12/20/11 Notes on AR impl for simulation

02/01/12

Key notions in Sam’s thesis, not to forget!

**resolution granularity:** offset from data rep to error rep; it also defines the partition size for decisions about resolution. In Sam’s thesis OpenGGCM data needs to have a granularity of 4 or 5 before the AR helps unless error tolerance (for whatever error measure he used?) got up to 15%. This makes sense, however, even though his calculations don’t really include anything “special” in terms of I/O. I think most of his times are generated by ViSit reconstruction from the silo format.

Subset of Sam’s work – his static case where data generation includes tolerance.

1. Primary goal is to decide what will NOT be saved; i.e., highest resolution needed for each region.
2. It’s a separate decision whether to do all of Sam’s computation at the same time simply to have access to the parallel configuration. We might want to do that, but it needs to be an alternative to “just as much as necessary” to write simulation output.
3. We’ve been thinking of error tolerance as the controlling factor; in principle, we could allow an alternative, which is a data size limit. Could even have 2 simultaneous guidelines: 1) Save every n-th time step based on error tolerance and 2) save some number of intermediate time steps based on a space tolerance.

Region definition options:

1. Pre-partition into regions; choose lowest acceptable resolution for each region; error measure probably needs to incorporate “max” somewhere. This is easiest
2. Follow octree implementation of Sam, but go down only 2-4 levels.

Output format options

1. n+1 files: where n is number of different resolution levels; 1 file is metadata that maps each source data region to a file and location in the file of the corresponding data.
2. Could output 2 files: all resolutions squashed together, with 1 metadata file; this would make a big file and complex accessing (and inefficient), but we could always do a later conversion to multifile format.
3. perhaps output complete files for lowest resolution(s) and sparse files for highest resolutions; the data reduction is so great, I can imagine that in most cases, only R0 and R1 will be sparse.

**SparseDataSource.java**

Why not create a Granite class: SparseDataSource or SparseBlockDataSource.

Version 1: all blocks must be same size, any subblock query that contains missing data throws exception?? Could have a method to say fill with a default value, but not worth implementing for now.

The mapping would be put into the xfdl file; it could get big if arbitrary block sizes/locations are allowed. Could have parent class that defaults to fixed partitioning, then later extend for more complex cases.

xml fields for fixed partitioning can be same fields used by ChunkedDataSource (does it currently store this information in xfdl file? )

Major question: does SparseDataSource have a simple binary file as a data member, or a 1D chunked data source? Advantage of ds member: all the code for hiding type and storage info and byte ordering, etc. is available. Disadv: does ChunkedDataSource really work and will it save any effort for code in SparseDataSource?

In order to be a public class, this class needs the semantics that every point has a value and that value should be defined by an instance variable that defaults to 0.

**ARDataSource.java**

I think we want a child of ARDataSource that contains an array of SparseDataSources. Sam’s current ARDataSource might actually already do that without having defined the class SparseDataSource.