PPPL PRINCETON PLASMA PHYSICS LABORATORY

Final presentation of M3DC1 group

Team: M3DC1
Member: Jin Chen, Chang Liu, Chen Zhao (PPPL)
Mentors: Levi Barnes (NVIDIA), James Norris

GPU Hackathon Princeton
June 2021

- M3D-C1 is a finite element code solving the fluid equation in magnetically confined fusion plasma.
  - We use the Galerkin method to construct a sparse matrix to represent the PDE
- In Galerkin method, one needs to calculate an inner product for every matrix element.

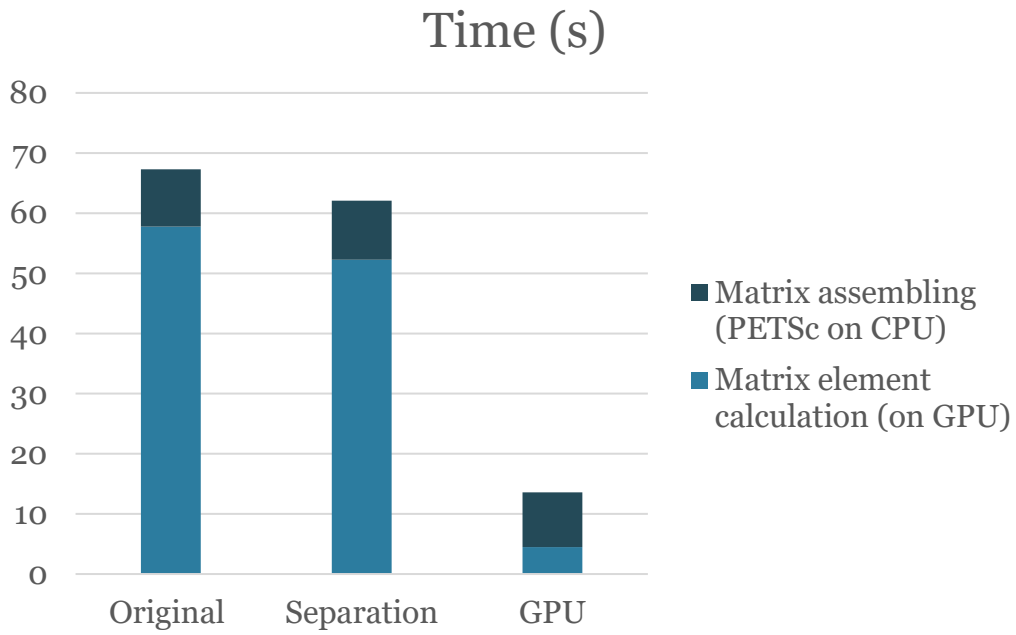
$$(\mu, Av) = \int_V \mu(x)A(x)v(x)J\,dx = \sum_i \mu_i A_i v_i J_i$$

  - We want to optimize the matrix element calculation and run it on GPUs.
- In current implementation, the physics part (calculating different terms in PDE) and the integral calculation are mixed together.

Separate the physics part and the integral calculation

- When trying to run both parts on GPU, we encounter a lot of OOM problems.
  - The physics part is very complicated and involved many (>20) large arrays
- However, most of the computation time is spent on the integral calculation.
  - This part can be written as nested for-loops and many operations are independent.
- We can try to separate the two parts and only do GPU optimization on the integral calculation.
  - However, it involves significant change to the code structure

Matrix element calculation (10 terms) gets 11x speedup on GPU

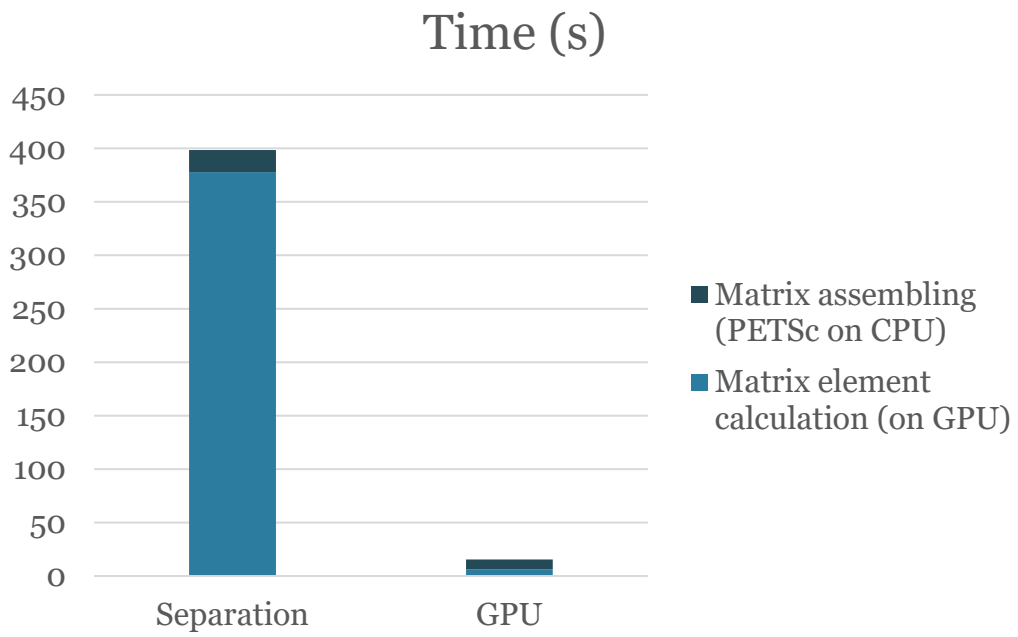


Calculation time for Galerkin matrix in a 3D mesh with 495424 elements using 4 Traverse nodes (128 MPI processes, 16 GPUs)

Nsight Compute result:
Achieved occupancy is 23.91%,
Theoretical occupancy is 25%

SOL SM: 27.16%
SOL Memory: 40.8%

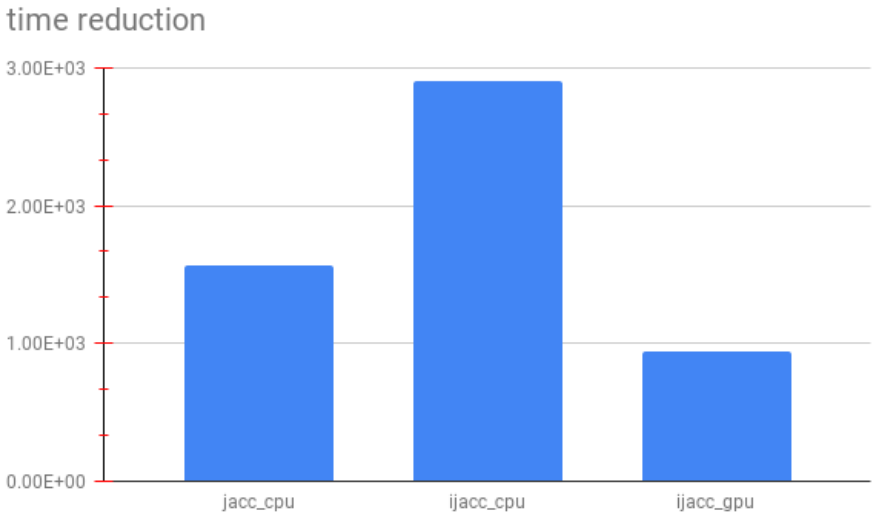
Matrix element calculation (70 terms) gets 61x speedup on GPU



Calculation time for Galerkin matrix in a 3D mesh with 495424 elements using 4 Traverse nodes (128 MPI processes, 16 GPUs)

Optimization of the original version

- We tried to use OpenACC and test the code on CPU/GPU, serially and parallelly.



- The ijacc code on gpu can beat the jacc code on cpu.

Problems encountered

- GPU memory (global, local, shared) is limited and one needs to be careful to avoid OOM error.
- We need to combine MPI and OpenACC, in which case many MPI processes need to share a single GPU on a node.
- We need to do a lot of communications between CPU and GPU before and after each kernel.
- We try to use `acc atomic` to deal with dependency and conflicts, but found that the current PGI compiler does not support `acc atomic` for double precession complex number (128bit).

What we learned

- With Multi-Process Service (MPS), OpenACC works well with OpenMPI.
- Optimize the OpenACC parallelization on nested loops
  - Use `acc loop collapse` for 2-layer and 3-layer nested loops
  - Try to use CUDA shared memory rather than local memory or registers
  - Change the order of nested loop and rearrange the data to reach better coalescing for memory reading and writing on GPU
- Use the `async` feature to do the CPU-GPU data copy and CPU calculation at the same time
- With these optimizations, we got significant speedup for the matrix element, but the matrix assembling is handled by PETSc on CPU, and it is taking the major part of computation time now.

Future plan

- Continue work on optimizing the original version of the code to solve OOM problem
- Try to implement more physical terms and see if we can get better performance gain.
- Reach the PETSc group to see if they can help optimize the matrix assembling part