AV-FLOW

Hadi Zolfaghari
Barna Becsek
Maria Nestola
Simone Riva

Mentors: Dave Norton, William Sawyer

AV-FLOW

- High-performance software for simulation of fluid-structure interactions in the aortic valve system (AV).
- High-order and MPI-parallel legacy solvers are coupled and integrated in MOOSE framework
- Scientific driver:

– Fluid-structure interaction in the aortic valve calls for high-fidelity solvers for turbulent blood flow and complex soft tissue dynamics. ***AV-FLOW library aims at increasing the fidelity of such simulations and it strives to use high-performance computing to apply the library in clinical decision making.***

Initial Profile

- We already had expensive operations of the FORTRAN flow solver ported to GPU using CUDA C.
- We adopted GPU-oriented numerical methods which are suited to benefit from GPU's shared-memory and data parallelism.
- We re-programmed and *locally* ported 90% of the double precision arithmetic operations of the flow solver.
- We achieved ~**10-30x** speedup vs. one CPU core for individual kernels, and ~**5x** speedup for the entire code.
- We did not quite recover the kernel speedups in the entire run, and we only could get speedup of **2x** vs. MPI version for very large grids (1B grid points/node!).

Evolution and Strategy

- What was your goal coming here?
  - Improving the GPU speedup for the entire run, so it can beat Haswell for lower grid points
  - Resolving the inefficiency problem of using CUDA Multi-process service
  - Using OpenACC or CUDA libraries upon discussion with mentors/team members.
- What was your initial strategy?
  - Use OpenACC and CUDA interoperability to minimize memory operations on the fly.
  - Hard-coding MPS for an efficient hybrid multicore/ manycore strategy.
- How did this strategy change?
  - We came up with re-factoring the CUDA code, so memory operations are pushed as further as posibble.
  - We realized that, for taking advantage of OpenACC and CUDA interoperability, we can not use GNU compilers! So we decided to use PGI compilers.
  - We started to use CUDA libraries in PETSc for accelerating algebraic operations of our structural solver.

Results and Final Profile

- What we were able to accomplish
  - After re-moduling the CUDA code, we seem to be able to reduce the memory operations. We still need to benchmark the code for CPU and GPU for large number of time-steps to report valid memory speedups.
  - We used extra kernel wrappers to compile and run the code using PGI compilers.
  - We ported (thanks to Dave) our original FORTRAN code to GPUs using pure OpenACC. It seems to be working two times faster than CPU!
  - We successfully used CUDA-enabled PETSc in our C++ structural solver!

What problems you encountered

- We suffered from scattered MPI operations in our flow solver code, which necessitate dynamic copies between CPU and GPU.
- We have device pointers for individual usage, refurbishing kernels is required for multiple use of one allocation.
- We had compiling errors while using CUDA-enabled PETSc (later resolved), and PETSc only encompasses cuSparse solver.

Wishlist

- What do you wish existed to make your life easier?
  - OpenACC support for GNU compilers.
  - cuBLAS and other CUDA libraries besides cuSparse in PETSc.
  - More meetings with PGI, NVIDIA and Cray experts.
  - GPU seminars in CSCS, to share experience and discuss issues.

Was it worth it?

- We did achieve improvements in our implementations and we acquired ideas for further GPU developement
- We benefited from discussions with eurohack17 mentors and participants and we want to thank CSCS for organizing such an inspiring event.