

Using Tpetra without CUDA UVM

Karen Devine for the Tpetra Team

Geoff Danielson, Karen Devine, Tim Fuller, Jonathan Hu, Brian Kelley,
Kyungjoo Kim, Chris Siefert, Timothy Smith

Updated: November 23, 2021

SAND2021-15110 PE

The Trilinos team has removed the requirement for UVM usage

- Motivation:
 - New platforms may or may not have reliable UVM-like capabilities
 - Debugging application and system issues with UVM is difficult
 - Explicit memory management should avoid performance surprises
- Trilinos still works with UVM enabled
 - And UVM enabled remains default for CUDA builds
 - But applications need to remove use of deprecated code and behavior
 - Build with
 - D Tpetra_ENABLE_DEPRECATED_CODE=OFF
 - D Kokkos_ENABLE_CUDA=ON
 - D Kokkos_ENABLE_CUDA_UVM=OFF

Biggest change: Tpetra manages sync / modify between host and device

- Tpetra has Kokkos::DualViews of matrix and vector data
- New Tpetra class WrappedDualView manages the sync / modify flags between host and device views
- Users no longer sync / modify explicitly
- Users cannot hold both host and device pointers concurrently
- Affects MultiVector, CrsMatrix, CrsGraph, and Block variants

Example: vector fill with UVM is straightforward

```
// Without UVM, this code will fail
multivector_t mv(...);
auto mvData =
    mv.getLocalViewHost();

for (j = 0; j < numData; j++)
    mvData(j,0) = rhs(j);

myDeviceFunction(mv);
```

*Code worked with UVM
but failed without UVM*

Without UVM, careful management of host and device views is needed

Without UVM, explicit modify/syncs were needed – messy and error-prone

```
multivector_t mv(...);
auto mvData =
    mv.getLocalViewHost();
mv.clear_sync_state();
mv.modify_host();
for (j = 0; j < numData; j++)
    mvData(j,0) = rhs(j);
mv.sync_device();
myDeviceFunction(mv);
```

Without UVM, careful management of host and device views is needed

Without UVM, explicit modify/syncs were needed – messy and error-prone

Tpetra now manages the sync/modify state for users

```
multivector_t mv(...);
auto mvData =
    mv.getLocalViewHost();
mv.clear_sync_state();
mv.modify_host();
for (j = 0; j < numData; j++)
    mvData(j,0) = rhs(j);
mv.sync_device();
myDeviceFunction(mv);
```

```
multivector_t mv(...);
{ auto mvData =
    mv.getLocalViewHost(
        Tpetra::Access::OverwriteAll);

    for (j = 0; j < numData; j++)
        mvData(j,0) = rhs(j);
}
myDeviceFunction(mv);
```

Key changes for Tpetra::MultiVector users (details to follow)

1. Capture host and device views in separate scopes
 - Don't hold raw pointers to multivector's data
 - Let views go out of scope as soon as you're done working with them
2. Separate scope for local operations and Trilinos operations on an object
 - Trilinos operations can choose where to access data
3. Indicate intended usage of views
 - ReadOnly, ReadWrite, OverwriteAll
4. Reduce switching between host and device accesses
 - Be aware of data synchronization

Key changes for Tpetra::CrsGraph/CrsMatrix users (details to follow)

Same as MultiVector

1. Capture host and device views in separate scopes
 - Don't hold raw pointers to data
 - Let views go out of scope as soon as you're done working with them
2. Separate scope for local operations and Trilinos operations on an object
 - Trilinos operations can choose where to access data
3. Indicate intended usage of views
 - ReadOnly, ReadWrite, OverwriteAll
4. Reduce switching between host and device accesses
 - Be aware of data synchronization
5. `getLocalMatrix*()` and `getLocalGraph*()` build Kokkos' matrix and graph ON DEMAND now (rather than returning stored data structures); use wisely
6. Functions returning `Teuchos::ArrayView` of `CrsMatrix/CrsGraph` data are dangerous and deprecated
7. Functions returning raw pointers to `CrsMatrix/CrsGraph` data are dangerous and deprecated

#1: Capture host and device views in separate scopes

```
// NOT OK  
  
auto v_h = mv.getLocalViewHost(tag);  
  
auto v_d = mv.getLocalViewDevice(tag);
```

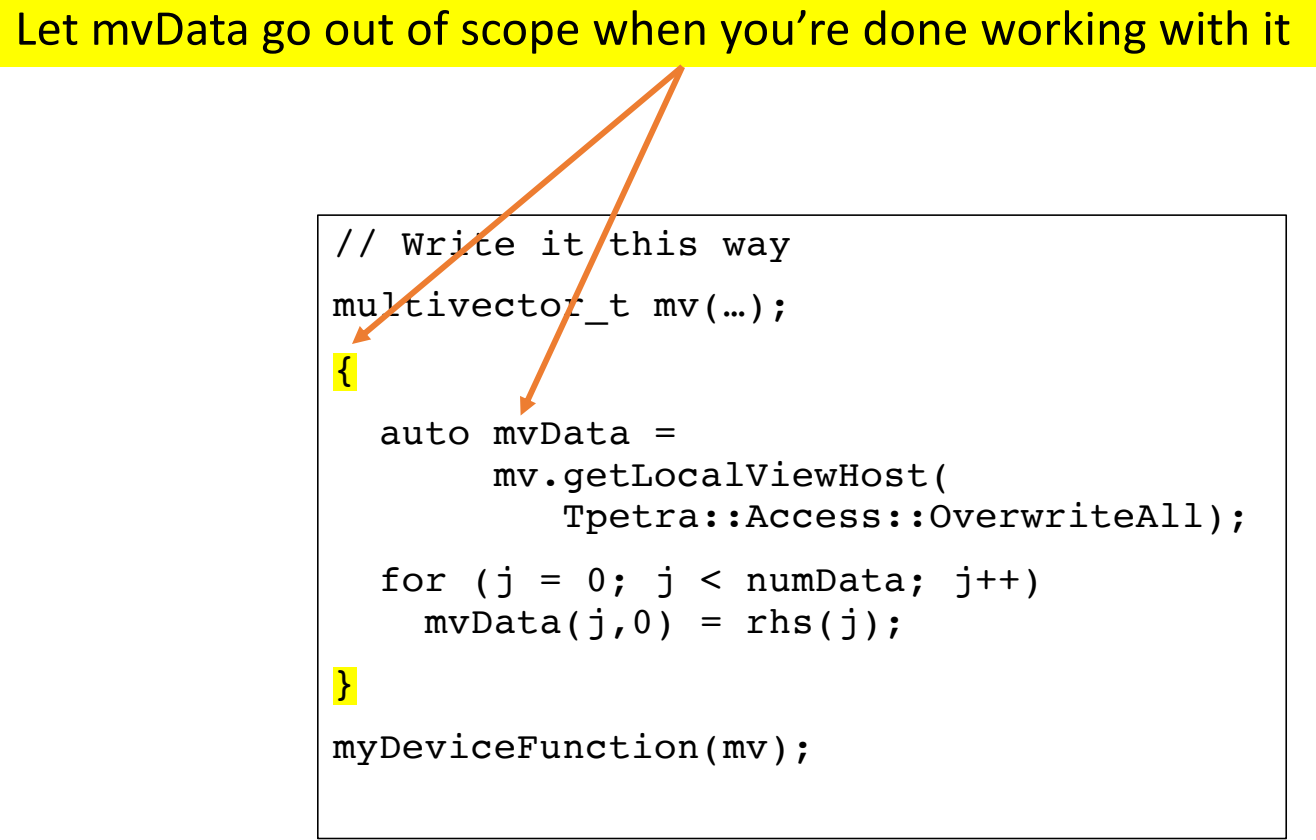
```
// OK  
{  
    auto v_h = mv.getLocalViewHost(tag);  
}  
{  
    auto v_d = mv.getLocalViewDevice(tag);  
}
```

Tpetra will track reference counts, including for subviews, on host and device to prevent simultaneous access

Example: Correct scoping in vector fill

Let mvData go out of scope when you're done working with it

```
// Write it this way
multivector_t mv(...);
{
    auto mvData =
        mv.getLocalViewHost(
            Tpetra::Access::OverwriteAll);
    for (j = 0; j < numData; j++)
        mvData(j,0) = rhs(j);
}
myDeviceFunction(mv);
```



Scoping rules apply to existing ArrayRCP interfaces, too

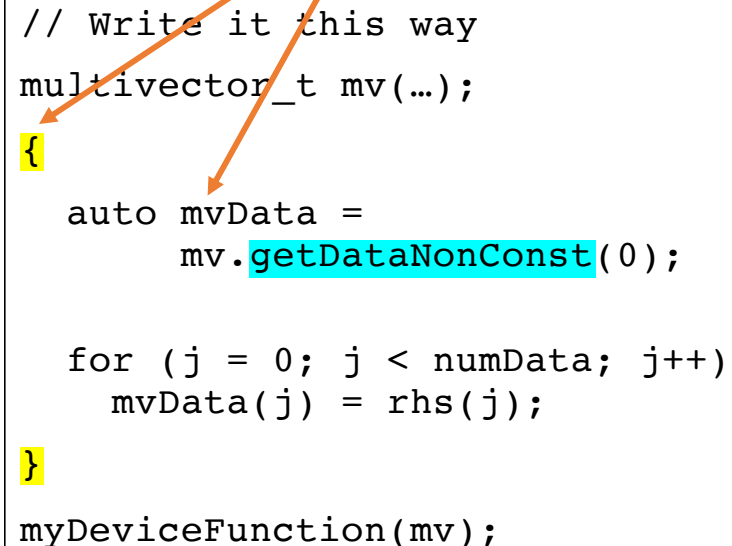
Let mvData go out of scope when you're done working with it.

Get an ArrayRCP (1D or 2D):

getData, getDataNonConst
get1dView, get1dViewNonConst
get2dView, get2dViewNonConst

```
// Write it this way
multivector_t mv(...);
{
    auto mvData =
        mv.getDataNonConst(0);

    for (j = 0; j < numData; j++)
        mvData(j) = rhs(j);
}
myDeviceFunction(mv);
```

The diagram shows a code block with a yellow highlight on the opening curly brace of a function scope. Two orange arrows originate from the yellow highlight box above the code. One arrow points to the opening curly brace of the function scope, and the other points to the closing curly brace of the function scope, illustrating that the variable mvData is only defined within this scope.

Don't hold/grab/hand-out pointers to raw data

```
// DANGER DANGER DANGER

typename
AbstractConcreteMatrixAdapter<
    Tpetra::RowMatrix<Scalar, LocalOrdinal, GlobalOrdinal, Node>, DerivedMat>
    ::super_t::spmtx_vals_t
AbstractConcreteMatrixAdapter<
    Tpetra::RowMatrix<Scalar,
                      LocalOrdinal,
                      GlobalOrdinal,
                      Node>,
    DerivedMat>::getSparseValues() const
{
    typename super_t::local_matrix_t lm = this->mat_->getLocalMatrixHost();
    return lm.values.data();
}
```

Tpetra can not track use of raw data() pointer; cannot sync appropriately
Applies to CrsGraph, CrsMatrix, MultiVector

#2: Separate scope for local operations and Tpetra operations on an object

```
// NOT OK

auto v_h = mv.getLocalViewHost(tag);
doStuffOnHost(v_h);

mv.doExport(...);
```

```
// OK

{
    auto v_h = mv.getLocalViewHost(tag);
    doStuffOnHost(v_h);
}

mv.doExport(...);
```

Trilinos operations (e.g., doExport) may choose to use host or device

#3: Indicate intended usage of views

Tpetra syncs as needed for type of access

- Tpetra::Access::ReadOnly
 - Tpetra syncs if needed
- Tpetra::Access::ReadWrite
 - Tpetra syncs if needed
 - Tpetra marks modified
- Tpetra::Access::OverwriteAll
 - Tpetra syncs only if view is a subview
 - Tpetra marks modified
 - Use only if writing ALL entries of view

```
// Use access tags to indicate intent
{
    auto read_h =
        mv.getLocalViewHost(
            Tpetra::Access::ReadOnly);
    auto readwrite_h =
        mv.getLocalViewHost(
            Tpetra::Access::ReadWrite);
    auto write_h =
        mv.getLocalViewHost(
            Tpetra::Access::OverwriteAll);
}
```

Access tags allow Tpetra to manage sync/modify status for users

Subview OverwriteAll may sync anyway

- Kokkos DualViews share modify flags with their subviews
- When sync'ing a subview, need to sync the entire view
- Subview with OverwriteAll access will behave as if ReadWrite to prevent corruption of other subviews

Will behave as if ReadWrite

```
// Write it this way
multivector_t mv(map, 3);
auto mySubVec =
    mv.getVectorNonConst(2);
{
    auto mySubData =
        mySubVec.getLocalViewHost(
            Tpetra::Access::OverwriteAll);

    for (j = 0; j < numData; j++)
        mySubData(j) = rhs(j);
}
myDeviceFunction(mv);
```

#4: Reduce switching between host and device accesses

Syncs mv to host in EVERY iteration

```
// Lots of data transfer
multivector_t mv(map, 3);

for (int v = 0; v < 3; v++) {
    // Fill vector on host; use it on device
    {
        auto mySubVec =
            mv.getDataNonConst(v);

        for (j = 0; j < numData; j++)
            mySubVec(j) = rhs(j);
    }
    myDeviceFunction(mySubVec);
}
```

Syncs mv to device in EVERY iteration

Syncs mv to host in FIRST iteration

```
// Write it this way
multivector_t mv(map, 3);

for (int v = 0; v < 3; v++) {
    // Fill all vectors on host
    auto mySubVec =
        mv.getDataNonConst(v);

    for (j = 0; j < numData; j++)
        mySubVec(j) = rhs(j);
}
// Use all vectors on device
myDeviceFunction(mv);
```

Syncs mv to device once

#5: Use CrsGraph::getLocalGraph() conservatively

```
// NOT Efficient

auto numRows =
    g.getLocalGraph().row_map.extent(0)-1;
auto nnz =
    g.getLocalGraph().entries.extent(0);

auto rowptr = g.getLocalGraph().row_map;
auto colidx = g.getLocalGraph().entries;
```

```
// Better

auto numRows = g.getNodeNumRows();
auto nnz = g.getNodeNumEntries();

auto lclGraph = g.getLocalGraphDevice();
auto rowptr = lclGraph.row_map;
auto colidx = lclGraph.entries;
```

getLocalGraphHost/Device() builds graph ON DEMAND now, rather than returning a stored pointer

#5: Use CrsMatrix::getLocalMatrix() conservatively

```
// NOT Efficient
```

```
rowptr = m.getLocalMatrix().graph.row_map;  
colidx = m.getLocalMatrix().graph.entries;  
values = m.getLocalMatrix().values;
```

```
// Better
```

```
auto mlocal = m.getLocalMatrixDevice();  
rowptr = mlocal.graph.row_map;  
colidx = mlocal.graph.entries;  
values = mlocal.values;
```

getLocalMatrixHost/Device() builds KokkosSparse::CrsMatrix ON DEMAND now, rather than returning a stored pointer

#6: Returned Teuchos::ArrayViews are dangerous and deprecated

```
// Deprecated  
  
m.getLocalRowView(row, indices_AV,  
                 values_AV);  
m.getGlobalRowView(row, indices_AV,  
                  values_AV);  
  
m.getLocalRowCopy(row, indices_AV,  
                 values_AV);  
m.getGlobalRowCopy(row, indices_AV,  
                  values_AV);
```

```
// New interface returns Kokkos::Views  
  
m.getLocalRowView(row, indices_KV,  
                 values_KV);  
m.getLocalRowCopy(row, indices_KV,  
                 values_KV);  
m.getGlobalRowView(row, indices_KV,  
                  values_KV);  
m.getGlobalRowCopy(row, indices_KV,  
                   values_KV);
```

Tpetra cannot track usage of data in Teuchos::ArrayView for sync/modify; use Kokkos::Views instead

#7: Returned raw pointers are dangerous and deprecated

```
// Deprecated  
  
m.getLocalRowView(row, indices_raw,  
                  values_raw, nentries);
```

```
// New interface returns Kokkos::Views  
  
m.getLocalRowView(row, indices_KV,  
                  values_KV);
```

Tpetra cannot track usage of data in raw pointers for sync/modify; use Kokkos::Views instead

Other deprecations will follow, but will be less disruptive

Deprecations:

- Fewer Teuchos::ArrayRCPs, ArrayViews in interfaces; more Kokkos Views
- Greater reliance on access tags (e.g., Tpetra::Access::ReadWrite) instead of function naming conventions (e.g., getDataNonConst and getData)
- More consistent naming (unambiguous Host/Device in function names, "Local" vs "Node", etc.)

Impact on applications / packages:

- Changes easily adopted by applications and packages (name changes rather than logic changes)
- Will be deprecated as time/staff permits
- Will be summarized and documented on wiki

For more info

- Email

- tpetra-developers@software.sandia.gov
- kddevin@sandia.gov

- Wiki

- Tpetra info: <https://github.com/trilinos/Trilinos/wiki/Tpetra-Information-Page>
- UVM removal info: <https://snl-wiki.sandia.gov/display/TRIL/UVM+Removal>