Argonne NATIONAL LABORATORY

MINI TUTORIAL: PETSC ON THE GPU

JUNCHAO ZHANG
jczhang@anl.gov
Argonne National Laboratory

June 6, 2023

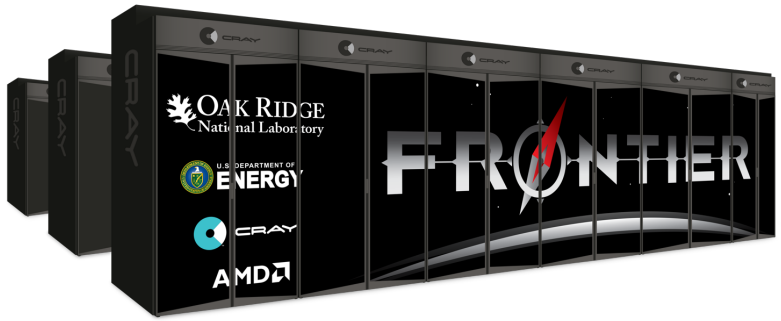
Outline

- Why should I consider using PETSc on GPUs
- PETSc GPU backends
- Sketch of PETSc GPU backend implementations
- Configure, build, run and profile PETSc tests on GPUs
- Port your PETSc code to GPUs
- Use external libraries with GPUs

Why should I consider using PETSc with GPUs?

- To ride the wave and get the power!
- 7 out of the top 10 supercomputers as of today, including the No.1 Frontier@OLCF, use GPUs to provide the *majority* of their peak performance

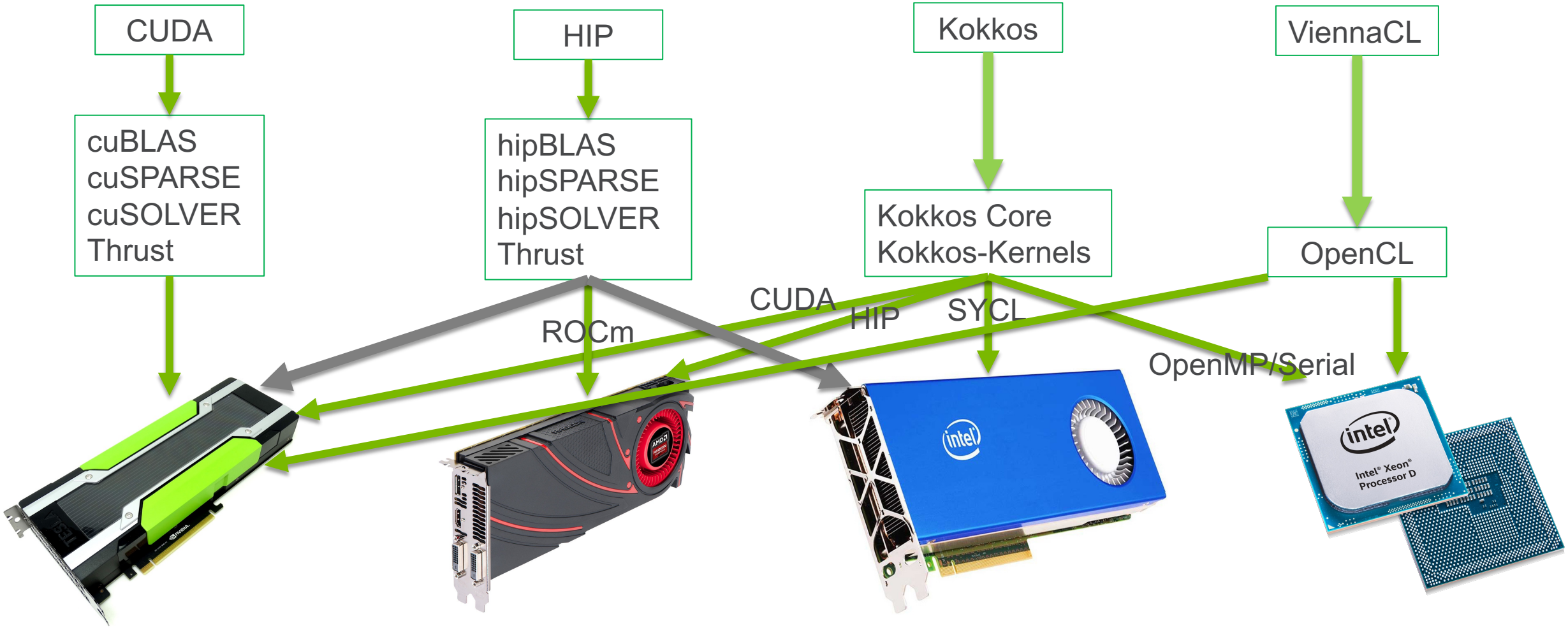
AMD EPYC + NVIDIA A100



AMD EPYC + AMD MI250X

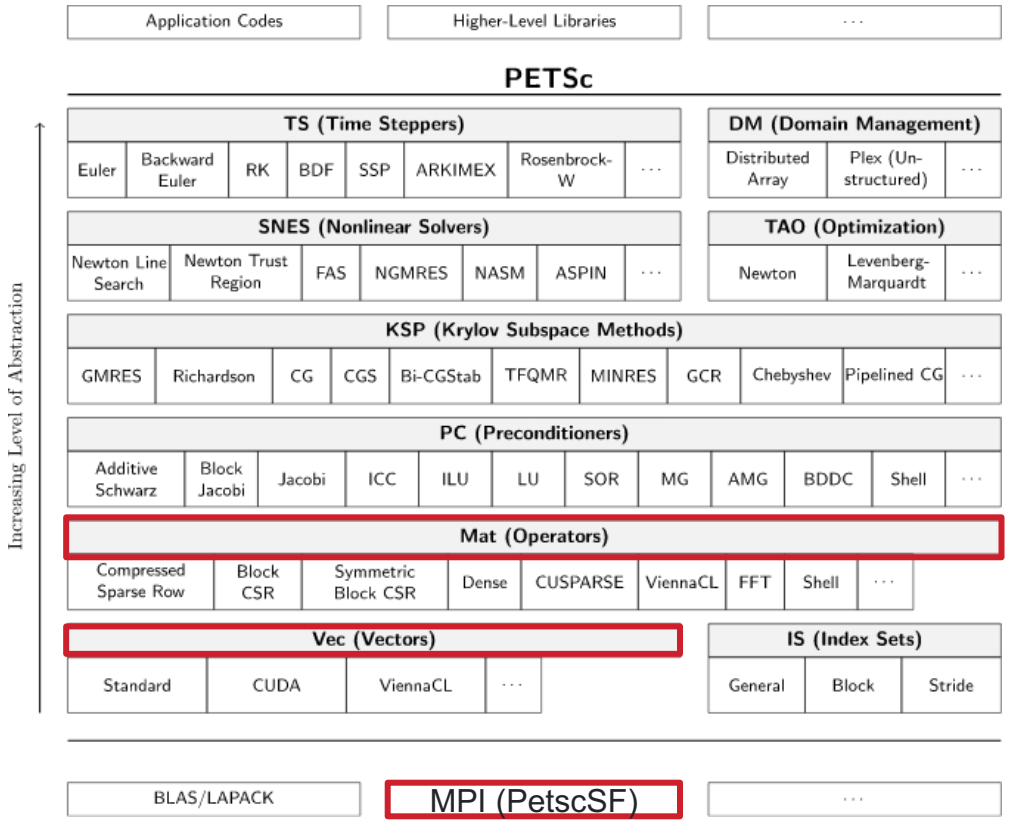
Intel Xeon + Intel PVC

PETSc GPU backends



Sketch of PETSc GPU backend implementations

- Use GPU-aware MPI to communicate data on device (via PetscSF)
  - -use_gpu_aware_mpi {0, 1}
- Add device matrix/vector types to offload their operations
  - MatAIJ{CUDA, HIP, Kokkos}
  - MatDense{CUDA, HIP}
  - Vec{CUDA, HIP, Kokkos}
- Device objects always have a host counterpart to fall back with (plan B)



The tale of twin arrays since we assume separate host/device memory

```
typedef struct _p_Vec {
  struct _VecOps ops[1];

  void *data;


  void *spptr;



  PetscOffloadMask offloadmask;

} * Vec;
```

- Function pointers installed by a Vec impl. (could mix host and device)

```
(*norm)(Vec,NormType,PetscReal*);
(*dot)(Vec,Vec,PetsScalar*);
(*setvalues)(..);
…
```

`(*norm)(Vec,NormType,PetscReal*);` → `VecNorm_SeqKokkos`

`(*setvalues)(..);` → `VecSetValues_Seq`

- Specific implementations on host, e.g.

```
struct Vec_Seq {
 PetscScalar *array; // host array
 …
};
```

- Associated specific implantations on device, e.g.

```
struct Vec_SeqKokkos {
  PetscScalarKokkosDualView v_dual;
  …
};
```

```
PETSC_OFFLOAD_{CPU,GPU,BOTH}
```

Maintain the flag via accessors like VecGetArrayRead/Write()

Configure / Build / Run / Profile PETSc with GPUs

```
$ ./configure
--PETSC_ARCH=arch-gpu-opt
--with-cc=mpicc
--with-cxx=mpicxx
--with-{cuda, hip, sycl}
--with-{cudac, hipc, syclc}={nvcc, hipcc, icpx}
--with-{cuda, hip, sycl}-arch={80, gfx908, pvc}
--download-kokkos
--download-kokkos-kernels
--download-superlu_dist // will configure with GPU

# see more examples under config/examples/

$ make PETSC_ARCH=arch-gpu-opt

$ make check PETSC_ARCH=arch-gpu-opt
```

```
$ mpiexec -n 4 ./ex1 -mat_type aij -vec_type standard

# petsc uses lazy GPU initialization; possible to run pure CPU tests on compute nodes without GPUs even with a GPU-built PETSc


$ mpiexec -n 4 ./ex1 -mat_type aijcusparse -vec_type cuda

$ mpiexec -n 4 ./ex1 -mat_type aijhipsparse -vec_type hip

$ mpiexec -n 4 ./ex2 -dm_mat_type aijkokkos -
dm_vec_type kokkos


# run tests requiring CUDA and find GPU examples

$ make test query=requires queryval=cuda -f gmakefile.test
```

Profiled with –log_view and got NaN?

```
$ cd src/ksp/ksp/tutorials
$ make ex1
$ ./ex1 -pc_type eisenstat -ksp_monitor_short -ksp_gmres_cgs_refinement_type refine_always -mat_type aijcusparse -vec_type cuda -log_view
```

```
Event                Count      Time (sec)     Flop                              --- Global ---  --- Stage ----  Total   GPU    - CpuToGpu -   - GpuToCpu - GPU
                   Max Ratio  Max     Ratio   Max  Ratio  Mess   AvgLen  Reduct  %T %F %M %L %R  %T %F %M %L %R Mflop/s Mflop/s Count   Size   Count   Size  %F
---------------------------------------------------------------------------------------------------------------------------------------------------------------
--- Event Stage 0: Main Stage
VecMDot               24 1.0   nan nan      1.30e+04 1.0 0.0e+00 0.0e+00 0.0e+00  0 75  0  0  0   0 75  0  0  0  -nan     -nan      0 0.00e+00   20 7.04e-04  13
VecNorm               15 1.0   nan nan      2.85e+02 1.0 0.0e+00 0.0e+00 0.0e+00  0  2  0  0  0   0  2  0  0  0  -nan     -nan      0 0.00e+00    0 0.00e+00 100
VecScale              14 1.0   nan nan      1.40e+02 1.0 0.0e+00 0.0e+00 0.0e+00  0  1  0  0  0   0  1  0  0  0  -nan     -nan      0 0.00e+00    0 0.00e+00 100
…
KSPSetUp               1 1.0   nan nan      0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0  -nan     -nan      0 0.00e+00    0 0.00e+00  0
KSPSolve               2 1.0 4.2383e-01 1.0 1.72e+04 1.0 0.0e+00 0.0e+00 0.0e+00 63 99  0  0  0  63 99  0  0  0     0     -nan     43 2.19e-03   41 2.38e-03  25
KSPGMRESOrthog        12 1.0   nan nan      1.49e+04 1.0 0.0e+00 0.0e+00 0.0e+00  1 86  0  0  0   1 86  0  0  0  -nan     -nan     24 7.36e-04   20 7.04e-04  24
PCSetUp                2 1.0   nan nan      0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0  -nan     -nan      0 0.00e+00    0 0.00e+00  0
PCApply               14 1.0   nan nan      1.40e+02 1.0 0.0e+00 0.0e+00 0.0e+00  0  1  0  0  0   0  1  0  0  0  -nan     -nan     16 1.28e-03    0 0.00e+00  0
```

Profile with –log_view_gpu_time –log_view

```
$ ./ex1 -pc_type eisenstat -ksp_monitor_short -ksp_gmres_cgs_refinement_type refine_always -mat_type aijcusparse -vec_type cuda -log_view_gpu_time -log_view
```

```
Event                Count      Time (sec)     Flop                              --- Global ---  --- Stage ----  Total   GPU    - CpuToGpu -   - GpuToCpu - GPU
                   Max Ratio  Max     Ratio   Max  Ratio  Mess   AvgLen  Reduct  %T %F %M %L %R  %T %F %M %L %R Mflop/s Mflop/s Count   Size   Count   Size  %F
---------------------------------------------------------------------------------------------------------------------------------------------------------------
--- Event Stage 0: Main Stage
VecMDot               24 1.0 7.7655e-04 1.0 1.30e+04 1.0 0.0e+00 0.0e+00 0.0e+00  0 75  0  0  0   0 75  0  0  0    17       9      0 0.00e+00   20 7.04e-04  13
VecNorm               15 1.0 8.3041e-04 1.0 2.85e+02 1.0 0.0e+00 0.0e+00 0.0e+00  0  2  0  0  0   0  2  0  0  0     0       0      0 0.00e+00    0 0.00e+00 100
VecScale              14 1.0 2.2795e-04 1.0 1.40e+02 1.0 0.0e+00 0.0e+00 0.0e+00  0  1  0  0  0   0  1  0  0  0     1       1      0 0.00e+00    0 0.00e+00 100
…
KSPSetUp               1 1.0 7.9050e-06 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0       0      0 0.00e+00    0 0.00e+00  0
KSPSolve               2 1.0 4.2426e-01 1.0 1.72e+04 1.0 0.0e+00 0.0e+00 0.0e+00 65 99  0  0  0  65 99  0  0  0     0       3     43 2.19e-03   41 2.38e-03  25
KSPGMRESOrthog        12 1.0 1.1733e-03 1.0 1.49e+04 1.0 0.0e+00 0.0e+00 0.0e+00  0 86  0  0  0   0 86  0  0  0    13      11     24 7.36e-04   20 7.04e-04  24
PCSetUp                2 1.0 4.4500e-04 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0     0       0      0 0.00e+00    0 0.00e+00  0
PCApply               14 1.0 3.1747e-04 1.0 1.40e+02 1.0 0.0e+00 0.0e+00 0.0e+00  0  1  0  0  0   0  1  0  0  0     0       0     16 1.28e-03    0 0.00e+00  0
```

```
$ ./ex1 -pc_type eisenstat -ksp_monitor_short -ksp_gmres_cgs_refinement_type refine_always -mat_type aijcusparse -vec_type cuda -log_view_gpu_time -log_view :filename.txt:ascii_flamegraph
```

Port your PETSc code to GPUs

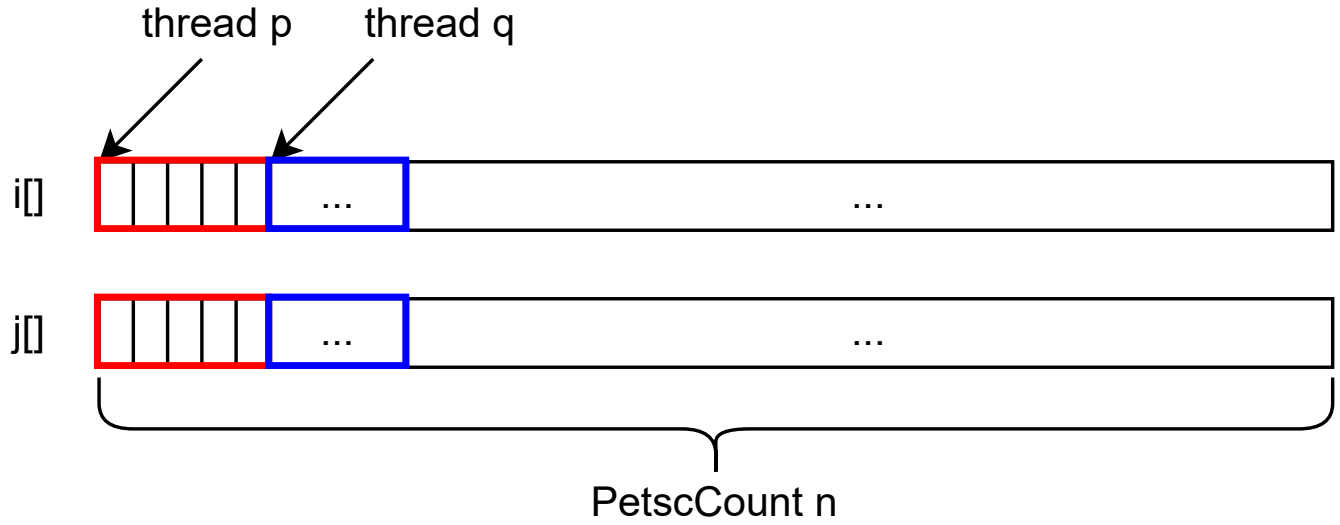
- Do nothing but with `-mat_type {aijcusparse, aijhipsparse, aijkokkos} -vec_type {cuda, hip, kokkos}`
- Offload matrix assembly to GPUs
- Offload PETSc callback functions, like `DMSNESSetJacobian/Function()`, to GPUs
  - Access device data of PETSc vectors
- Get the device stream PETSc is using
- Compile your device code with PETSc makefile

Matrix assembly on GPUs

- Host API `MatSetValues(A,m,idxm,n,idxn,v,addv)`, which adds individual entries to the matrix, is not a good prototype for GPUs.
  - Must avoid data races among threads inserting to the same location
  - Inefficient to do binary search to find insert location
  - Entries might be remote
- We added new split phase matrix coordinate (COO) assembly APIs
  - User gives coordinates of nonzeros in one batch in the first phase
  - PETSc builds the sparsity pattern, insertion plan and communication plan
  - User gives values of nonzeros in the same order in the second phase

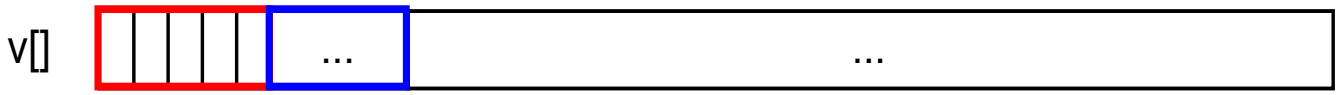
Matrix coordinate (COO) assembly

- `MatSetPreallocationCOO(Mat A,PetscCount n,PetscInt i[],PetscInt j[])`



- n can be 64-bit
- i/j[], provided by users on host, can be freed after the call
- Use *global* indices and can point to remote entries
- Negative indices are ignored

- `MatSetValuesCOO(Mat A,const PetscScalar v[],InsertMode imode)`



- v[], on device, has entries in the same order as i/j[]
- Very efficient: entries in v[] are directly inserted to A or send buffer

- Support `MatDuplicate(A,op,&B);`

  `MatSetValuesCOO(B,…);`

Access vector data in callbacks

1. Traditional approach
   - `VecGetArray(Vec x, PetscScalar **a)`
     - Return latest data on host even when x is a device vector
2. Device specific approach (snes/tutorials/ex47cu.cu)
   - `VecCUDAGetArray(Vec x, PetscScalar **a)`
     - x must be of type VECCUDA
3. Device neutral way
   - `VecGetArrayAndMemType(Vec x, PetscScalar **a, PetscMemType *mtype)`
     - Return an array and its memory type depending on the vector type
       - `PETSC_MEMTYPE_{HOST, CUDA, HIP, SYCL}`
     - Users can then switch-case on the memory type to write their *device-specific* code

Access vector data in callbacks (cont.)

4. Another device neutral approach (snes/tutorials/ex55.kokkos.cxx)
   - `VecGetKokkosView(Vec x, Kokkos::View<PetscScalar *, MemorySpace> *kv)`
   - `DMDAVecGetKokkosOffsetView(DM da, Vec x, Kokkos::Experimental::OffsetView<PetscScalar*,MemorySpace>* kv)`
     - Return a Kokkos 1~4D view with the latest data in the given memory space
     - OffsetView supports global indices
   - Then write your callbacks in Kokkos

Example: snes/tutorials/ex55.c

```
PETSC_EXTERN PetscErrorCode
FormFunctionLocalVec(DMDALocalInfo *info,Vec x,Vec f,AppCtx *user);


int main() {


DMDASNESSetFunctionLocalVec(da,INSERT_VALUES,(DMDASNESFunctionVec)FormFun
ctionLocalVec,&user);


}
```

Example: snes/tutorials/ex55.kokkos.cxx

```
PetscErrorCode FormFunctionLocalVec(DMDALocalInfo *info,Vec x,Vec f,AppCtx *user) {
    ConstPetscScalarKokkosOffsetView2D xv; // da is 2D; read-only xv
    PetscScalarKokkosOffsetView2D      fv; // write-only fv

    DMDAVecGetKokkosOffsetView(info->da,x,&xv);
    DMDAVecGetKokkosOffsetViewWrite(info->da,f,&fv);

    Kokkos::parallel_for ("FormFunctionLocalVec",
       MDRangePolicy <Rank<2,Iterate::Right,Iterate::Right>>({ys,xs},{ys+ym,xs+xm}),
       KOKKOS_LAMBDA (PetscInt j,PetscInt i){
       …
       fv(j, i) = 2.0 * (hydhx + hxdhy) * (xv(j, i) - mms_solution);

    });

    DMDAVecRestoreKokkosOffsetView(info->da,x,&xv);
    DMDAVecRestoreKokkosOffsetViewWrite(info->da,f,&fv);

}
```

Get the device stream PETSc is using

```
#include <petscdevice.h>
PetscDeviceContext dctx;
PetscDeviceType    type;
void               *handle;
PetscDeviceContextGetCurrentContext(&dctx);
PetscDeviceContextGetStreamHandle(dctx, &handle);
PetscDeviceContextGetDeviceType(dctx, &type);
if (type == PETSC_DEVICE_CUDA) {
    cudsStream_t stream = *(cudaStream_t*)handle;
    my_cuda_kernel<<<1, 2, 3, stream>>>(); }
else if (type == PETSC_DEVICE_SYCL) {
    sycl::queue myq = *(sycl::queue*)handle;
    ...
}
```

Compile your device code with PETSc makefile

- See https://petsc.org/release/manual/getting_started/#writing-c-c-or-fortran-applications on using PETSc make rules and variables
- File suffixes used by PETSc makefile

Source files	Compiled by
*.c	Host C or C++ compiler
.F/.F90	Fortran compiler
*.cxx	Host C++ compiler
*.cu	CUDA compiler
*.hip.cpp	HIP compiler (e.g., hipcc)
*.sycl.cpp	SYCL compiler (e.g., icpx)
*.kokkos.cxx	Kokkos compiler

Use external libraries with GPUs

- Ideally, if an external package X supports GPUs, then
  - `--download-X` should configure X with GPU support if PETSc GPU is on
  - If a PETSc code is run with GPU (e.g, `-mat_type aijcusparse`), then we should directly pass device data to the package
- Current status
  - `--download-{hypre, superlu_dist, strumpack}` supports GPU
  - PETSc passes device data directly to hypre
  - PETSc passes host data to superlu_dist, strumpack, and they then compute on GPUs

Thank you!