Removal of the UVM Requirement from Tpetra: MultiVector and BlockMultiVector

Karen Devine for the Tpetra Team

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

Updated: March 23, 2021

SAND2021-2718 PE

We are removing the requirement for UVM usage in Tpetra

- 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
- Timeline:
  - Tpetra MultiVector and BlockMultiVector nearly complete; PR expected in February
  - Tpetra CrsMatrix/CrsGraph underway; expect completion in March
  - Downstream Trilinos packages underway; expect completion in FY21
- Trilinos will still work with UVM enabled
  - But applications will need to remove use of deprecated code and behavior

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 Tpetra operations on an object
   - Tpetra (and, indeed, 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. Update code to remove use of deprecated interfaces

Expect similar changes for CrsGraph / CrsMatrix

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 works with UVM
but fails without UVM

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

```
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, explicit modify/syncs are needed – messy and error-prone

To ease transition to non-UVM, Tpetra will manage the sync/modify state for users

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

```
// With new design this code will fail
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, explicit modify/syncs are needed – messy and error-prone

To ease transition to non-UVM,
Tpetra will manage the sync/modify state for users

With new design, this code will fail.

#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);
}
```

MultiVector 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

```
// With new design this code will fail
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);
```

```
// 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

Get an ArrayRCP (1D or 2D):
getData, getDataNonConst
get1dView, get1dViewNonConst
get2dView, get2dViewNonConst

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

```
// With new design this code will fail
multivector_t mv(…);
auto mvData =
     mv.getDataNonConst(0);
mv.clear_sync_state();
mv.modify_host();
for (j = 0; j < numData; j++)
   mvData(j) = rhs(j);
mv.sync_device();
myDeviceFunction(mv);
```

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

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

#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(…);
```

Tpetra operations (e.g., doExport), and, indeed, all Trilinos operations 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

Functions accepting access tags:

- `getLocalViewHost`
  `getLocalViewDevice`
  `getLocalView<>`
- `getLocalBlock`

```
// 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

Example: using Access tags in vector fill

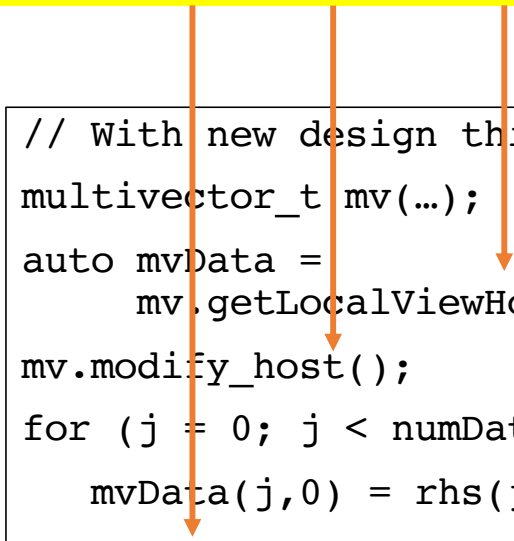
Use Access tags to allow Tpetra to manage sync/modify state

```
// With new design this code will fail
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);
```

```
// 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);
```

Functions getLocalView*, getLocalBlock without Access tags are deprecated

These functions are deprecated



```
// With new design this code will fail
multivector_t mv(…);
auto mvData =
     mv.getLocalViewHost();
mv.modify_host();
for (j = 0; j < numData; j++)
   mvData(j,0) = rhs(j);
mv.sync_device();
myDeviceFunction(mv);
```

```
// 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);
```

NonConst ArrayRCP interfaces are ReadWrite (including sync)

Get an ArrayRCP (1D or 2D):
getData, getDataNonConst
get1dView, get1dViewNonConst
get2dView, get2dViewNonConst

Sync/modify behavior is as with getLocalViewHost(Tpetra::Access::ReadWrite)

```
// With new design this code will fail
multivector_t mv(…);
auto mvData =
     mv.getDataNonConst(0);
mv.modify_host();
for (j = 0; j < numData; j++)
   mvData(j) = rhs(j);
mv.sync_device();
myDeviceFunction(mv);
```

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

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

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

```
// 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);
```

Will behave as if ReadWrite

#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 Update code to remove use of deprecated interfaces

For now, most interfaces remain

- Get an ArrayRCP (1D or 2D):
  - `getData, getDataNonConst`
  - `get1dView, get1dViewNonConst`
  - `get2dView, get2dViewNonConst`
- Get a single column as Vector:
  - `getVector, getVectorNonConst`

Removed without deprecation

- `Tpetra::withLocalAccess`
- `Tpetra::for_each`
- `Tpetra::transform`

Deprecated interfaces

- Accessors without Access tags
  - `getLocalViewHost()`
  - `getLocalViewDevice()`
  - `getLocalView<>()`
  - `getLocalBlock()`
- Sync/modify now handled by MultiVector
  - `mv.sync_host(), mv.sync_device(), mv.sync<>()`
  - `mv.modify_host(), mv.modify_device(), mv.modify<>()`
  - `mv.clear_sync_state()`

For more info

- Email
  - tpetra-developers@software.sandia.gov
  - kddevin@sandia.gov
- Github
  - Branch trilinos/TpetraDualViewRefactor
  - https://github.com/trilinos/Trilinos/issues/8591
  - https://github.com/trilinos/Trilinos/pull/8353
- Wiki
  - https://snl-wiki.sandia.gov/display/TRIL/UVM+Removal