ICON I/O Optimizations by the DWD

Nathanael Hübbe

2017-09-26

Table of Contents

Disk Data Layout

- Top level: time step and variables
- 2nd level: height level
- 3rd level: ground coordinate(s)
  - Structured Grid: 2D
  - Unstructured Grid: fused 1D ground coordinate

Memory Data Layout

- Top level: variables
- Decomposition layer: ground coordinate
- Local layer: height level ground coordinate

Table of Contents

Typical File-reading Code

```
InputFile* file = openInputFile("/path/to/datafile.grb");

read2D(file, "var1", gWidth, gHeight, gData->variable1);
read3D(file, "var2", gWidth, gHeight, gLevels, gData->variable2);
read2D(file, "var3", gWidth, gHeight, gData->variable3);
... //ten to fifty more lines like this,
... //some conditional, some nested in subroutine structures,
... //some mixed with processing of the read data

closeInputFile(file);
```

Whats wrong with this?

- Program dictates order of reading
  $\Rightarrow$ non-sequential access pattern
- File is a GRIB file
  $\Rightarrow$ no index, open call must build in-memory index
  $\Rightarrow$ data is read twice

What’s wrong with this? cont.

Benchmark Result reading a GRIB file with CDI:

- file open call: 6.363 s
- read calls: 6.343 s

Solution: File-driven Input

Changes in CDI:

- New CDI API that reads data as it "arrives" from file
- Iterator based interface, pseudo code:

```
CdiIterator* iterator = cdiIterator_new("path/to/datafile.grb");
while(!cdiIterator_nextField(iterator)) {
    //calls to introspect metadata
    if(wantThisField) {
        cdiIterator_readField(iterator, buffer, NULL);
        distributeData(buffer, ...);
    }
}
cdiIterator_delete(iterator);
```

New Infrastructure within ICON

`t_InputRequestList`

1. all variables of interest are registered
2. `inputRequesList_readFile()` reads all interesting data, scattering it to the processes
3. data is retrieved at the discretion of the calling code
    - perfectly local to the process, no MPI-calls triggered

Take-aways from file-driven input

- Iterator based CDI-Interface is nice...
- ...but ICON code restructuring was complex
    - Mainly due to conditional reading of fields depending on the presence of other fields
- File formats that don’t include an index are best avoided
- Useful spin-off: Typesafe use of CDI now possible

Table of Contents

Why Multifile Restart?

Benchmark shows: Concurrent access to many files is fastest

Chain-Job Restarts are written at the end of an integration
Reading always happens first thing in a job
$\Rightarrow$ Cannot be hidden behind computation

ICON is the only consumer of ICON restart files
$\Rightarrow$ No format restrictions

Typically, restarting PE N needs the data from original PE N
$\Rightarrow$ Multifile Restart reduces need to communicate

$\Rightarrow$ Multifile I/O is a perfect fit for restart data

Multifile Restart in ICON

File Structure

- Restart "file" is a directory: `restartFile.mfr/`
- `restartFile.mfr/attributes.nc` (global metadata)
- `restartFile.mfr/patch<JG>_metadata` (patch metadata)
- `restartFile.mfr/patch<JG>_<N>.nc` (payload data)
  Each payload file contains global indices of its points

Multifile Restart in ICON

Features

- Any number of compute/dedicated PEs may write
  $\Rightarrow$ Count of writers determines count of files
  - Each writer PE is exclusively responsible
    for a single file and a group of PEs
    $\Rightarrow$ Only communication among subgroups necessary
- Restart writing PEs may be disjunct of compute PEs
- Restart may be read with any process count
- Significant speedup seen in test case
  (unfortunately I couldn't perform any thorough testing)

Multifile Restart Take-aways

- Writing multifiles is easy
- Data redistribution when reading is a challenge
- Understanding legacy code is harder
- The effort pays off
- Dedicated restart writers don’t pay off

Summary

File-driven Input

- 2x speedup when reading GRIB files
- Typesafe CDI iterator interface

Multifile Restart in ICON

- Direct consequence of the benchmark results
- Avoids unnecessary reshuffling and communication
- Significant speedup achieved