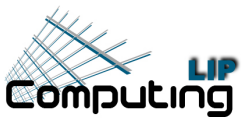
MPI and Parallel I/O

Miguel Afonso Oliveira

Laboratório de Instrumentação e Física Experimental de Partículas

LIP

LNEC April 2010

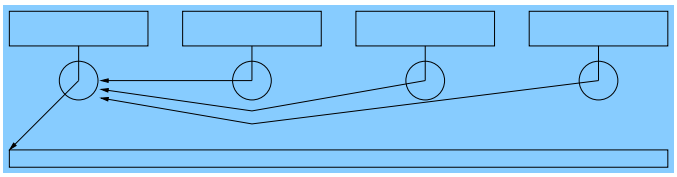


Introduction

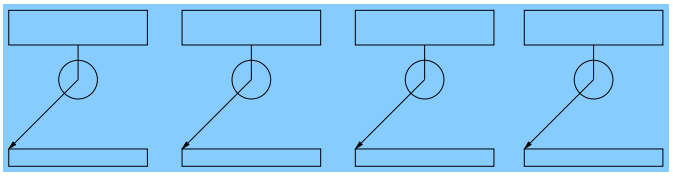
- Programming Languages have predefined functions to handle files.
- Typical operations are open, close, read and write.
- MPI supports counterparts.
- These MPI routines can conveniently express parallelism.

I/O types supported within MPI programs

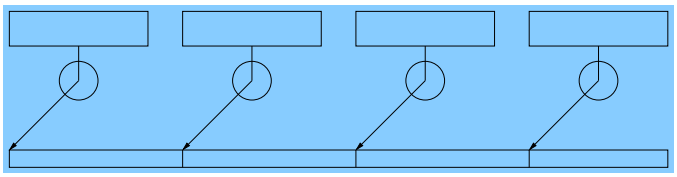
- Non-parallel I/O



- I/O to separate files



- Parallel I/O



Non-parallel I/O from an MPI program

```
#include "mpi.h"
#include <stdio.h>
#define BUFSIZE 100

int main(int argc, char *argv[])
{
    int i, myrank, numprocs, buf[BUFSIZE];
    MPI_Status status;
    FILE *myfile;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    for (i=0; i<BUFSIZE; i++)
         buf[i] = myrank * BUFSIZE + i;
    if (myrank != 0)
         MPI_Send(buf, BUFSIZE, MPI_INT, 0, 99, MPI_COMM_WORLD);
    else {
         myfile = fopen("testfile", "w");
         fwrite(buf, sizeof(int), BUFSIZE, myfile);
         for (i=1; i<numprocs; i++) {
              MPI_Recv(buf, BUFSIZE, MPI_INT, i, 99, MPI_COMM_WORLD, &status);
              fwrite(buf, sizeof(int), BUFSIZE, myfile);
        }
        fclose(myfile);
        }
    MPI_Finalize();
    return 0;
}
```

Non-MPI I/O to separate files


```
#include "mpi.h"
#include <stdio.h>
#define BUFSIZE 100

int main(int argc, char *argv[])
{
    int i, myrank, buf[BUFSIZE];
    char filename[128];
    FILE *myfile;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    for (i=0; i<BUFSIZE; i++)
          buf[i] = myrank * BUFSIZE + i;
    sprintf(filename, "testfile.%d", myrank);
    myfile = fopen(filename, "w");
    fwrite(buf, sizeof(int), BUFSIZE, myfile);
    fclose(myfile);
    MPI_Finalize();
    return 0;
}
```

MPI I/O to separate files


```
#include "mpi.h"
#include <stdio.h>
#define BUFSIZE 100

int main(int argc, char *argv[])
{
    int i, myrank, buf[BUFSIZE];
    char filename[128];
    MPI_File myfile;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    for (i=0; i<BUFSIZE; i++)
        buf[i] = myrank * BUFSIZE + i;
    sprintf(filename, "testfile.%d", myrank);
    MPI_File_open(MPI_COMM_SELF, filename, MPI_MODE_WRONLY | MPI_MODE_CREATE,
                                  MPI_INFO_NULL, &myfile);
    MPI_File_write(myfile, buf, BUFSIZE, MPI_INT, MPI_STATUS_IGNORE);
    MPI_File_close(&myfile);
    MPI_Finalize();
    return 0;
}
```

Parallel MPI I/O to a single file


```
#include "mpi.h"
#include <stdio.h>
#define BUFSIZE 100

int main(int argc, char *argv[])
{
    int i, myrank, buf[BUFSIZE];
    MPI_File thefile;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    for (i=0; i<BUFSIZE; i++)
        buf[i] = myrank * BUFSIZE + i;
    MPI_File_open(MPI_COMM_WORLD, "testfile",
                                  MPI_MODE_CREATE | MPI_MODE_WRONLY, MPI_INFO_NULL, &thefile);
    MPI_File_set_view(thefile, myrank * BUFSIZE * sizeof(int),
                                       MPI_INT, MPI_INT, "native", MPI_INFO_NULL);
    MPI_File_write(thefile, buf, BUFSIZE, MPI_INT, MPI_STATUS_IGNORE);
    MPI_File_close(&thefile);
    MPI_Finalize();
    return 0;
}
```

Reading a file


```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[])
{
    int myrank, numprocs, bufsize, *buf, count;
    MPI_File thefile;
    MPI_Status status;
    MPI_Offset filesize;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_File_open(MPI_COMM_WORLD, "testfile", MPI_MODE_RDONLY, MPI_INFO_NULL, &thefile);
    MPI_File_get_size(thefile, &filesize); /* in bytes */
    filesize = filesize / sizeof(int);     /* in number of ints */
    bufsize = filesize / numprocs + 1;     /* local number to read */
    buf = (int *) malloc (bufsize * sizeof(int));
    MPI_File_set_view(thefile, myrank * bufsize * sizeof(int),
                                       MPI_INT, MPI_INT, "native", MPI_INFO_NULL);
    MPI_File_read(thefile, buf, bufsize, MPI_INT, &status);
    MPI_Get_count(&status, MPI_INT, &count);
    printf("process %d read %d ints\n", myrank, count);
    MPI_File_close(&thefile);
    MPI_Finalize();
    return 0;
}
```

Fileviews?

- Fileviews
  - MPI_File_set_view(...)
  - MPI_File_read(...)
  - MPI_File_write(...)
- Individual File Pointers
  - MPI_File_seek(...)
  - MPI_File_read(...)
  - MPI_File_write(...)
- Explicit offsets
  - MPI_File_read_at(...)
  - MPI_File_write_at(...)