ACM/CS 114
Parallel algorithms for scientific applications

Michael A. G. Aïvázis

California Institute of Technology

Winter 2012

Messages

- in general, data exchanges through MPI calls involve
  - a communicator
    - specifies which processes participate in the exchange
    - resolves process ranks into processes
  - *collective* operations involve the entire communicator
  - *point-to-point* operations require the rank of the message source or destination
  - the details of the message payload
    - the address of the source buffer
    - the data type of the buffer contents
    - the number of items in the buffer
- `MPI` provides some data abstractions to
  - hide machine dependencies in the data representations to enhance portability and support heterogeneous clusters
  - support user defined data types
  - support non-contiguous data layouts

Collective operations: global reductions

- *collective* operations involve all processes in a given communicator
- the `MPI` version of our global reduction example uses

```
int MPI_Allreduce(
      void* send_buffer, void* recv_buffer,
      int count, MPI_Datatype datatype, MPI_Op operation,
      MPI_Comm communicator
      );
```

- example legal values for `MPI_Datatype`
  - C: `MPI_INT`, `MPI_LONG`, `MPI_DOUBLE`
  - FORTRAN: `MPI_INTEGER`, `MPI_DOUBLE_PRECISION`, `MPI_COMPLEX`
- legal values for `MPI_Op`
  - `MPI_MAX`, `MPI_MIN`, `MPI_MAXLOC`, `MPI_MINLOC`
  - `MPI_SUM`, `MPI_PROD`
  - `MPI_LAND`, `MPI_LOR`, `MPI_LXOR`
  - `MPI_BAND`, `MPI_BOR`, `MPI_BXOR`
  - `MPI_REPLACE`

Example reduction using MPI

```
1  #include <mpi.h>
2  #include <stdio.h>
3
4  int main(int argc, char* argv[]) {
5     int status;
6     int rank;
7     int square, sum;
8
9     /* initialize MPI */
10    status = MPI_Init(&argc, &argv);
11    if (status != MPI_SUCCESS) {
12       printf("error in MPI_Init; aborting...\n");
13       return status;
14    }
15
16    /* get the process rank */
17    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
18    /* form the square */
19    square = rank*rank;
20    /* each process contributes the square of its rank */
21    MPI_Allreduce(&square, &sum, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
22    /* print out the result */
23    printf("%03d: sum = %d\n", rank, sum);
24
25    /* shut down MPI */
26    MPI_Finalize();
27
28    return 0;
29 }
```