SIO "High-level" API

Jeff Hollingsworth

Anurag Acharya

Joel Saltz

Alan Sussman

Why adopt MIO-IO as the SIO H.L. API?

- Meets the Goals established last time
  - codifies existing practice
    - file pointers and direct I/O
    - blocking and non-blocking I/O
    - collective I/O
  - provides a single target for vendors
    - can use or ignore the low level API as desired
  - is acceptable for “higher” level libraries
- Well under way
  - draft standard exists
  - references implementations are started

Issues for non-MPI Applications?

- Communicators
  - Just use `MPI_comm_self` or `MPI_comm_world`
  - create new communicators for collective I/O
- Types
  - ignore type system
    - files are a collection of `MPI_byte`
  - use type constructors as needed
    - provides additional info to I/O system
- Questions:
  - Should there be some convenience functions to help non-MPI programmers use MPI-IO?
  - Are there “critical” features missing?

Using MPI-IO

- A non-MPI example
- Pathfinder (earth imaging application)
  - can use byte as etype or construct record

```
MPI_Type_contiguous(4,MPI_UNSIGNED_SHORT,&geolocation_pt);
MPI_Type_contiguous(51,geolocation_pt,&scanline_types[0]);
MPI_Type_contiguous(5,MPI_UNSIGNED_SHORT,&pixel);
MPI_Type_contiguous(409,pixel,&scanline_types[1]);

scanline_disp[0] = 0;
MPI_Type_extent(scanline_types[0], &scanline_disp[1]);

scanline_num[0] = 1;
scanline_num[1] = 1;

MPI_Type_struct(2,scanline_num,scanline_disp,scanline_types,&scanline);

MPI_Type_contiguous(128,scanline,&chunk);
```

Pathfinder (cont)

```
for (i = 0, j = 0; i < N && j < count; i++)
{
 if (chunk_assign[i] >= 0)
  {
 partition_num[j] = 1;
    partition_dis[j] = chunk_extent * chunk_assign[i];
    /* don't know if you can copy datatypes */
    MPI_Type_contiguous(1,chunk,&partition_types[j]);
    j++;
  }
}

MPI_Type_struct(count,partition_num,partition_disp,partition_types,
                      &partition);

/* now partition contains the file type to be used by this processor.
 * every processor will create its own file type */

MPIO_Open(comm, "input", mode, header_disp, scanline, partition, hints, *fh);

/* input reads are done as follows */
MPIO_Read(fh,offset,scanline_buf,1,status);
```

Useful modes

- MPI-IO supports many modes
- Expect several modes to be most common
  - non-collective direct I/O
    - maybe collective if it would improve speed
  - independent pointer
    - used for a processor specific “view” into a file
  - shared file pointer
    - great for logfiles
    - it will be slow, but this is mostly for debugging
- Need to ensure popular modes are fast

Where to go from here?

- Break out sessions
  - focus on “open” issues
    - MPI-IO for non MPI programs
    - peer to peer configurations
- Near term
  - Maryland: will convert our I/O kernels to MPI-IO
  - Encourage other kernels and applications in the repository to be converted to MPI-IO
  - study performance implications of using MPI-IO