

## Message Passing with PVM and MPI

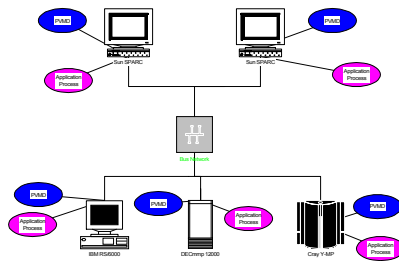
## PVM

- 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)

CMSC 818 – Alan Sussman (from J. Hollingsworth)

2

## PVM Environment (UNIX)



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

CMSC 818 – Alan Sussman (from J. Hollingsworth)

3

## 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)`

CMSC 818 – Alan Sussman (from J. Hollingsworth)

4

## 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

CMSC 818 – Alan Sussman (from J. Hollingsworth)

5

## 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`

CMSC 818 – Alan Sussman (from J. Hollingsworth)

6

## 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

CMSC 818 – Alan Sussman (from J. Hollingsworth)

7

## Sample PVM Program

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

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

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

            pvm_recv(tid,msgid);
            pvm_upkin(message,MESSAGE_SIZE,1);
        }
    } else {
        pvm_recv(tid,msgid);
        pvm_upkin(message,MESSAGE_SIZE,1);
        pvm_initSend(PvmDataDefault);
        pvm_pkin(message,MESSAGE_SIZE,1);
        pvm_send(tid,msgid);
    }
}

/* Initialize process and spawn if necessary */
myGroupNum=pvm_joinGroup("ping-pong");
mytid=pvm_mytid();
if (myGroupNum==0) { /* I am the first process */
    pvm_catOut(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);
pvm_exit();
exit(0);
}
```

CMSC 818 – Alan Sussman (from J. Hollingsworth)

8

## 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.

CMSC 818 – Alan Sussman (from J. Hollingsworth)

9

## 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, ...)

CMSC 818 – Alan Sussman (from J. Hollingsworth)

10

## 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)

CMSC 818 – Alan Sussman (from J. Hollingsworth)

11

## 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

CMSC 818 – Alan Sussman (from J. Hollingsworth)

12

## 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

CMSC 818 – Alan Sussman (from J. Hollingsworth)

13

## For more details

- **PVM** – [http://www.csm.ornl.gov/pvm/pvm\\_home.html](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, ...)

CMSC 818 – Alan Sussman (from J. Hollingsworth)

14