SAND2012-6714P

Removing Field Objects from Worklet Parameters

From Daxtoolkit

Currently in the master branch, worklets take "field objects" as parameters. These objects serve as handles to query fields. However, these objects are currently less important than I originally thought they would be. For example, they cannot be used to independently pull point field values. It was found to be more efficient to pull them all en mass, and it was seldom useful to pull only one.

Encouraged by other designers, I have explored the possibility of not using field objects at all. Instead, values are passed directly to worklets when invoked.

Contents

Obvious changes to worklets

Obviously, this change requires the prototype of all worklets to change. For example, the Square worklet changes from

```
template<class CellType, class FieldType, class ExecutionAdapter>
DAX_WORKLET void Square(
    dax::exec::WorkMapField<CellType, ExecutionAdapter> &work,
    dax::exec::FieldIn<FieldType, ExecutionAdapter> &inField,
    dax::exec::FieldOut<FieldType, ExecutionAdapter> &outField)
{
  FieldType inValue = work.GetFieldValue(inField);
  FieldType outValue = inValue * inValue;
  work.SetFieldValue(outField, outValue);
}
```

to

```
template<class FieldType>
DAX_WORKLET void Square(const FieldType &inValue,
                        FieldType &outValue)
{
  outValue = inValue * inValue;
}
```

As can be seen, the declaration changes significantly. First, the parameters take far less typing. Second, the field values do not need to be retrieved by a secondary query. Third, for reasons described later several template parameters like CellType and ExecutionAdapter no longer need to be declared.

I am lying here a little bit in that this declaration for a worklet is not exactly compatible with the final version. As described later, worklets have become functors.

Removing ExecutionAdapter

The ExecutionAdapter is a recent addition that was introduced to allow field access to adapt to both a DeviceAdapter and ArrayContainerControl (both of which may effect how arrays are accessed). Since all field values are queried before the worklet is called, the worklet itself no longer needs to have a template parameter that points to how field arrays are accessed.

Apart from array typedefs, the only other facility ExecutionAdapter provided (and the only one I can currently foresee needing) is a RaiseError method that allows error reporting to be based on the devices abilities. Although the serial device adapter had a special version that threw an exception, all other implementations used a universal method of writing to a shared array, which was checked after all threads finished. It seems easier for everyone to simply use this universal method for all device types and get rid of the pain of an ExecutionAdapter entirely.

Simultaneous use of multiple container types

One limitation of the ExecutionAdapter that is much more severe than I originally anticipated is that it constrains all arrays for a given worklet invocation to be the same type. If this were not the case, then worklet designers would have to necessarily have a separate template parameter for each parameter.

However, because fields and arrays are queried before the worklet is invoked, then this restriction can be mostly lifted. It is still necessary for "glue" code that invokes the worklet from the control environment, but this can all be internally written. In the future, the list of template parameters will probably itself be templates (not unlike the boost tuple metastructure).

Point coordinates (and implicit arrays)

Now that we can use ArrayHandle objects using different types of containers simultaneously, we have the option of creating special containers for special purposes. In particular, we can create separate containers with different access patterns or completely implicit arrays.

One simple example of an implicit array is an array of some size with the same value everywhere (say,

5). One way to implement this is to allocate the array and initialize all values to 5. But a much more efficient method is to create a special ArrayPort that returns 5 for any call to Get.

A much more practical example of an implicit array is that for point coordinates of uniform grids. From the compact description of a uniform grid's topology, origin, spacing, and extent, any point coordinate is easily computed. Previously to implement this important implicit array we had a special version of field objects for point coordinates and special methods to get point coordinates of uniform grids. Now instead the UniformGrid class has a GetPointCoordinates method that returns an ArrayHandle just like the UnstructuredGrid class. The difference, however, is that UniformGrid's ArrayHandle has a container that implicitly generates the point coordinates.

Changed nature of work objects

The previous implementation of work objects spent most of its interface in providing the query functions for getting and setting field values. It acted as the gatekeeper to the field arrays. However, with field values queried before the worklet call, none of these are necessary. The only thing really used in the work objects is the RaiseError method, which holds enough state to find the shared error string.

Since the work objects are so stripped down, I took the opportunity to implement a suggestion given a few months back of making worklets be actual functors that inherit from the work object. The advantages of this is that it is easier for templated methods to discover the worklet type and that the worklets can hold some state that can be propagated to all invocations (for example, the min/max values in threshold).

Because worklets are inheriting from them, the names of the work objects were changed to worklet (to support the is-a relationship). There is a WorkletBase class (in the internal namespace), and all actual worklet types are trivial subsets of WorkletBase.

As alluded to before, because worklets are functors, their declarations change accordingly. So the real implementation of the Square worklet is

```
class Square : public dax::exec::WorkletMapField
{
public:
  template<class ValueType>
  DAX_EXEC_EXPORT
  void operator()(const ValueType &inValue,
                  ValueType &outValue) const
  {
    outValue = inValue * inValue;
  }
};
```

Changed Schedule parameters

This idea of holding state within a functor object has lead to another suggested change, which is the removal of the parameters argument to the DeviceAdapter's Schedule function. This simplifies the

scheduling a bit and discourages the practice of having separate parameters and functor methods.

Acknowledgements

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.