GLOperationDescription

Jeffrey J. Early

October 7, 2013

1 Objectives

GLNumericalModelingKit is primarily composed of GLVariables and GLVariableOperations. From these two simple object classes, mathematical expressions can be described and computed.

There are, however, several reasons that we might want to record the object graph that is created when expressing a particular mathematical problem.

1. For computational efficiency, we don't want to unnecessarily repeat any operation and therefore we should cache any previous steps.
2. We may want to recover an equation saved to file.
3. We may want to display the mathematical steps in a human readable form (from the object graph).
4. A series of mathematical steps could actually be used to compile the program for another computer.

These objectives each require a different amount of information to be saved about each operation. At the bare minimum, we save the following information:

- the operation name,
- the GLVariable operands,
- the scalar operands (if any),
- the dimension operands (if any),
- the GLVariable result.

That amount of information would be enough to display the mathematical steps in human readable form, and it would also be enough to recover an equation saved to file, assuming the object graph includes the dimensions from which everything originates.

The only other pieces of information left off are

- the number of elements in the data chunks,
- the specific algorithm (block) being used,
- details about each variable.

However, these additional pieces of information are actually extra in some contexts. For example, if the algorithm starts with the specification of particular dimensions, then these additional attributes should be deterministically deduced given the object graph reproduced by the bare minimum information. These extra pieces of information are certainly useful for debugging, and they would be most important for future work where the object graph is used to compile MPI implementations for cluster scaling.

2 Implementation

The most pressing objective is to prevent repeated computations. For example, if the user attempts transformation a variable from one basis to another, we need to check that this has not already been done. The operand variable should then look in its cache to see if the operation has already been performed with the same arguments. Thus, each variable should have a mapping from variable operations to the result variable.

The GLVariableOperation class should return a GLOperationDescription object from the appropriate set of input variables. This could actually be a class method written at the abstract GLVariableOperation class level. As long as we always provide the operands in the same order (which we obviously should), then this will be fairly general. A subclass can implement a more detailed description, and provide additional information, if necessary. This implementation strategy has two drawbacks—a result variable cannot be returned, and the implementation details can't be returned (yet).

On the other hand, maybe we don't need a GLOperationDescription class. Perhaps this check just gets done after the GLVariableOperation has been created. A GLVariableOperation is considered equal to another operation if the operands are the same—and maybe even if the chosen implementation block is the same.

The implementation blocks should be enumerated, and the number specified. This allows us to determine exactly the implementation being used.

In my first implementation attempt, I simply implemented -isEqualToOperation: starting at the abstract GLVariableOperation class and working down to specific subclass implementations. Next, I implemented a cache of all operations in GLVariable. The problem is that any operations created manually (outside of GLVariable) are NOT then cached. Thus, operations that call another operation first do not get cached.

One could cache everything at the equation level.

Just before an operation returns self, it could check the various variables to see if the operation has already been cached. That would be the most robust, although you'd never be able to get around the cache. Yeah, I don't like that. When I create a new operation object, it had damn well better be creating a new operation object. So, only if you request an operation on a variable directly does it look for a cached version. That makes sense.

I ran into trouble with my cache. If an operation you create actually returns some sub-operation (IOW, an operation with an operand that isn't the variable in question) then adding it to the cache is obviously a bad thing to do.

Also, clearly differential operators need to be operators. Although, they also clearly need to be variables. Sheesh.

Deleting the differential operator caching at the variable level made things worse. Apparently I don't have this working very well yet.

3 Differential Operators