Chapter 1

I/O

1.1 Introduction

POSIX provides a model of a widely portable file system, but the portability and optimization needed for parallel I/O cannot be achieved with the POSIX interface.

The significant optimizations required for efficiency (e.g., grouping [7], collective buffering [1, 2, 8, 9, 10], and disk-directed I/O [6]) can only be implemented if the parallel I/O system provides a high-level interface supporting partitioning of file data among processes and a collective interface supporting complete transfers of global data structures between process memories and files. In addition, further efficiencies can be gained via support for asynchronous I/O, strided accesses, and control over physical file layout on storage devices (disks). The I/O environment described in this chapter provides these facilities.

Instead of defining I/O access modes to express the common patterns for accessing a shared file (broadcast, reduction, scatter, gather), we chose another approach in which data partitioning is expressed using derived datatypes. Compared to a limited set of predefined access patterns, this approach has the advantage of added flexibility and expressiveness.

1.1.1 Definitions

file An MPI file is an ordered collection of typed data items. MPI supports random or sequential access to any integral set of these items. A file is opened collectively by a group of processes. All collective I/O calls on a file are collective over this group.

displacement A file *displacement* is an absolute byte position relative to the beginning of a file. The displacement defines the location where a *view* begins. Note that a "file displacement" is distinct from a "typemap displacement."

etype An *etype* (*elementary* datatype) is the unit of data access and positioning. It can be any MPI predefined or derived datatype. Derived etypes can be constructed using any of the MPI datatype constructor routines, provided all resulting typemap displacements are non-negative and monotonically nondecreasing. Data access is performed in etype units, reading or writing whole data items of type etype. Offsets are expressed as a count of etypes; file pointers point to the beginning of etypes. Depending on context, the term "etype" is used to describe one of three aspects of an elementary datatype: a particular MPI type, a data item of that type, or the extent of that type.

filetype A *filetype* is the basis for partitioning a file among processes and defines a template for accessing the file. A filetype is either a single etype or a derived MPI datatype constructed from multiple instances of the same etype. In addition, the extent of any hole in the filetype must be a multiple of the etype's extent. The displacements in the typemap of the filetype are not required to be distinct, but they must be non-negative and monotonically nondecreasing.

view A *view* defines the current set of data visible and accessible from an open file as an ordered set of etypes. Each process has its own view of the file, defined by three quantities: a displacement, an etype, and a filetype. The pattern described by a filetype is repeated, beginning at the displacement, to define the view. The pattern of repetition is defined to be the same pattern that MPI_TYPE_CONTIGUOUS would produce if it were passed the filetype and an arbitrarily large count. Figure 1.1 shows how the tiling works; note that the filetype in this example must have explicit lower and upper bounds set in order for the initial and final holes to be repeated in the view. Views can be changed by the user during program execution. The default view is a linear byte stream (displacement is zero, etype and filetype equal to MPI_BYTE).

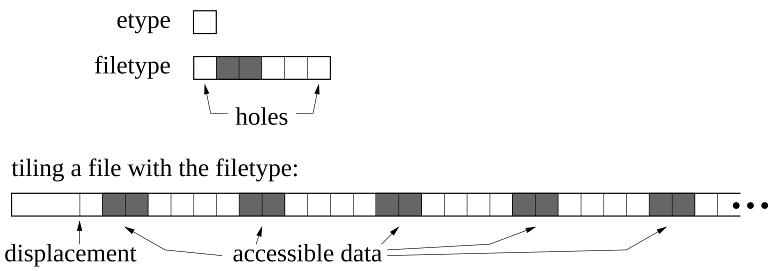

Figure 1.1: Etypes and filetypes

A group of processes can use complementary views to achieve a global data distribution such as a scatter/gather pattern (see Figure 1.2).

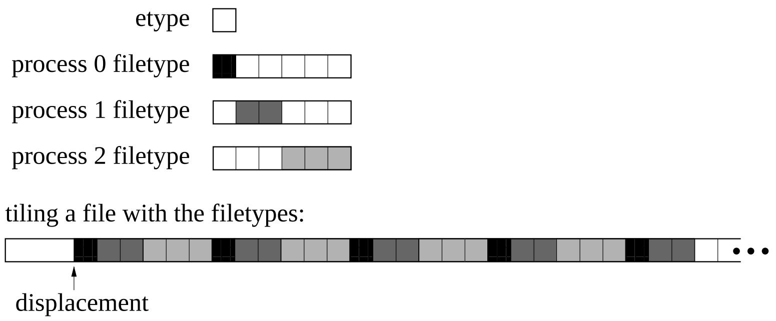

Figure 1.2: Partitioning a file among parallel processes

offset An *offset* is a position in the file relative to the current view, expressed as a count of etypes. Holes in the view's filetype are skipped when calculating this position. Offset 0 is the location of the first etype visible in the view (after skipping the displacement and any initial holes in the view). For example, an offset of 2 for process 1 in Figure 1.2 is the position of the 8th etype in the file after the displacement. An "explicit offset" is an offset that is used as a formal parameter in explicit data access routines.

Advice to implementors. It is expected that a call to MPI_FILE_SET_VIEW will immediately follow MPI_FILE_OPEN in numerous instances. A high-quality implementation will ensure that this behavior is efficient. (*End of advice to implementors.*)

The disp displacement argument specifies the position (absolute offset in bytes from the beginning of the file) where the view begins.

Advice to users. disp can be used to skip headers or when the file includes a sequence of data segments that are to be accessed in different patterns (see Figure 1.3). Separate views, each using a different displacement and filetype, can be used to access each segment.

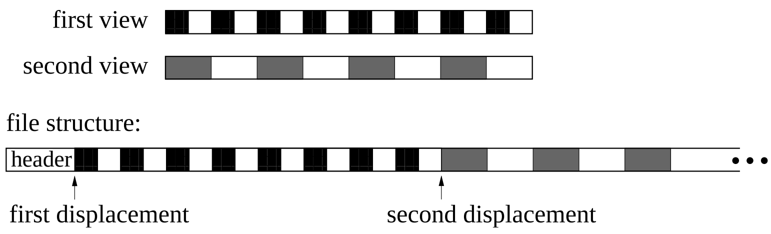

Figure 1.3: Displacements

(*End of advice to users.*)

An *etype* (*elementary* datatype) is the unit of data access and positioning. It can be any MPI predefined or derived datatype. Derived etypes can be constructed by using any of the MPI datatype constructor routines, provided all resulting typemap displacements are non-negative and monotonically nondecreasing. Data access is performed in etype units, reading or writing whole data items of type etype. Offsets are expressed as a count of etypes; file pointers point to the beginning of etypes.

Advice to users. In order to ensure interoperability in a heterogeneous environment, additional restrictions must be observed when constructing the etype (see Section 1.5, page 38). (*End of advice to users.*)

A filetype is either a single etype or a derived MPI datatype constructed from multiple instances of the same etype. In addition, the extent of any hole in the filetype must be a multiple of the etype's extent. These displacements are not required to be distinct, but they cannot be negative, and they must be monotonically nondecreasing.

If the file is opened for writing, neither the etype nor the filetype is permitted to contain overlapping regions. This restriction is equivalent to the "datatype used in a receive cannot specify overlapping regions" restriction for communication. Note that filetypes from different processes may still overlap each other.

If filetype has holes in it, then the data in the holes is inaccessible to the calling process. However, the disp, etype and filetype arguments can be changed via future calls to MPI_FILE_SET_VIEW to access a different part of the file.

It is erroneous to use absolute addresses in the construction of the etype and filetype.

The info argument is used to provide information regarding file access patterns and file system specifics to direct optimization (see Section 1.2.8, page 10). The constant MPI_INFO_NULL refers to the null info and can be used when no info needs to be specified.