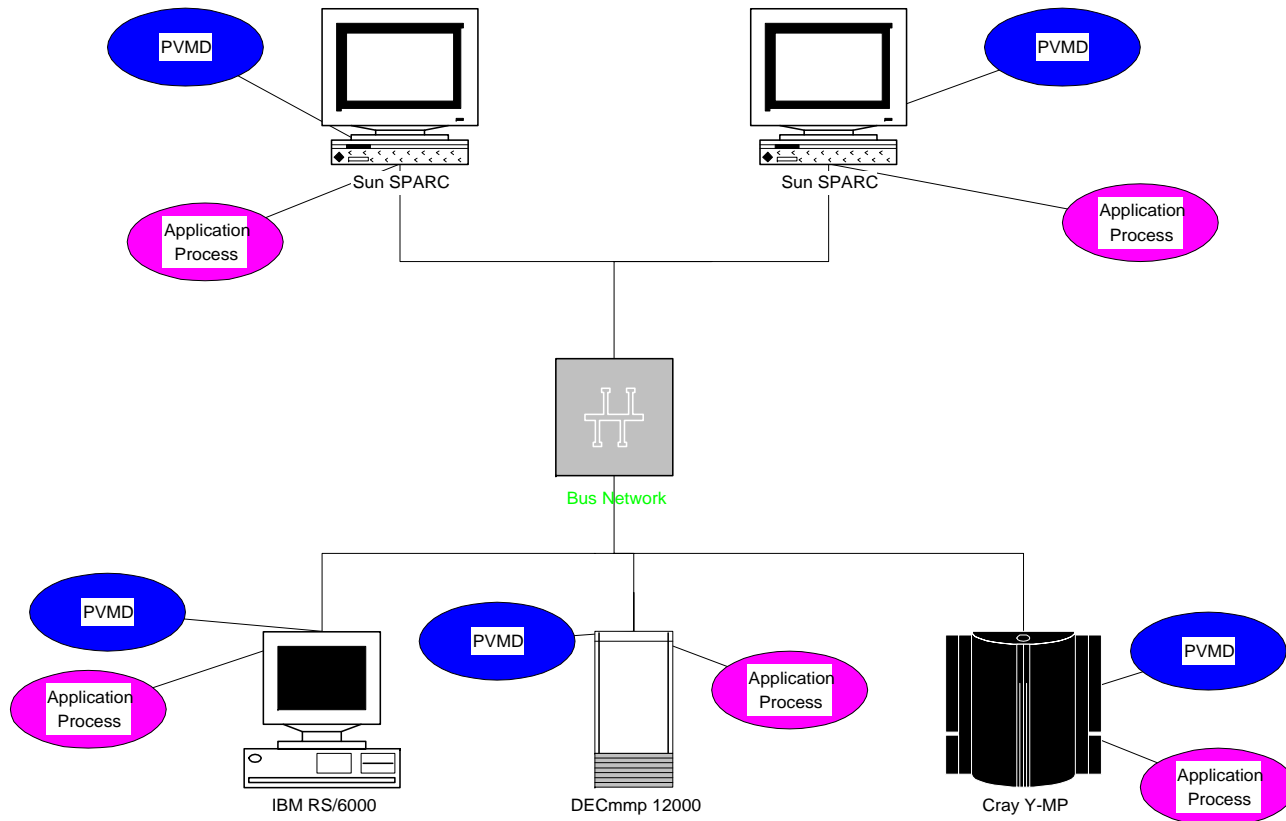


Message Passing with PVM and MPI

PVM

- Provide a simple, free, portable parallel environment
- Run on everything
 - Parallel Hardware: SMP, MPPs, Vector Machines
 - Network of Workstations: ATM, Ethernet,
 - UNIX machines and PCs running Win32 API
 - Works on a heterogenous collection of machines
 - handles type conversion as needed
- Provides two things
 - message passing library
 - point-to-point messages
 - synchronization: barriers, reductions
 - OS support
 - process creation (`pvm_spawn`)

PVM Environment (UNIX)



- One PVMD per machine
 - all processes communicate through pvmd (by default)
- Any number of application processes per node

CMSC 818S – Alan Sussman (from J. Hollingsworth)

PVM Message Passing

- All messages have tags
 - an integer to identify the message
 - defined by the user
- Messages are constructed, then sent
 - `pvm_pk{int,char,float}(*var, count, stride)`
 - `pvm_unpk{int,char,float}` to unpack
- All processes are named based on task ids (tids)
 - local/remote processes are the same
- Primary message passing functions
 - `pvm_send(tid, tag)`
 - `pvm_recv(tid, tag)`

PVM Process Control

- **Creating a process**
 - `pvm_spawn(task, argv, flag, where, ntask, tids)`
 - `flag` and `where` provide control of where tasks are started
 - `ntask` controls how many copies are started
 - program must be installed on target machine
- **Ending a task**
 - `pvm_exit`
 - does not exit the process, just the PVM machine
- **Info functions**
 - `pvm_mytid()` - get the process task id

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_joiningroup("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);

    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++) {
            if (myGroupNum==0) {
                pvm_initsend(PvmDataDefault);
                pvm_pkint(message,MESSAGESIZE,1);
                pvm_send(friendTid,msgid);
                pvm_rcv(friendTid,msgid);
                pvm_upkint(message,MESSAGESIZE,1);
            }
            else {
                pvm_rcv(friendTid,msgid);
                pvm_upkint(message,MESSAGESIZE,1);
                pvm_initsend(PvmDataDefault);
                pvm_pkint(message,MESSAGESIZE,1);
                pvm_send(friendTid,msgid);
            }
        }
        pvm_exit();
        exit(0);
    }
}
```

MPI

- **Goals:**

- Standardize previous message passing:
 - PVM, P4, NX, MPL, ...
- Support copy-free message passing
- Portable to many platforms

- **Features:**

- point-to-point messaging
- group/collective communications
- profiling interface: every function has a name shifted version

- **Buffering (in standard mode)**

- 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
 - plus a context – system allocated unique tag
- 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 with `MPI_COMM_WORLD`
 - Functions for creating communicators from other communicators (split, duplicate, etc.)
 - Functions for finding out about processes within communicator (size, my_rank, ...)

Non-Blocking Point-to-point Functions

- **Two Parts**
 - post the operation
 - wait for results
- **Also includes a poll/test option**
 - checks if the operation has finished
- **Semantics**
 - must not alter buffer while operation is pending (wait returns or test returns true)
 - and data not valid for a receive until operation completes

Collective Communication

- Communicator specifies process group to participate
- Various operations, that may be optimized in an MPI implementation
 - Barrier synchronization
 - Broadcast
 - Gather/scatter (with one destination, or all in group)
 - Reduction operations – predefined and user-defined
 - Also with one destination or all in group
 - Scan – prefix reductions
- Collective operations may or may not synchronize
 - Up to the implementation, so application can't make assumptions

MPI Misc.

- MPI Types
 - All messages are typed
 - base/primitive 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
- Language bindings for C, Fortran, C++, ...
- What's not in MPI-1
 - process creation
 - I/O
 - one sided communication

Sample MPI Program

```
#include "mpi.h"
int main(int argc, char **argv) {
    int myrank, friendRank;
    int message[MESSAGESIZE];
    int i, tag=MSG_TAG;
    MPI_Status status;

    /* Initialize, no spawning necessary */
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
    if (myrank==0) { /* I am the first process */
        friendRank = 1;
    }
    else { /*I am the second process */
        friendRank=0;
    }
    MPI_Barrier(MPI_COMM_WORLD);
    if (myrank==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++) {
        if (myrank==0) {
            MPI_send(message, MESSAGESIZE,
                MPI_CHAR, friendRank, tag,
                MPI_COMM_WORLD);

            MPI_Recv(message, MESSAGESIZE,
                MPI_CHAR, friendRank, tag,
                MPI_COMM_WORLD, &status);
        }
        else {
            MPI_Recv(message, MESSAGESIZE,
                MPI_CHAR, friendRank, tag,
                MPI_COMM_WORLD, &status);

            MPI_send(message, MESSAGESIZE,
                MPI_CHAR, friendRank, tag,
                MPI_COMM_WORLD);
        }
    }
    MPI_Finalize();
    exit(0);
}
```

For more details

- **PVM** – http://www.csm.ornl.gov/pvm/pvm_home.html
 - current version is 3.4.3, available for download from netlib
 - book from MIT Press is *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing*
- **MPI** – <http://www.mpi-forum.org>
 - includes both 1.1 and 2.0 documentation (API)
 - books from MIT Press include *Using MPI* and *MPI: The Complete Reference*
 - multiple public domain implementations available
 - mpich – Argonne National Lab
 - LAM – Ohio Supercomputing Center
 - vendor implementations available too (IBM, Compaq/HP, ...)
 - for UMIACS Linux cluster info, see <http://www.umiacs.umd.edu/research/parallel/> - start with **Class guide** link – you'll be using the red/blue cluster