Some comments on M3D-C1 on GPUs

Jin recent tests indicate poor performance when using PETSc on GPU’s

- Is the problem size large enough to be meaningful? It takes a lot of work to keep a GPU busy – Would think real M3D-C1 3D problems are large enough to keep an appropriate number of GPU’s busy.
- 78% of the time spent in [CUDA memcpy HtoD] and [CUDA memcpy DtoH] – this is memory movement between host (CPU?) and Device (GPU?).
- Consider operations involved in an analysis step
  - Form element matrices – Where is this done?
  - System Matrix and RHS vector assembly – Assume this is on CPU (to our knowledge PETSc does not support this on the GPUs
  - Solve system – using PETSc on GPUs
- If there is CPU to GPU communication done for each element for assemble it is not a surprise that 78% of the time is spent in memory movement.

Some comments on M3D-C1 on GPUs

For good GPU performance you want to get all the data needed on the GPU at the start of the process and do everything there.

Getting all the calculations for M3D-C1 onto GPUs is a big job and needs:

- Execution of element level calculations on GPU – version done for many core is a starting point, but likely only a starting point for good GPU performance (at least that is what people tell me)
- Mesh data is also an issue – PUMI not suited to GPU execution of operations, Omega_h (RPI Ph.D. thesis by Dan Ibanez – now at Sandia) was a first step at a GPU enabled unstructured mesh adaptation capability
- GPU based assembly – we are discussing this with Mark Adams (A PETSc developer)

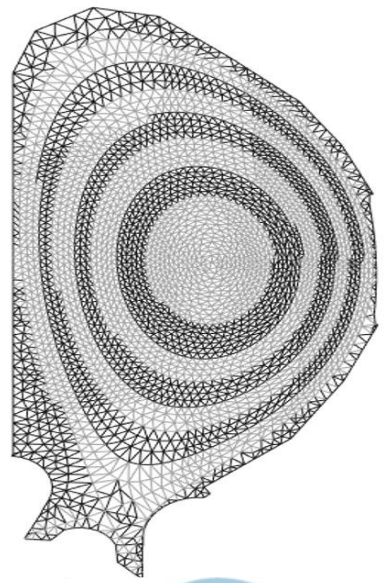
We are building some experience in this direction as part of our work on the unstructured mesh centric version of XGC (PIC code) we are developing.

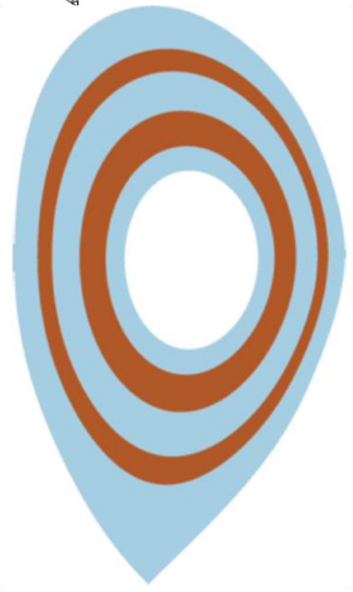
Implementation of XGCm gyrokinetic Poisson solver on GPU

Prepared by Chonglin Zhang

XGCm Gyrokinetic Poisson equation implementation on GPU

- P1 finite element space, with fixed linear matrix and varying right-hand-side vector every time step (key point here is we form the assembled matrix on the CPU and send it over once – currently only the RHS gets updated on each step)
- Unstructured mesh is based on Omega_h mesh structure/library, and PUMIPic library:
  - https://github.com/SNLComputation/omega_h
  - https://github.com/SCOREC/pumi-pic
  - Omega_h mesh structure handles mesh field related operations on GPU device; (Push and RHS formation is based on unstructured mesh level operations)
  - Omega_h mesh and associated mesh field data are completely stored on GPU memory;
  - Omega_h is built on top of Kokkos library, which could be used in different GPU architectures, MPI/OpenMP environments.
- For a linear system with fixed matrix, the right-hand side vector update is performed on GPU, according to Omega_h and PUMIPic mesh structure



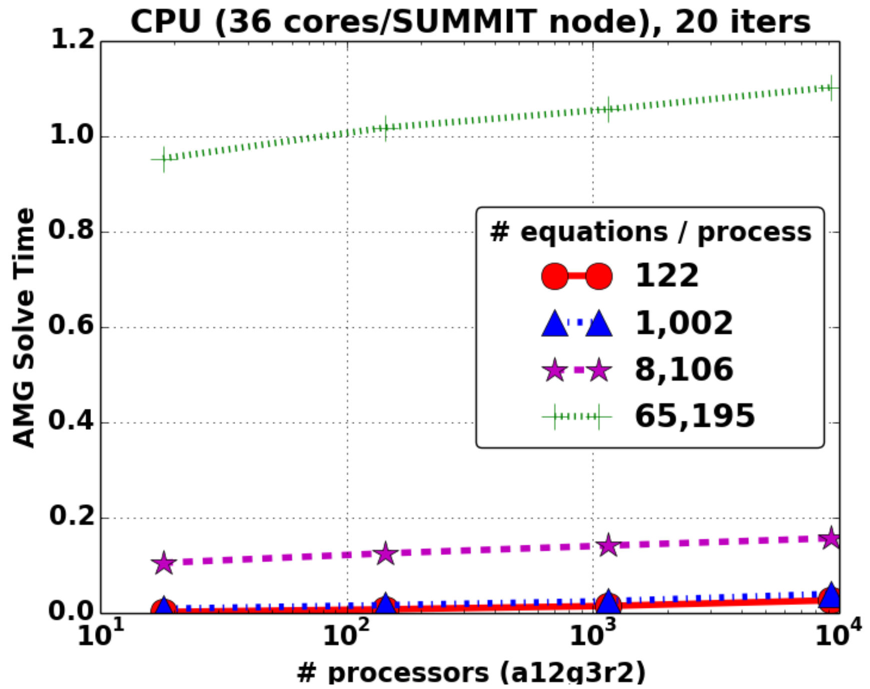


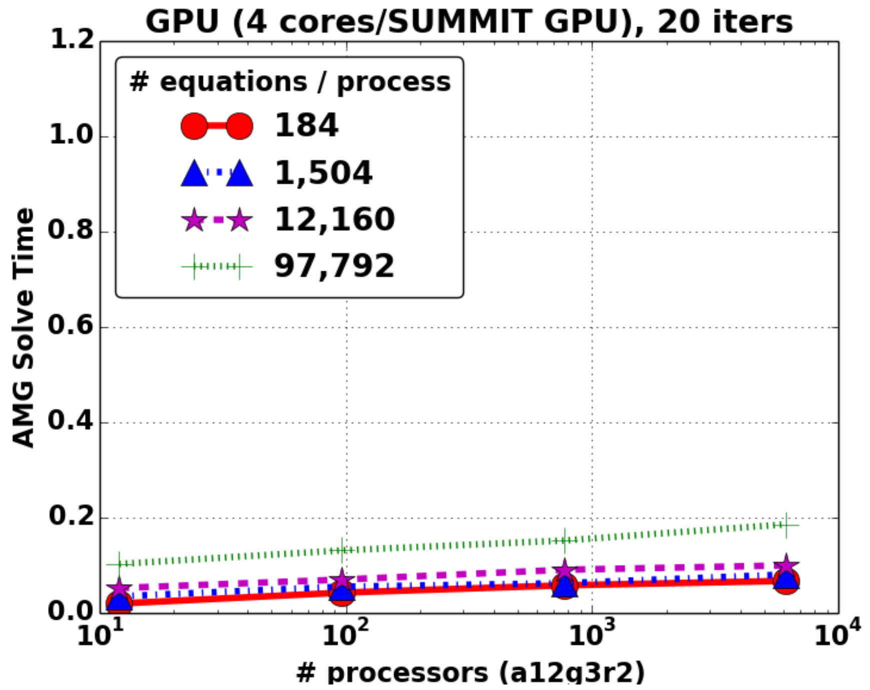
XGCm Gyrokinetic Poisson equation implementation on GPU

- PETSc is used for finite element linear system solve process;
- PETSc DM class, especially DMPlex class is used to manage the unstructured mesh:
  - problem discretization, finite element linear matrix assembly (on CPU);
  - interface with different PETSc solvers;
- Currently, linear matrix is assembled on CPU; GPU assembly is being developed by PETSc;
- The nonlinear solver interface, **SNES** class is used as interface for the linear equation solve. (complete linear solver could be accessed within SNES): -snes_type ksponly;
- The linear system solve is completely performed on GPU within PETSc framework, using CUDA:
  - matrix is specified as CUDA matrix: -(dm_)mat_type seqaijcusparse, mpiaijcusparse, or aijcusparse; with MatSetOptions();
  - vector is specified as CUDA vector: -(dm_)vec_type seqcuda, mpicuda, or cuda; with VecSetFromOptions();
  - with above matrix and vector type specification, matrix and vectors are also stored on GPU memory, matrix-vector operations are performed on GPU as possible.

PETSc performance comparison running with CPU and GPU

- Results shown here is based on running PETSc example case 13: petsc/src/snes/tutorial/ex13.c on GPU. – Results for the procedures we are developing are forth coming (We hope to have XGCm results soon)
- Results are from Dr. Mark Adams, running on 1 Summit node.
- **When problem size is small (~1,000 equations/process for this example), GPU time cost is higher than CPU; time cost comparison is problem dependent.**
- 4 processes/MPI ranks concurrently sharing a single GPU, using NVidia MPS (CUDA Multi-Process Service): BSUB -alloc_flags gpumps





Some details using GPU and running on Summit

- **CUDA-Aware MPI**
  - **Allows GPU buffers (e.g., memory allocated with cudaMalloc) to be used directly in MPI calls, rather than needing to manually transfer data to/from a CPU buffer (e.g., using cudaMemcpy) before/after passing data in MP calls.** [1]
  - **To enable CUDA-Aware MPI, use jsrun with flag, --smpiargs="-gpu" on Summit.**
- **Currently, not all preconditioner and linear iteration methods are completely performed on GPU within PETSc: (This may be another issues if what M3D-C1 needs is not on GPU)**
  - **For some preconditioner, part of the operations may be performed on CPU; data will need to be copied from/to GPU to/from CPU.**
  - **Similarly, some iterative methods may not be completely performed on GPU.**
  - **Details of each Preconditioner and linear solver, need to be consulted with PETSc developer.**
  - **-pc_type jacobi, -pc_type gamg is on GPU.**

1. https://www.olcf.ornl.gov/wp-content/uploads/2020/02/Summit_Tips_and_Tricks_2020-06-03.pdf