Lecture 11 Summary

This lecture was about MPI parallel file I/O. There are two methods to do the I/O: Sequential I/O and Parallel I/O. For Sequential I/O, it can be implemented with any system since some parallel machine may support I/O from only one process or some I/O libraries like PMPIO are not parallel. Also, its output single file is more convenient for some transfer application like ftp or mv. Big file blocks also improve performance. However, because of lack of parallelism, scalability and overall performance are limited. Although the scalability can be improved by letting each process to write to a separate file. But this also create los of small files to manage, which could stress parallel filesystem and make it difficult to read back data from different number of process. Therefore, Parallel I/O comes out, which lets multiple processes of a parallel program accessing data form a common file. The provided single file can be used with other tools like visualization programs. Like before, a parallel I/O system will need a mechanism to define collective operations, noncontiguous data layout in memory and file and test completion of nonblocking operations, which would lead to lots of MPI-like machinery. For each process, they can use whether individual file pointers or explicit offsets. The core MPI I/O functions are MPI\_File\_open (), MPI\_File\_close (), MPI\_File\_read (), MPI\_File\_read\_at (), MPI\_File\_seek (), MPI\_File\_write (), and MPI\_File\_write\_at (). For MPI\_File\_write and MPI\_File\_write\_at, if the file exists, use MPI\_MODE\_WRONLY or MPI\_MODE\_RDWR as the flags to MPI\_File\_open. Otherwise, MPI\_MODE\_CREATE should also be passed to MPI\_File\_open using bitwise-or. Besides some basic datatype pre-defined by MPI, users can also define their custom datatype by calling MPI\_Type\_contiguous () and MPI\_Type\_commit (). For each process, their regions of the file can be defined by calling MPI\_FILE\_set\_view (). There are three ways to write to a shared file. The first one is like UNIX seek, the second one combines seek and I/O for thread safety, the third one uses shared file pointer when order does not matter, and the last one is using collective operations. Collective I/O is a critical optimization in parallel I/O. The basic idea is to build large blocks to make reads/writes in I/O system being large.

The last is the general guidelines for achieving high I/O performance: 1. Buy sufficient I/O hardware. 2. Use fast file system. 3. Do not perform I/O from only one process. 4. Make large requests wherever possible. 5. For noncontiguous requests, use derived datatypes and a single collective I/O call.