6 The Beowulf Concept

- Due to Don Becker and Tom Sterling
- Don’t try to attack the parallel computer engineering problem
- Build a machine from COTS parts
- Not really the same as SNoW

6.1 daniel at USC

- One head node
- 32 processing nodes
- Each is a 900 MHz Pentium III with 1 Gbyte memory
- Ethernet connecting nodes

6.2 Beowulf requirements

- MUST be able to distribute the processing
- If research dollars, concentrate only on the interconnect
- If we can overcome the algorithmic issues of distributed computation, then we win regardless of hardware technology
7 Message Passing, Specifically MPI

- SPMD mode
- Every processor gets a rank number
- Code is executed or not based on rank number checking in the program
- Everything stems from the basic send and receive commands
- Synchronous message passing
- Blocking and nonblocking message passing
- Message buffers
- Message sizes and communication cost
- Message tags

7.1 Message Passing Primitives

- Broadcast
- Gather
- Scatter
- Reduction
- NB also total exchange and the history of “gather,” “scatter,” and “reduction”

7.2 Message Passing Systems

- MPI
- PVM
7.3 The First Sample Program

• Basic MPI needs

#include "mpi.h"
...
/* BEFORE ANY OTHER MPI CALL */
int MPI_Init(
    int* argc,    /* the usual c language args to main */
    char** argv[]);
/* PROBABLY THEN IMMEDIATELY */
int MPI_Comm_size(
    MPI_Comm comm,    /* defines the universe of processes */
    int* number_of_processes);    /* and the number of processes */
int MPI_Comm_rank(
    MPI_Comm comm,    /* in the universe of processes */
    int* my_rank);    /* what’s my rank number? */
...

more code
...

/* AFTER ALL OTHER MPI CALLS */
int MPI_Finalize(void);

• Sample program timetest.c
• Send and receive calls

```c
int MPI_Send(
    void* message,   /* data buffer for data to be sent */
    int count,       /* number of items to send */
    MPI_Datatype datatype, /* type of each item */
    int dest,        /* destination rank of process */
    int tag,         /* tag for synchronisation */
    MPI_Comm comm); /* communicator */
```

```c
int MPI_Recv(
    void* message,   /* data buffer for data to be rec’d */
    int count,       /* number of items to receive */
    MPI_Datatype datatype, /* type of each item */
    int source,      /* source process rank of data */
    int tag,         /* tag for synchronisation */
    MPI_Comm comm,   /* communicator */
    MPI_Status* status); /* status structure for debug/check */
```

Datatypes are

- `MPI_CHAR`
- `MPI_SHORT`
- `MPI_INT`
- `MPI_LONG`
- `MPI_UNSIGNED_CHAR`
- `MPI_UNSIGNED_SHORT`
- `MPI_UNSIGNED_INT`
- `MPI_UNSIGNED_LONG`
- `MPI_FLOAT`
- `MPI_DOUBLE`
- `MPI_LONG_DOUBLE`
- `MPI_BYTE`
- `MPI_PACKED`
• Broadcast, reduction

```
int MPI_Bcast(
    void* message,    /* data buffer for data to be sent */
    int count,       /* number of items to send */
    MPI_Datatype datatype, /* type of each item */
    int root,        /* root from whom to broadcast */
    MPI_Comm comm);  /* communicator */
```

```
int MPI_Reduce(
    void* operand,    /* data buffer from which to reduce */
    void* result,     /* data buffer into which to reduce */
    int count,        /* number of items to receive */
    MPI_Datatype datatype, /* type of the items */
    MPI_Op operator,  /* reduction operation */
    int root,         /* rank onto whom to reduce */
    MPI_Comm comm);   /* communicator */
```

Operands are
- `MPI_MAX` /* maximum */
- `MPI_MIN` /* minimum */
- `MPI_SUM` /* sum */
- `MPI_PROD` /* product */
- `MPI_LAND` /* logical and */
- `MPI_BAND` /* binary and */
- `MPI_LOR` /* logical or */
- `MPI_BOR` /* binary or */
- `MPI_LXOR` /* logical xor */
- `MPI_BXOR` /* binary xor */
- `MPI_MAXLOC` /* max and location of max */
- `MPI_MINLOC` /* min and location of min */
• Reduction onto all processes

```c
int MPI_Allreduce(
    void* operand, /* data buffer from which to reduce */
    void* result, /* data buffer into which to reduce */
    int count, /* number of items to receive */
    MPI_Datatype datatype, /* type of the items */
    MPI_Op operator, /* reduction operation */
    MPI_Comm comm); /* communicator */
```

(Allreduce uses an internally-defined "best communication path")

page 77 butterfly

### 7.4 Some Issues

• I/O

• Buffering

• Synchronisation

> The number of parallel programs I have embarked on is less than the number of deadlocks I have created.

>(Ken Kennedy)