Parallel I/O Optimizations

Sources/Credits:

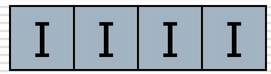
- R. Thakur, W. Gropp, E. Lusk. A Case for Using MPI's Derived Datatypes to Improve I/O Performance. Supercomputing 98
- URL: http://www.mcs.anl.gov/~thakur/dtype

High Performance I/O with Derived Data Types

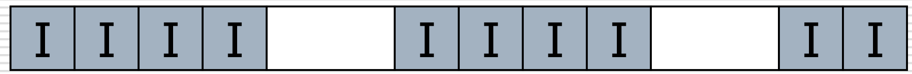
- Potential of parallel file systems not fully utilized because of application’s I/O access patterns
  - Parallel file systems: Tuned for access to large contiguous blocks
  - User applications: Many small requests to non-contiguous blocks
- Can be improved using a single call made using derived data types
- Using templates with holes can improve performance

Datatype Constructors in MPI

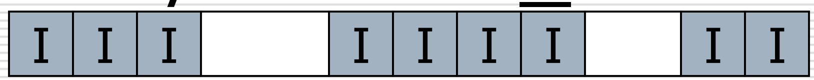
1. contiguous



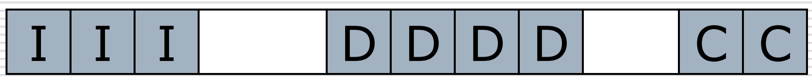
2. vector/hvector



3. indexed/hindexed/indexed_block

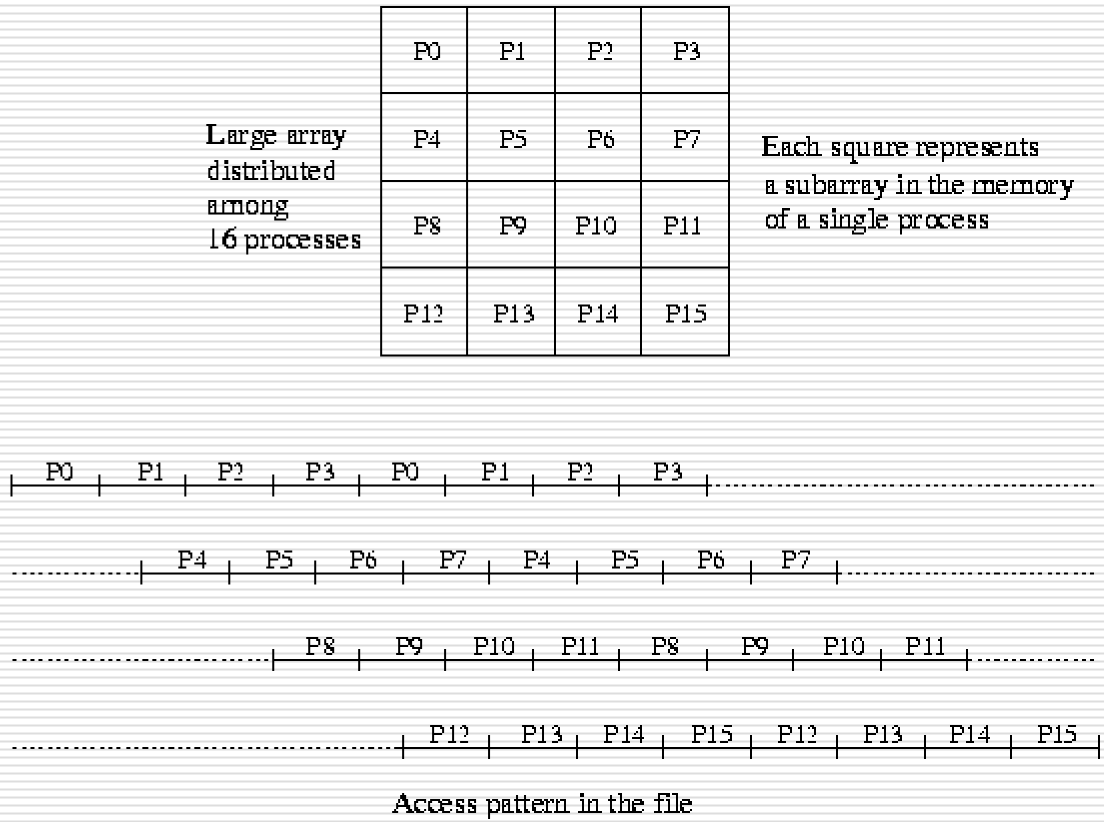


4. struct



5. subarray
6. darray

Different levels of access



Different levels of access

```
MPI_File_open(..., "filename", ..., &fh)
for (i=0; i<n_local_rows; i++) {
   MPI_File_seek(fh, ...)
   MPI_File_read(fh, row[i], ...)
}
MPI_File_close(&fh)
```

Level 0
(many independent, contiguous requests)

```
MPI_File_open(MPI_COMM_WORLD, "filename", ..., &fh)
for (i=0; i<n_local_rows; i++) {
   MPI_File_seek(fh, ...)
   MPI_File_read_all(fh, row[i], ...)
}
MPI_File_close(&fh)
```

Level 1
(many collective, contiguous requests)

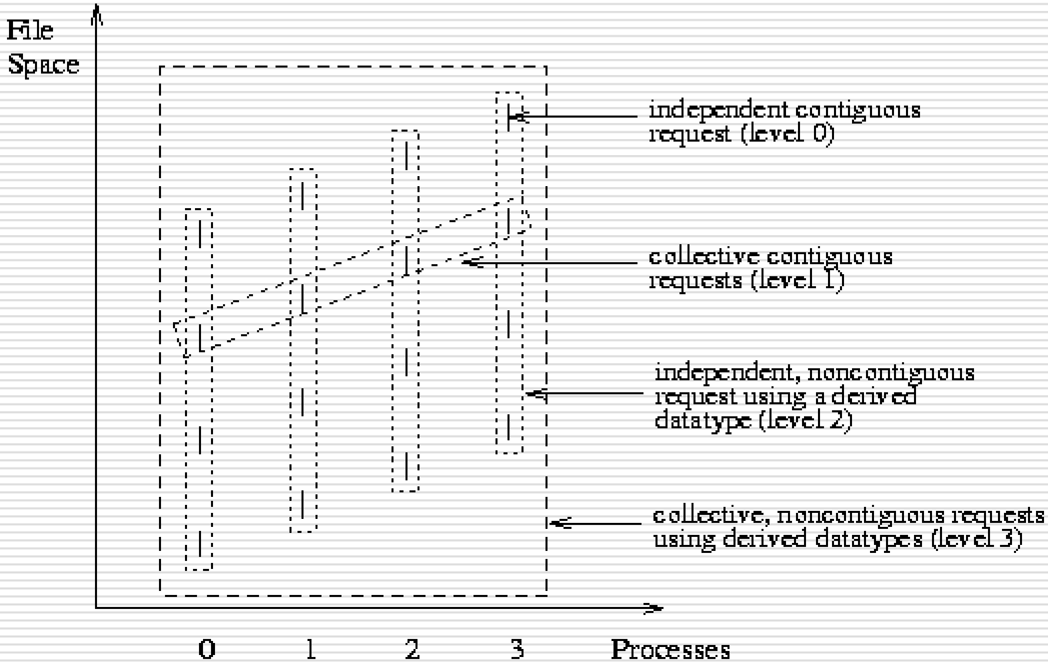
```
MPI_Type_create_subarray(..., &subarray, ...)
MPI_Type_commit(&subarray)
MPI_File_open(..., "filename", ..., &fh)
MPI_File_set_view(fh, ..., subarray, ...)
MPI_File_read(fh, local_array, ...)
MPI_File_close(&fh)
```

Level 2
(single independent, noncontiguous request)

```
MPI_Type_create_subarray(.., &subarray, ...)
MPI_Type_commit(&subarray)
MPI_File_open(MPI_COMM_WORLD, "filename", ..., &fh)
MPI_File_set_view(fh, ..., subarray, ...)
MPI_File_read_all(fh, local_array, ...)
MPI_File_close(&fh)
```

Level 3
(single collective, noncontiguous request)

Different levels of access



MPI I/O Optimizations

- 2 popular optimizations – data sieving and collective I/O

Optimizations for derived-datatype noncontiguous access

1. Data sieving
   - Make a few, large contiguous requests to the file system even if the user’s requests consists of several, small, nocontiguous requests
   - Extract (pick out data) in memory that is really needed
   - This is ok for read? For write?

     Read-modify-write along with locking
   - Use small buffer for writing with data sieving than for reading with data sieving. Why?

   Greater the size of the write buffer, greater the contention among processes for locks

Optimizations for derived-datatype noncontiguous access

1. Data sieving
2. Collective I/O
   - During collective-I/O functions, the implementation can analyze and merge the requests of different processes
   - **The merged request can be large and continuous although the individual requests were noncontiguous.**
   - Perform I/O in 2 phases:
     - I/O phase – processes perform I/O for the merged request. Some data may belong to other processes. If the merged request is not contiguous, use data sieving
     - Communication phase – processes redistribute data to obtain the desired distribution
   - Additional cost of communication phase can be offset by performance gain due to contiguous access.

- Data sieving and collective-I/O also help improve caching and prefetching in underlying file system

Collective I/O Illustration