Using GPUs with PETSc

Patrick Sanan (Argonne National Laboratory, ETH Zurich)
On behalf of the PETSc team (Particular thanks to Richard Mills)

CIG Developers's Workshop, 2022-02-28

Slide source and full example configuration and logs at
https://gitlab.com/psanan/petsc-gpu-cig-2022-02-28

PETSc - the Portable, Extensible Toolkit for Scientific computation

Application Codes

Higher-Level Libraries and Frameworks

PETSc

TS Time Steppers
- Pseudo-Transient, Runge-Kutta, IMEX, SSP, ...
- Local and Global Error Estimators
- Adaptive Timestepping
- Event Handling
- Sensitivity via Adjoints
- ...

TAO Optimization Solvers
- PDE-Constrained
- Adjoint-Based
- Derivative-Free
- Levenberg-Marquardt
- Newton's Method
- Interior Point Methods
- ...

SLEPc Eigensolvers

SNES Nonlinear Solvers
- Newton Linesearch
- Newton Trust Region
- BFGS (Quasi-Newton)
- Nonlinear Gauss-Seidel
- Successive Substitution
- Nonlinear CG
- Active Set VI
- ...

KSP Linear Solvers
- CG, GMRES, BiCGStab, FGMRES, ...
- Pipelined Krylov Methods
- Hierarchical Krylov Methods
- ...

DM Domain Management
- **DMDA** Regular Grids
- **DMPlex** Unstructured Meshes
- **DMForest** Forest-of-octrees AMR
- **DMStag** Staggered Grids
- **DMNetwork** Networks
- **DMSwarm** Particles

PC Preconditioners
- ILU/ICC
- Additive Schwarz
- Fieldsplit (Block Preconditioners)
- PCMG (Geometric Multigrid)
- GAMG (Algebraic Multigrid)
- ...

Vec Vectors

IS Index Sets

Mat Linear Operators
- AIJ (Compressed Sparse Row)
- SAIJ (Symmetric)
- BAIJ (Blocked)
- Dense
- GPU Matrices
- ...

PetscSF Parallel Communication

Communication and Computational Kernels

MPI | BLAS/LAPACK | Kokkos | CUDA | . . .

This Presentation

- Quickly motivate the use of GPUs for scientific computing
- Show you how a first example with GPUs in PETSc
- That's it, since time is short, but tell you how to get more help and information.

What's a GPU?

- GPU = "Graphics Processing Unit", designed to:
  - Do lots of small, simple computations on pixels or triangles/quads
  - Do so fast enough to draw things on a monitor
  - Do so without consuming too much power
  - Ultimately produce 2D arrays of values in buffers to be displayed
- Thus, they
  - Prioritize *throughput* over *latency*
  - Perform best when many cores are active (the workload is highly parallel)
  - Have simpler (more energy-efficient) processing elements than CPUs
  - Have large numbers of processing elements
  - Don't need fast data transfers anywhere except to and from their own memory.
- Scientific computing uses GPUs as General Purpose GPUs (GPGPUs)
- GPUs are now available which support things, like double precision floating point numbers and fast GPU-GPU communication, that the original GPUs had little need for.

Why does PETSc care about GPUs?

- GPUs can do more computational work per dollar or Watt than CPUs, for certain workloads
- Some of these workloads align with the kinds of solvers PETSc focuses on
- They are increasingly available/usable
- So, supporting them can help our users solve their problems faster.

Why it's not so easy, under the hood

- Performance (these days) is so much about data movement, and GPUs can have severe data movement bottlenecks, so the library needs to have a tightly-integrated abstraction of GPUs and cannot completely hide their existence from the user.
- This is the same situation with two of the other main things that PETSc helps manage
  1. MPI parallelism - not all solvers can be parallelized this way.
  2. Domain Management (`DM`) - scalable solvers often need to know the geometry of the domain

How PETSc uses GPUs

- Provides several new implementations of PETSc's `Vec` distributed vector class[1] which allow data storage and manipulation in device (GPU) memory
- A "lazy-mirror" model
- Embue all `Vec` (and `Mat`) objects with the ability to track the state of a second "offloaded" copy of the data, and synchronize these two copies of the data (only) when required.

Host and Device Data

```
struct _p_Vec {
  ...
  void              *data;        // host buffer
  void              *spptr;       // device buffer
  PetscOffloadMask offloadmask;   // which copies are valid
};
```

Possible Flag States

```
  typedef enum {PETSC_OFFLOAD_UNALLOCATED,
                PETSC_OFFLOAD_GPU,
                PETSC_OFFLOAD_CPU,
                PETSC_OFFLOAD_BOTH} PetscOffloadMask;
```

[1] Richard Tran Mills, Mark F. Adams, Satish Balay, Jed Brown, Alp Dener, Matthew Knepley, Scott E. Kruger, Hannah Morgan, Todd Munson, Karl Rupp, Barry F. Smith, Stefano Zampini, Hong Zhang, and Junchao Zhang. *Toward Performance-Portable PETSc for GPU-based Exascale Systems*. 2020. arXiv: 2011.00715 [cs.MS].

Configuring PETSc for use with GPUs

- Key: you need GPU-aware MPI (e.g. CUDA-aware MPI) if you want good performance on more than one GPU. Your cluster hopefully has this, and locally you can use `--download-openmpi`
- Key `./configure` options
  - `--with-cuda`
  - `--download-kokkos --download-kokkos-kernels`
- Figure out the basics of how one compiles and runs code in C and CUDA (or HIP or SYCL..) on your machine of interest and use this to inform your configuration.
- But do not suffer too much! By clearly documenting your attempt in an email to `petsc-maint@mcs.anl.gov`, with `configure.log` and `make.log`, most problems can be quickly resolved.

Single-GPU system and configuration used here

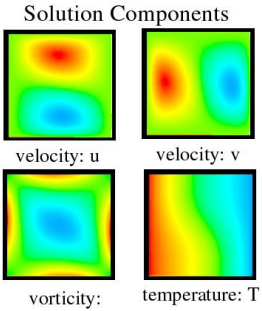
- Red Hat Linux `4.18.0-240.22.1.el8_3.x86_64`
- PETSc `main v3.16.4-987-gac50153c78`
- GPU: NVIDIA Titan Xp (expensive, advertised peak memory bandwidth: 548 GB/s)
- CPU: Intel® Core™ i5-4460 (cheap, advertised peak memory bandwidth: 25.6 GB/s)
- PETSc configuration (not all required for these demos)

```
./configure \
--with-cc=gcc --with-cxx=g++ --with-fortran=gfortran \
--with-fortran-bindings=0 --download-fblaslapack=1 \
--download-openmpi \
--download-kokkos --download-kokkos-kernels \
--download-viennacl --download-libceed --download-strumpack \
--download-scalapack --download-metis --download-suitesparse \
--download-cmake \
--with-cuda=1 \
--with-debugging=0 \
--COPTFLAGS=\"-g -O3 -march=native\" \
--CXXOPTFLAGS=\"-g -O3 -march=native\" \
--FOPTFLAGS=\"-g -O3 -march=native\" \
--CUDAOPTFLAGS="-O3" \
```

Example

SNES ex19

Velocity-vorticity formulation for nonlinear driven cavity. This is what runs with `make check` after you configure PETSc, and can be solved nicely leveraging multigrid.



$$
\begin{aligned}
-\Delta U - \partial_y \Omega &= 0 \\
-\Delta V + \partial_x \Omega &= 0 \\
-\Delta \Omega + \nabla \cdot ([U\Omega, V\Omega]) - \mathrm{Gr}\ \partial_x T &= 0 \\
-\Delta T + \mathrm{Pr}\ \nabla \cdot ([UT, VT]) &= 0
\end{aligned}
$$

- No-slip (zero Dirichlet) boundary conditions for U, V,
- Vorticity Ω on the boundary from $\Omega = -(\nabla_y U + \nabla_x V)$,
- insulated top and bottom, and
- fixed temperature on the left and right.

Example

```
$ # Set PETSC_DIR and PETSC_ARCH (if not using a prefix install)
$ cd $PETSC_DIR/src/snes/tutorials
$ make ex19
$ $PETSC_DIR/lib/petsc/bin/mpiexec -n 1 ./ex19 \
  -da_refine 7 \
  -snes_monitor -snes_converged_reason \
  -pc_type mg -mg_levels_pc_type jacobi
```

```
lid velocity = 0.000106281, prandtl # = 1., grashof # = 1.
  0 SNES Function norm 1.036007954337e-02
  1 SNES Function norm 3.547498241914e-05
  2 SNES Function norm 1.031529071036e-09
  3 SNES Function norm 1.145567739037e-14
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 3
Number of SNES iterations = 3
```

Example: Log

Run a CPU-only case [2] and add PETSc's `-log_view` - more in the Profiling chapter in the manual.

```
$ # Set PETSC_DIR (and PETSC_ARCH if not using a prefix install)
$ cd $PETSC_DIR/src/snes/tutorials
$ make ex19
$ $PETSC_DIR/lib/petsc/bin/mpiexec -n 2 ./ex19 \
  -da_refine 7 \
  -log_view :log_7_ref.txt \
  -pc_type mg -mg_levels_pc_type jacobi
```

If we look in `log_7_ref.txt`, we can see the time for the main nonlinear solve:

```
---------------------------------------------- ...
Event                Count      Time (sec)     ...
                   Max Ratio   Max     Ratio    ...
---------------------------------------------- ...
...
SNESSolve              1 1.0 4.2338e+00 1.0 ...
...
```

[2] Using 2 ranks since PETSc's `make streams` implies that this is enough to saturate the available memory bandwidth.

Using command-line options to run using CUDA

Once PETSc is configured `--with-cuda`, by simply using command line options, one may move some operations to the GPU. This is not expected to give optimal performance, but on some systems (like the one I'm using here), this does give some "free speedup":

```
$ # Set PETSC_DIR and PETSC_ARCH (if not using a prefix install)
$ cd $PETSC_DIR/src/snes/tutorials
$ make ex19
$ $PETSC_DIR/lib/petsc/bin/mpiexec -n 1 ./ex19 \
  -da_refine 7 \
  -log_view :log_7_cuda.txt \
  -pc_type mg -mg_levels_pc_type jacobi \
  -dm_vec_type cuda -dm_mat_type aijcusparse
```

```
---------------------------------------------- ...
Event                Count      Time (sec)     ...
                   Max Ratio  Max     Ratio    ...
---------------------------------------------- ...
...
SNESSolve              1 1.0 1.4922e+00 1.0 ...
...
```

Using command-line options to run using CUDA via Kokkos

Once PETSc is configured `--with-cuda --download-kokkos --download-kokkos-kernels`, one can use `Vec`s and `Mat`s backed by Kokkos views. The big advantage here is that you can implement operations on this data which is backend-portable, and can also interface with optimized libraries like LibCEED[3] which provide Kokkos-based kernels (amongst others).

```
$ # Set PETSC_DIR and PETSC_ARCH (if not using a prefix install)
$ cd $PETSC_DIR/src/snes/tutorials
$ make ex19
$ $PETSC_DIR/lib/petsc/bin/mpiexec -n 1 ./ex19 \
  -da_refine 7 \
  -log_view :log_7_kokkos.txt \
  -pc_type mg -mg_levels_pc_type jacobi \
  -dm_vec_type kokkos -dm_mat_type aijkokkos
```

```
--------------------------------------------- ...
Event                Count      Time (sec)     ...
                   Max Ratio  Max     Ratio    ...
--------------------------------------------- ...
...
SNESSolve              1 1.0 2.0729e+00 1.0 ...
...
```

[3] https://ceed.exascaleproject.org/libceed/

Comparing Performance

- We know from earlier that we expect GPUs to work best for large problems, so let's increase the problem size a bit by adding more levels of refinement (and levels to our multigrid hierarchy).
- SNES Solve times:

Levels	DOF	Time (ref)	Time (CUDA)	Time (Kokkos)
7	592900	4.23	1.49	2.07
8	2365444	18.3	5.32	7.06
9	9449476	76.5	21.3	28.1

- Let's take a look at the 9-level logs to see where the speedup is really happening, e.g. `MatMult`

Levels	DOF	Time (ref)	Time (CUDA)	Time (Kokkos)
9	9449476	42.0	3.76	3.80

Seeing CPU-GPU transfers in the logs

The most challenging part of extracting good performance from a PETSc solve using GPUs is to minimize GPU-GPU transfers. This can be a bit tricky with the lazy-mirror model, but `--log-view` has recently been improved to show transfers in every log stage.

```
------------------ ... --------------------------------
Event              ...  - CpuToGpu -   - GpuToCpu - GPU
                   ...  Count   Size   Count   Size  %F
------------------ ... --------------------------------

...
SNESSolve          ...  1430 2.19e+04  787 7.94e+03 91
...
```

See the full log on the Gitlab repository for this talk.

How to get support

- petsc.org
- Our mailing lists are very responsive. Send your `-log_view` output if you can.
  - petsc-users@mcs.anl.gov (public, preferred)
  - petsc-maint@mcs.anl.gov (private, use if you have big or sensitive attachments)

Thank you!