MPI-Vector-IO: Parallel I/O and Partitioning for Geospatial Vector Data

Qin Feiran, SIST, ShanghaiTech University

Introduce

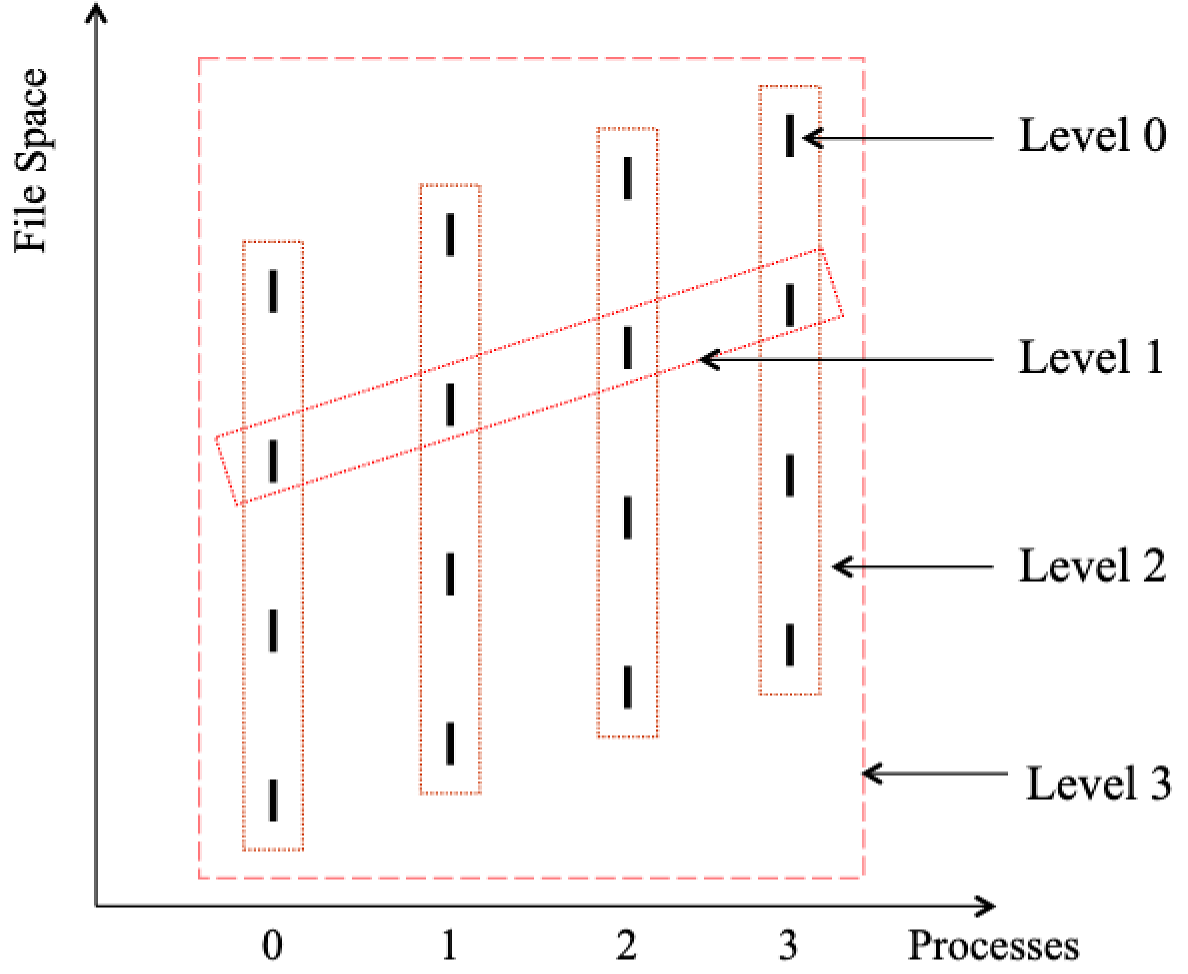
MPI-Vector-IO uses MPI to optimize parallel GIS systems.

GIS datasets are usually devided in grids, which can be optimized using **Partition read** and **Partition write**.

Backgrounds

- Parallel I/O and MPI-IO
  - Lusture: store as stripe (*stripe count* and *stripe size*)
  - MPI: N process to 1 file

The Four Levels of Access



Backgrounds

- MPI is faster than Hadoop
- Quadtree, R-tree, computational geometry and GIS algorithms and parsing WKT geometries are common in Geology.
- Spatial filtering and refinement

Challenges

1. Data Partitioning
   Partition with MPI affinity (collective and continuous)

2. Expressing vector data I/O using MPI
   A MPI read / write function designed for GIS. Existing function in MPI is not formatted
   `MPI_File_read_at(fh, offset, buf, nints, MPI_INT, &status);`

3.
```
int MPI_Send(const void* buf, int count, MPI_Datatype datatype,
int dest, int tag, MPI_Comm comm);
```

- `INT_MAX` => 2GB max file size or splitting file into small pieces / `BigMPI`

File Reading 放缩法

Algorithm 1 Iterative File Reading - Message based

```
Input variables: fileSize, blockSize, N, rank
MPI_Offset globalOffset ← 0
MPI_Offset fileChunkSize ← N * blockSize
iterations ← ⌈fileSize/fileChunkSize⌉
for (i=0; i <(iterations-1); i++) do
  globalOffset ← i * fileChunkSize
  start ← globalOffset + rank * blockSize
  MPI_File_read_at_all(file, start, fileBuffer, blockSize)
  lastDelimPos ← blockSize-1
  while (fileBuffer[lastDelimPos] != DELIMITER) do
    lastDelimPos--
  if (rank%2 == 0) then
    MPI_Send((fileBuffer+lastDelimPos),
             (blockSize-lastDelimPos), (rank+1)%N)
    MPI_Recv(recvBuffer, maxBufferSize, (rank-1+N)%N)
  else
    MPI_Recv(recvBuffer, maxBufferSize, (rank-1+N)%N)
    MPI_Send((fileBuffer+lastDelimPos),
             (blockSize-lastDelimPos), (rank+1)%N)
handleLastIteration()
```

Collective Computation and Communication for Spatial types

- Use MPI types to represent GIS types.

Spatial Type	Spatial Reduction	
	Operator	Supported Types
MPI_POINT	MPI_MIN	RECT, LINE, POINT
MPI_LINE	MPI_MAX	RECT, LINE, POINT
MPI_RECT	MPI_UNION	RECT

Table 2: Spatial data types and reduction operators.

- Collective Reduction Operators for Spatial types
- Communication buffer management
  - All to all and sliding window

Paralize

Filter and Refine based Spatial Join on two input datasets: L1, L2

1. Partition the datasets into file splits of blockSize bytes.
```
MPI_File_Partitioner partitioner(L1, L2);
pair<FileSplits, FileSplits> splits = partitioner.partition();
```
2. Parse the file splits into geometry objects.
```
Parser parser;
list<Geometry> L1Geoms_id, L2Geoms_id = parser.parse(splits);
```
3. **Filter step**:
   Spatial partitioning into uniform/adaptive grid cells.
```
Grid grid (numPartitions, universe);
grid.populateGridCells(L1Geoms_id, L2Geoms_id);
```
4. Generate a mapping from MPI process id to list of cells.
   (Default is block-cyclic).
5. Perform MPI all-to-all communication to exchange geometries.
   Each MPI process gets a list of (cellId, geoms) pairs.
   (Geometries are grouped by cell).
6. **Refine step**: Spatial join computation in each cell.
```
int refine(int cell, list<Geometry> L1_cell, list<Geometry> L2_cell);
```

Figure 7: Main steps for performing spatial computation using MPI-Vector-IO.

Experiments

- MPI communication plays important role
  - More stripes and more nodes brings significant overhead
- Polygony structure affects performance

Conclusion

- Split the polygony to take the advantage of MPI (Introduce some adaptive partitioning) since IO is performance-consuming.
- No NetCDF or HDF5.