Argonne NATIONAL LABORATORY

ParGAL Algorithm Re-design

Jayesh Krishna

Mathematics and Computer Science Division
Argonne National Laboratory
2013

U.S. DEPARTMENT OF ENERGY

Outline

- Background
- Limitations
- New Design
- Performance
- Code
- Future work

Background

- ParGAL implements algorithms for calculating
  - Dim_avg
  - Max/Min
  - Vorticity/Divergence
- The effort is to make the code more
  - Readable
  - Extensible
- The current work involves redesigning the trivial ParGAL algorithms
  - Vorticity/Divergence algorithms need to be revisited (may be modified slightly)
  - Dim_avg_n, Max/Min and the new algorithms will follow the new design

Limitations (before redesign)

- The algorithm functions were too long
- Dimension checks were not accurate
- No generic design
- Difficult to read at times
  - No description of the behavior (Not documentation)
- Difficult to extend
  - Variables with record dimensions
  - Variables with levels in the last dimension
  - Operations on MOAB sparse tags
  - MOAB tags with multiple sequences
- Code for handling missing values mixed with other code
- More C-like with a C++ interface

Design - Overall

- Fix the issues mentioned
- Re-think the algorithm, find the generic behavior
  - Algorithms operate on chunks of data
    - Iteration of input and result chunks are different
  - An operation is performed on each chunk of data
    - result = Op(input1, input2, …)
  - After operation on some chunks another reduction operation (s) is/are performed
    - result = Op2(result)
  - Sometimes we count the number of data values
    - Need a dimension-specific counter
- Incorporate the behavior of the code inside the code
  - Iterators
- Almost a complete re-write of the algorithm

Design - chunks, counters, dimensions & traits

- A tag chunk class to indicate mesh data chunks
  - No timesteps, just mesh data chunks (corresponds to a MOAB tag)
- Tag chunks are further divided to tag sequences
  - Iterate through chunks to get sequences
  - These sequences are iterable with little cost
  - All algorithms will now work with tag sequences
  - Remove the concept of working on timesteps (getting timesteps)
- A dimension-specific counter to count the number of values in chunks
  - There are chunk counters corresponding to tag chunks
  - There are sequence counters corresponding to tag sequences
  - All algorithms work with sequence counters

Design - chunks, counters, dimensions & traits

- A more accurate identification of dimensions
  - Look for dimension name (rather than position)
  - Incorporate partition info, if required, to the “dimension name”
    - Allows incorporating dimension-specific behavior into code
- Added traits for finding dimension-specific info at compile time
  - Traits classes carry information about other classes
  - Indicates the iterator type to use for that dimension (& partition method)
- Adding some traversal info to tag sequences
  - Some info required for data traversal is added to sequence classes
  - Iteration of data in mesh depends on the dimension (& partition method)
  - This information is used by algorithms based on behavior “defined” by marker iterators

Design - Iterators

- Polymorphic iterators
  - These iterators are used to iterate on chunks of data (chunk by chunk)
  - There are two types of iterators
    - General iterators – Just iterate over the different mesh data chunks. Used for iterating over the input variables
    - Reduction iterators – Encapsulate the “reduction iteration” behavior. Used for iterating over the result variable
  - Run-time cost (cannot be used for tight loops)
- Marker iterators
  - These iterators are NOT used to iterate on a chunk
  - The iterators “define” the behavior of the algorithms using them
  - The algorithms extract iteration info from tag sequences based on the behavior “defined” by the iterators
  - No run-time cost (but defines behavior – improves readability, reusability)

Design - Iterators

- Base marker iterators
  - moab_tag_static_iterator
    - Iteration is static for a number of steps
  - moab_tag_reset_iterator
    - Iteration resets after a number of steps
  - moab_tag_skip_iterator
    - Iteration skips after a number of steps
- The code uses iterators that are a combination of the basic iterators
  - To iterate through the input chunk sequences for dim_avg_n(time)
    - moab_tag_gen_dim_iterator – No specific base marker iterator behavior
  - To iterate though the result chunk sequences for dim_avg_n(time)
    - moab_tag_reset_dim_iterator – Reset after specific number of steps. The number of steps in this case would be number of vals in a timestep in the result
  - To iterate through the result chunk sequences for dim_avg_n(lev)
    - moab_tag_static_skip_dim_iterator – Remain “static” for nlevs and then “skip” nlevs

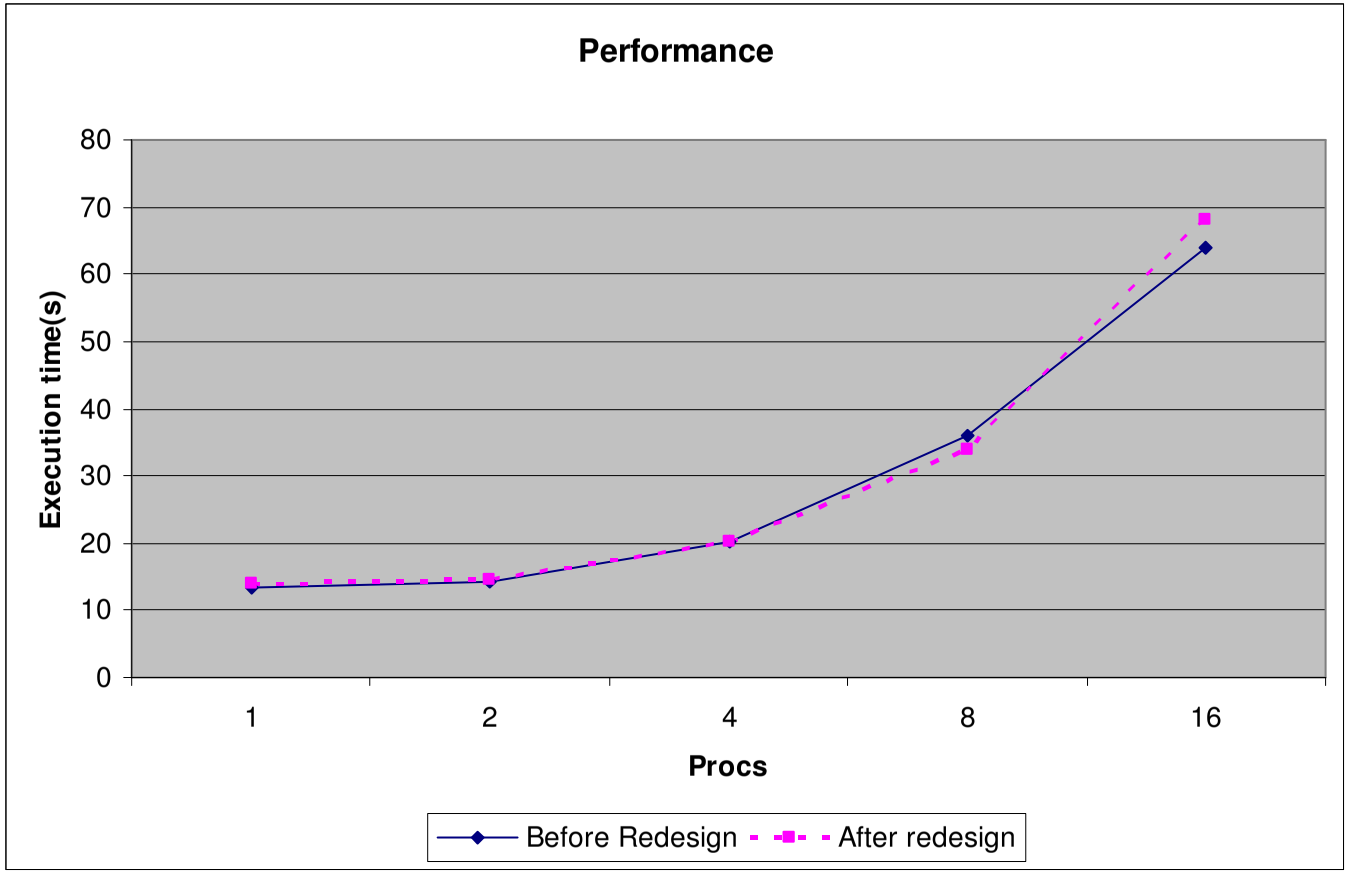
Replacement for pcvar.get_storage()

- pcvar.get_storage()/delete_storage() will be eventually replaced
  - cache_read()/cache_delete()
  - Already used in the chunk iterators

```
while(tstep < NUM_TSTEPS){
  void *buf = var.get_storage(tstep);
  // Work with buf
  var.delete_storage(tstep);
  tstep++;
}
```

```
bool cont = false;
while(tstep < NUM_TSTEPS){
  void *buf;
  std::size_t buf_len;
  moab::Range::iterator mb_iter;
  do{
    cont = var.cache_read(tstep, buf,
       buf_len, cont, mb_iter);
    // Work with buf
  }while(cont == true);
   var.cache_delete(tstep);
   tstep++;
}
```

Performance



- Time average of T on a ¼ deg FV grid for 1 yr

Code

- See code?

Future Work

- Incorporate the design to other algorithms
- Test with MPAS datasets
- The following algorithms are in the works
  - dim_min_*, dim_max_*, dim_sum_*, dim_product_*
- Add new algorithms
  - dim_avg_wgt*, dim_variance_*, dim_stddev_*

Thank You!

Questions?