GPU accelerated OpenFOAM with petsc4Foam and performance comparison

NEXTfoam

http://www.nextfoam.co.kr

Bosung Lee

bslee@nextfoam.co.kr

Backgrounds and objectives

GPU performance has increased dramatically due to the development of AI, enabling faster and more efficient computation for various applications. One such application is CFD, which solves complex fluid dynamics problems using numerical methods. Many CFD solvers, such as Ansys Fluent, Siemens StarCCM+, and NASA FUN3D, have adopted GPU to speed up their calculations and reduce their computational costs.

OpenFOAM is another CFD solver that is working on GPU acceleration. OpenFOAM is an open-source software for computational fluid dynamics. A major challenge of GPU acceleration for OpenFOAM is to minimize the code modification needed to adapt to the GPU architecture. There are two possible ways to achieve this: accelerating the sparse iterative solver, which solves large systems of linear equations using GPU libraries, or accelerating the discretization and the iterative solver, which are the main steps of the numerical methods used in OpenFOAM.

This document aims to test the OpenFOAM GPU acceleration with the PETSc and AmgX libraries, which are two popular linear solvers for sparse systems that support GPU. The document will present the results of a simple test case that compares the speed, accuracy, and scalability of the GPU and CPU versions of OpenFOAM. The document will also discuss the current challenges and future directions of OpenFOAM GPU acceleration, such as code modification, compatibility, and performance optimization.

Approaches of OpenFOAM GPU Acceleration

GPU acceleration for OpenFOAM poses some challenges, such as code modification, memory transfer, and model compatibility. There are two main approaches to address these challenges: accelerating the sparse iterative solver or accelerating the discretization and the iterative solver.

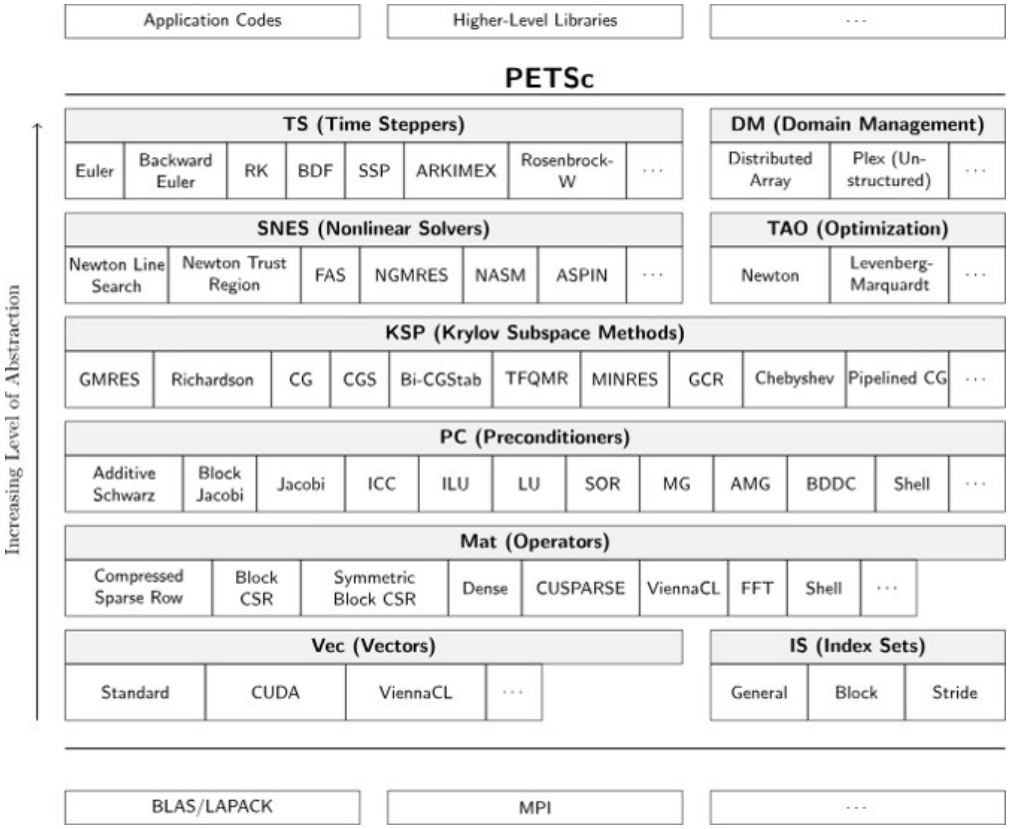
The first approach is to use existing libraries, such as PETSc, cuSPARSE, ofgpu, and AmgX, that can perform the sparse matrix operations on the GPU. The sparse iterative solver is the most time-consuming component of OpenFOAM solvers, as it solves large systems of linear equations that arise from the discretization of the governing equations. The advantage of this approach is that it requires less code modification, as the libraries can be easily integrated with the existing OpenFOAM code. The disadvantage is that it introduces additional overhead for the memory transfer between the CPU and the GPU, as the data needs to be moved back and forth between the host and the device. Another drawback is that it has limited capabilities for various models, such as turbulence, multiphase, or combustion, that may require additional solvers or algorithms.

The second approach is to rewrite or adapt the original OpenFOAM code to the GPU architecture, so that almost all the calculations can be performed on the GPU. This involves accelerating both the discretization

and the iterative solver, which are the main steps of the numerical methods used in OpenFOAM. The discretization is the process of converting the continuous governing equations into discrete algebraic equations using finite volume or finite element methods. The iterative solver is the process of finding the solution of the discrete equations using iterative methods, such as Gauss-Seidel or conjugate gradient. The advantage of this approach is that it eliminates the unnecessary overhead for the memory transfer between the CPU and the GPU, as the data stays on the device throughout the simulation. The disadvantage is that it requires more complicated and difficult code modification, as the OpenFOAM code needs to be modified for each version and model. Another drawback is that it is not updated frequently, as the existing implementation, RapidCFD, is based on an old version of OpenFOAM (v2.3.1) and has not been maintained since 2016.

PETSc (Portable Extensible Toolkit for Scientific Computation)[1]

PETSc is a software library that provides scalable parallel linear and nonlinear equation solvers, ODE integrators, and optimization algorithms for application codes written in C, C++, Fortran, and Python. PETSc can run on various platforms, including GPU systems, and supports various matrix and vector formats, such as CSR, SELL, SIF, and FFTW.



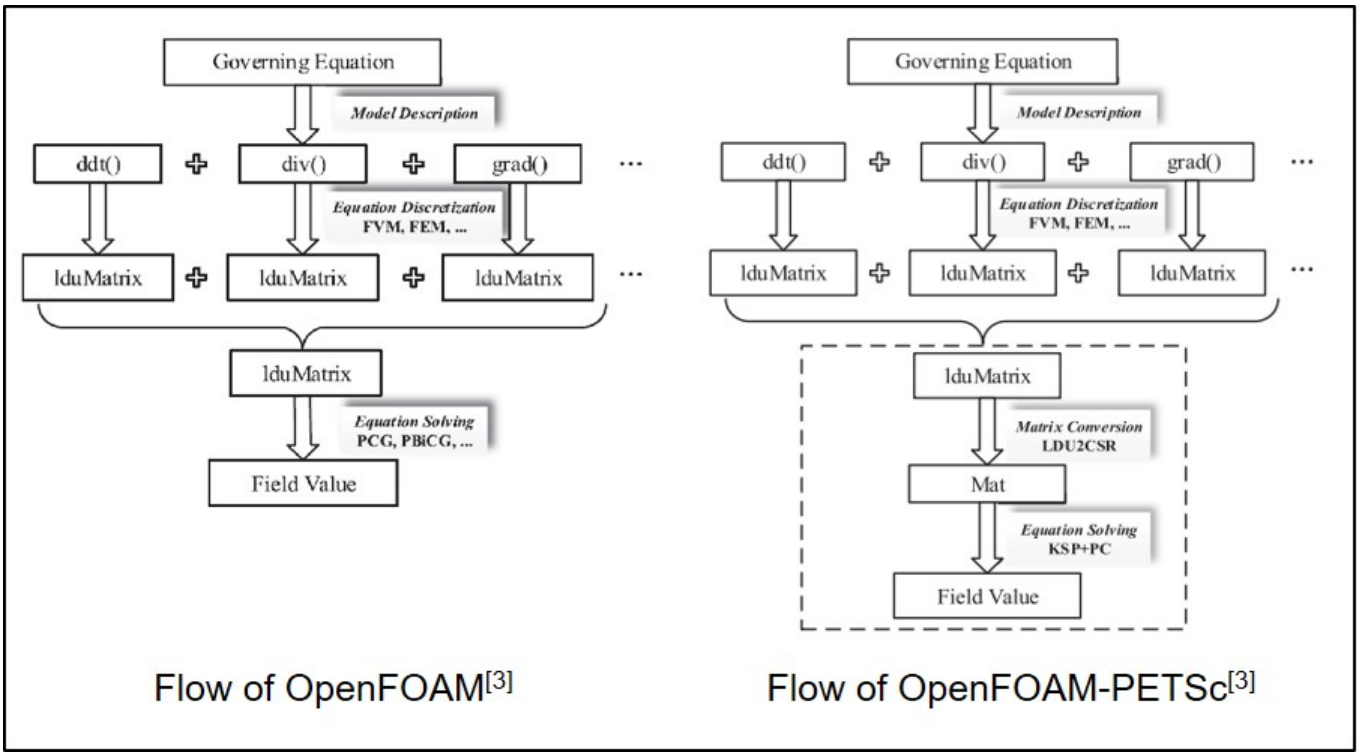
PETSc can also use external packages, such as PETSc, cuSPARSE, ofgpu, and AmgX, to perform the algebraic solvers on GPU systems using CUDA, HIP, OpenCL, or ViennaCL. PETSc is designed to be flexible, modular, and extensible, and allows users to customize their solvers and models using user callbacks, shell objects, and nested objects.

Programming Model	Supporting Package	`Vec` Status	`Mat` Status	Supported GPU types
CUDA	cuBLAS/cuSparse	SUPPORTED	SUPPORTED	NVIDIA GPUs
HIP	Rocm	SUPPORTED	IN DEVELOPMENT	AMD GPUs
SYCL	MKL	NOT YET SUPPORTED	NOT YET SUPPORTED	NOT YET SUPPORTED
OpenCL	ViennaCL	SUPPORTED	SUPPORTED	NVIDIA, AMD, INTEL GPUs
Kokkos	Kokkos and Kokkos-Kernels	SUPPORTED	SUPPORTED	NVIDIA, AMD, INTEL GPUs

petsc4Foam[2]

petsc4Foam is a library that plugs in PETSc and its external dependencies, such as Hypre, into arbitrary OpenFOAM simulations. By using petsc4Foam, OpenFOAM users can benefit from the performance and flexibility of PETSc and its external libraries.

The petsc4Foam library provides the following features.[3]



- ldu2csr matrix conversion: This is a process of converting the matrix format used by OpenFOAM (ldu) to the matrix format used by PETSc (csr). ldu stands for lower-diagonal-upper, which is a compressed row storage format that stores only the lower, diagonal, and upper elements of a sparse matrix. csr stands for compressed sparse row, which is another compressed row storage format that stores the non-zero elements of a sparse matrix and their column indices. The conversion is necessary to use the PETSc solvers with the OpenFOAM matrices.
- selection of solvers and preconditioners available in PETSc and in its external libraries: This is a feature that allows the users to choose from a variety of solvers and preconditioners that are implemented in PETSc and its external libraries, such as Hypre. Solvers are algorithms that find the solution of a system of linear or nonlinear equations, such as Gauss-Seidel or conjugate gradient. Preconditioners are

techniques that improve the convergence and stability of the solvers, such as Jacobi or incomplete LU. The selection can be done by specifying the solver and preconditioner names in the OpenFOAM control dictionary.

- run-time options by a dictionary and/or a rc file: This is a feature that allows the users to modify the PETSc options at run-time, without recompiling the code. The options can be set by a dictionary file, which is a text file that contains key-value pairs, or by a rc file, which is a text file that contains PETSc command-line arguments. The options can control various aspects of the PETSc solvers, such as the tolerance, the maximum number of iterations, the verbosity, etc
- basic caching implementation of a PETSc matrix: This is a feature that improves the efficiency of the petsc4Foam library by caching the PETSc matrix object. Caching is a technique that stores the frequently used data in a fast-access memory, such as RAM, to reduce the access time. The PETSc matrix object is cached to avoid the repeated creation and destruction of the object, which can be costly in terms of time and memory.

To integrate PETSc with OpenFOAM in Ubuntu Linux with NVidia GPU, the following steps are needed.

- Install proper nVidia video driver. nvidia-driver-535 is used.

```
$ sudo apt-get update
$ apt install nvidia-driver-535
```

- Install CUDA 11.7 for nvidia-driver-535

```
$ sudo apt-get install wget gpg
$ wget
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/c
uda-ubuntu2004.pin
$ sudo mv cuda-ubuntu2004.pin /etc/apt/preferences.d/cuda-repository-pin-600
$ wget
https://developer.download.nvidia.com/compute/cuda/11.7.0/local_installers/c
uda-repo-ubuntu2004-11-7-local_11.7.0-515.43.04-1_amd64.deb
$ sudo dpkg -i cuda-repo-ubuntu2004-11-7-local_11.7.0-515.43.04-1_amd64.deb
$ sudo cp /var/cuda-repo-ubuntu2004-11-7-local/cuda-*-keyring.gpg
/usr/share/keyrings/
$ sudo apt-get -y update
$ sudo apt-get install -y cuda-11-7
$ sudo echo 'export PATH=$PATH:/usr/local/cuda/bin' >> /etc/bash.bashrc
$ sudo echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64'
>> /etc/bash.bashrc
```

- Build OpenMPI 4.0.5 with CUDA support for multiple GPUs.

```
$ sudo apt-get install -y git build-essential m4 flex zlib1g-dev
$ sudo mkdir -p /opt/openmpi-4.0.5
$ wget https://download.open-mpi.org/release/open-mpi/v4.0/openmpi-
```

```
4.0.5.tar.gz
$ tar zxvf openmpi-4.0.5.tar.gz
$ rm -f *.tar.gz
$ cd openmpi-4.0.5
$ ./configure --prefix=/opt/openmpi-4.0.5 --with-cuda=/usr/local/cuda
$ make -j 8 all
$ sudo make install
$ sudo echo 'export PATH=$PATH:/opt/openmpi-4.0.5/bin' >> /etc/bash.bashrc
```

- Install OpenFOAM-v2306

```
$ sudo mkdir -p /opt/OpenFOAM
$ cd /opt/OpenFOAM
$ sudo wget https://dl.openfoam.com/source/v2306/OpenFOAM-v2306.tgz
$ sudo wget https://dl.openfoam.com/source/v2306/ThirdParty-v2306.tgz
$ sudo tar zxf OpenFOAM-v2306.tgz
$ sudo tar zxf ThirdParty-v2306.tgz
$ sudo rm *.tgz
$ sudo sed -i 's/projectDir="$HOME/projectDir="\/opt/'
/opt/OpenFOAM/OpenFOAM-v2306/etc/bashrc
$ sudo sed -i 's/WM_PROJECT_USER_DIR="$HOME/WM_PROJECT_USER_DIR="\/opt/'
/opt/OpenFOAM/OpenFOAM-v2306/etc/bashrc
$ sudo sed -i 's/${USER:-user}/nextfoam/' /opt/OpenFOAM/OpenFOAM-
v2306/etc/bashrc
$ source /opt/OpenFOAM/OpenFOAM-v2306/etc/bashrc
$ export WM_NCOMPPROCS=8
$ cd /opt/OpenFOAM/OpenFOAM-v2306
$ sudo ./Allwmake
$ sudo echo 'source /opt/OpenFOAM/OpenFOAM-v2306/etc/bashrc' >>
/etc/bash.bashrc
```

- Build PETSc

```
$ git clone -b release https://gitlab.com/petsc/petsc.git
$ cd petsc
$ sudo mkdir -p /opt/petsc
$ ./configure --with-64-bit-indices=0 --with-precision=double --
prefix=/opt/petsc --with-cuda=1 --gpu-architecture=sm_80 --with-openmpi=1 --
with-openmpi-dir=/opt/openmpi-4.0.5/ --download-f2cblaslapack --with-fc=0 --
force
$ make PETSC_DIR=petsc PETSC_ARCH=arch-linux-c-debug -j 8 all
$ sudo make PETSC_DIR=petsc PETSC_ARCH=arch-linux-c-debug install
$ sudo echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/petsc/lib' >>
/etc/bash.bashrc
```

- Build petsc4foam

```
```

```
$ git clone https://develop.openfoam.com/modules/external-solver
$ cd external-solver
$ export PETSC_ARCH_PATH=/opt/petsc/
$ ./Allwclean
$ sudo ./Allwmake
```

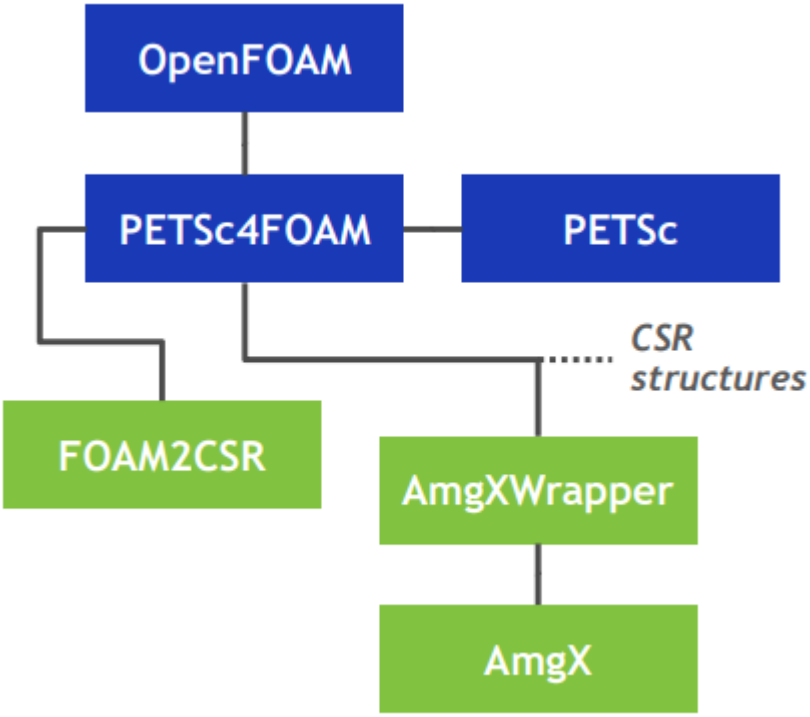
AmgXWrapper with PETSc

AmgX is a GPU-accelerated core solver library that speeds up the linear solver part of simulation. It supports algebraic multigrid (AMG) preconditioners and Krylov solvers, such as CG, BiCGStab, and GMRES. It can handle various types of matrices, such as symmetric, asymmetric, positive definite, or indefinite. It can also work with multiple GPUs and support MPI communication. It is designed to be fast, robust, and easy to use, and allows users to control the solver parameters using configuration files or command-line options.

AmgXWrapper is an interface that connects PETSc and AmgX. PETSc is a software library that provides scalable parallel linear and nonlinear equation solvers, ODE integrators, and optimization algorithms for application codes written in C, C++, Fortran, and Python. AmgXWrapper simplifies the usage of AmgX with PETSc, and helps developers use AmgX in their PETSc applications without a thorough understanding of the AmgX API. With AmgXWrapper, developers may need only a few lines of code modification to add AmgX solvers in legacy PETSc applications. AmgXWrapper also features implicit data transfer when there are mismatched numbers of CPU cores and GPU devices in a computing node, which allows exploiting all possible resources on modern heterogeneous platforms.

AmgXWrapper is also an extension of petsc4Foam, which is a library that plugs in PETSc and its external dependencies, such as Hypre, into arbitrary OpenFOAM simulations. OpenFOAM is an open-source software for computational fluid dynamics (CFD) that can solve complex fluid problems using numerical methods. However, CFD simulations can be very time-consuming and computationally intensive, especially for large-scale or high-resolution cases. Therefore, PETSc and AmgX can be good options for adding GPU computing to OpenFOAM applications. By using petsc4Foam and AmgXWrapper, OpenFOAM users can benefit from the performance and flexibility of PETSc and AmgX and their external libraries.

High level call structure for initial acceleration approach using AmgX[4] is shown below.



To integrate AmgXWrapper with OpenFOAM in Ubuntu Linux with NVidia GPU, the following steps are needed.

- Install proper nVidia video driver. nvidia-driver-535 is used.

```
$ sudo apt-get update
$ apt install nvidia-driver-535
```

- Install CUDA 11.7 for nvidia-driver-535

```
$ sudo apt-get install wget gpg
$ wget
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/c
uda-ubuntu2004.pin
$ sudo mv cuda-ubuntu2004.pin /etc/apt/preferences.d/cuda-repository-pin-600
$ wget
https://developer.download.nvidia.com/compute/cuda/11.7.0/local_installers/c
uda-repo-ubuntu2004-11-7-local_11.7.0-515.43.04-1_amd64.deb
$ sudo dpkg -i cuda-repo-ubuntu2004-11-7-local_11.7.0-515.43.04-1_amd64.deb
$ sudo cp /var/cuda-repo-ubuntu2004-11-7-local/cuda-*-keyring.gpg
/usr/share/keyrings/
$ sudo apt-get -y update
$ sudo apt-get install -y cuda-11-7
$ sudo echo 'export PATH=$PATH:/usr/local/cuda/bin' >> /etc/bash.bashrc
$ sudo echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64'
>> /etc/bash.bashrc
```

- Build OpenMPI 4.0.5 with CUDA support for multiple GPUs.

```
$ sudo apt-get install -y git build-essential m4 flex zlib1g-dev
$ sudo mkdir -p /opt/openmpi-4.0.5
$ wget https://download.open-mpi.org/release/open-mpi/v4.0/openmpi-
4.0.5.tar.gz
$ tar zxvf openmpi-4.0.5.tar.gz
$ rm -f *.tar.gz
$ cd openmpi-4.0.5
$ ./configure --prefix=/opt/openmpi-4.0.5 --with-cuda=/usr/local/cuda
$ make -j 8 all
$ sudo make install
$ sudo echo 'export PATH=$PATH:/opt/openmpi-4.0.5/bin' >> /etc/bash.bashrc
```

- Install OpenFOAM-v2306

```
$ sudo mkdir -p /opt/OpenFOAM
$ cd /opt/OpenFOAM
$ sudo wget https://dl.openfoam.com/source/v2306/OpenFOAM-v2306.tgz
$ sudo wget https://dl.openfoam.com/source/v2306/ThirdParty-v2306.tgz
$ sudo tar zxf OpenFOAM-v2306.tgz
$ sudo tar zxf ThirdParty-v2306.tgz
$ sudo rm *.tgz
$ sudo sed -i 's/projectDir="$HOME/projectDir="\/opt/'
/opt/OpenFOAM/OpenFOAM-v2306/etc/bashrc
$ sudo sed -i 's/WM_PROJECT_USER_DIR="$HOME/WM_PROJECT_USER_DIR="\/opt/'
/opt/OpenFOAM/OpenFOAM-v2306/etc/bashrc
$ sudo sed -i 's/${USER:-user}/nextfoam/' /opt/OpenFOAM/OpenFOAM-
v2306/etc/bashrc
$ source /opt/OpenFOAM/OpenFOAM-v2306/etc/bashrc
$ export WM_NCOMPPROCS=8
$ cd /opt/OpenFOAM/OpenFOAM-v2306
$ sudo ./Allwmake
$ sudo echo 'source /opt/OpenFOAM/OpenFOAM-v2306/etc/bashrc' >>
/etc/bash.bashrc
```

- Install cmake 3.27 for building AmgX

```
$ sudo mkdir -p /opt/cmake
$ wget https://github.com/Kitware/CMake/releases/download/v3.27.2/cmake-
3.27.2-linux-x86_64.sh
$ sudo sh cmake-3.27.2-linux-x86_64.sh --skip-license --prefix=/opt/cmake
$ sudo echo 'export PATH=$PATH:/opt/cmake/bin' >> /etc/bash.bashrc
```

- Install AmgX

```
$ git clone --recursive https://github.com/nvidia/AmgX.git
$ mkdir -p AmgX/build
$ cd AmgX/build
$ sudo mkdir -p /opt/AmgX
$ cmake  -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/opt/AmgX -
DCUDA_ARCH="60;70;80;86" ..
$ make -j 8 all
$ sudo make install
$ sudo echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/AmgX/lib' >>
/etc/bash.bashrc
```

- Build PETSc

```
$ git clone -b release https://gitlab.com/petsc/petsc.git
$ cd petsc
$ sudo mkdir -p /opt/petsc
$ ./configure --with-64-bit-indices=0 --with-precision=double --
prefix=/opt/petsc --with-cuda=1 --gpu-architecture=sm_80 --with-openmpi=1 --
with-openmpi-dir=/opt/openmpi-4.0.5/ --download-f2cblaslapack --with-fc=0 --
force
$ make PETSC_DIR=petsc PETSC_ARCH=arch-linux-c-debug -j 8 all
$ sudo make PETSC_DIR=petsc PETSC_ARCH=arch-linux-c-debug install
$ sudo echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/petsc/lib' >>
/etc/bash.bashrc
```

- Build FOAM2CSR

```
$ export CUBROOT=./AmgX/thrust/
$ export PETSC_INC=/opt/petsc/include
$ export AmgX_INC=/opt/AmgX/include
$ export SPECTRUM_MPI_HOME=/opt/openmpi-4.0.5
$ export PETSC_LIB_DIR=/opt/petsc/lib
$ export AmgX_LIB=/opt/AmgX/lib

$ git clone https://gitlab.hpc.cineca.it/openfoam/foam2csr.git
$ cd foam2csr

# Edit NVARCH in the Make/nvcc to your NVARCH
$ sudo ./Allwmake
```

- Build petsc4foam

```
$ git clone --branch AmgXwrapper
https://develop.openfoam.com/modules/external-solver.git petsc4foam
$ export CUBROOT=./AmgX/thrust/
$ export PETSC_INC=/opt/petsc/include
```

```
$ export AmgX_INC=/opt/AmgX/include
$ export SPECTRUM_MPI_HOME=/opt/openmpi-4.0.5
$ export PETSC_LIB_DIR=/opt/petsc/lib
$ export AmgX_LIB=/opt/AmgX/lib
$ export PETSC_ARCH_PATH=/opt/petsc/
$ export FOAM2CSR_INC=./foam2csr/src/
$ export CUDA_INC=/usr/local/cuda/include

$ cd petsc4foam
$ sed -i 's/-I$(FOAM2CSR_INC)/-I$(FOAM2CSR_INC) -I$(CUDA_INC)/'
src/petsc4Foam/Make/options
$ ./Allwclean
$ sudo ./Allwmake
```

Performance comparison

In this study, we investigate the performance of GPU acceleration for solving the 3D lid-driven cavity flow problem using the icoFoam solver in OpenFOAM. The test environment consists of Ubuntu 20.04, OpenFOAM-v2306, OpenMPI-4.0.5 with CUDA, PETSc-3.19.5 with sm_80 architecture, CUDA-11.7, and AmgX 2.3.0.

Test conditions

- Hardwares

  For the small-sized problem, the following test hardware: an AMD Ryzen9 6900HX processor (3.3 GHz ~ 4.9 GHz, 8 cores) and an NVIDIA GeForce RTX3060 Laptop GPU (6 GB) is used.

  For midium-sized case, AWS g5.12xlarge instance which has AMD EPYC 7R32 (2.8 GHz ~ 3.3 GHz) 24 cores and four NVIDIA A10G 24GB is used.

- Problem conditions

  The mesh size of small-sized proglem is 1M (100x100x100) and 10 time steps from 0 to 0.01 with dt = 0.001 are used. maxITER for P is set as 250. The medium size mesh is 3.375M (150x150x150) and 8M (200x200x200) are used. Total 10 time steps from 0 to 0.001 with dt = 0.001 is used. maxITER for P is also set as 250.

- Solver and Preconditioner for PETSc case, the `fvSolution`[5] is set as following:

```
solvers
{
    p
    {
        solver petsc;
        petsc
        {
            options
            {
                ksp_type cg;
```

```
                mat_type aijcusparse;
                pc_type gamg;
            }
        }
        tolerance       0;
        relTol          0;
        maxIter         250;
    }

    U
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       0;
        relTol          0;
        maxIter         5;
    }
}
```

- Solver and Preconditioner For AmgX case, PCG solver and BLOCK_JACOBI preconditioner are used. the `fvSolution` is set as following:

```
solvers
{
    p
    {
        solver AmgX;
        AmgX {};
        tolerance       0;
        relTol          0;
        maxIter         250;

    }

    U
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       0;
        relTol          0;
        maxIter         5;
    }
}
```

And, `AmgXpOptions`[6] file needs to be created.

```
{
    "config_version": 2,
    "solver": {
```

```
        "preconditioner": {
            "scope": "precond",
            "solver": "BLOCK_JACOBI"
        },
        "solver": "PCG",
        "print_solve_stats": 1,
        "obtain_timings": 1,
        "max_iters": 200,
        "monitor_residual": 1,
        "store_res_history": 1,
        "scope": "main",
        "convergence": "RELATIVE_INI_CORE",
        "tolerance": 1e-03
    }
}
```

In the `controlDict` file for both cases, `libs (petscFoam);` is added as below.

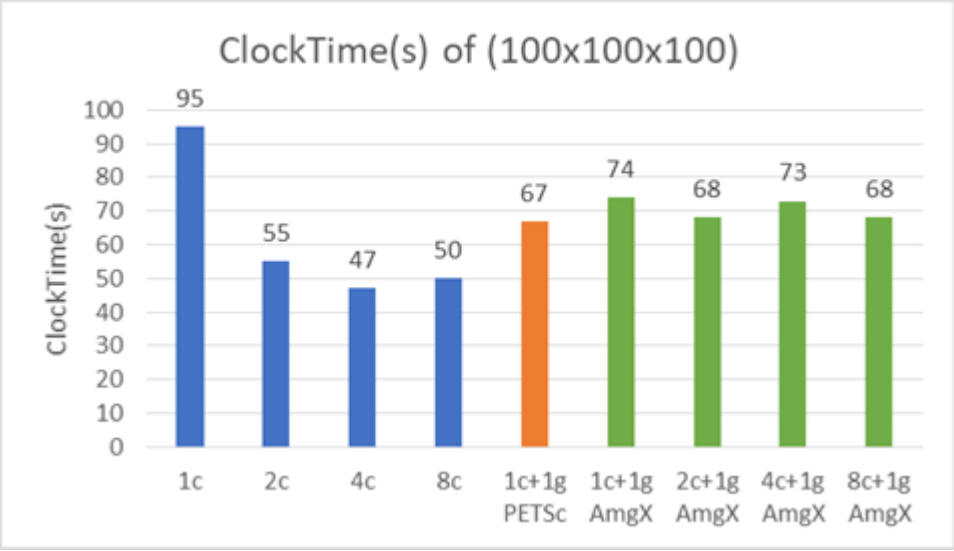
```
libs            (petscFoam);
application     icoFoam;
startFrom       startTime;
```

In the CPU only case, PCG solver and DIC preconditioner are used.

Test results

- 1M (100x100x100) case

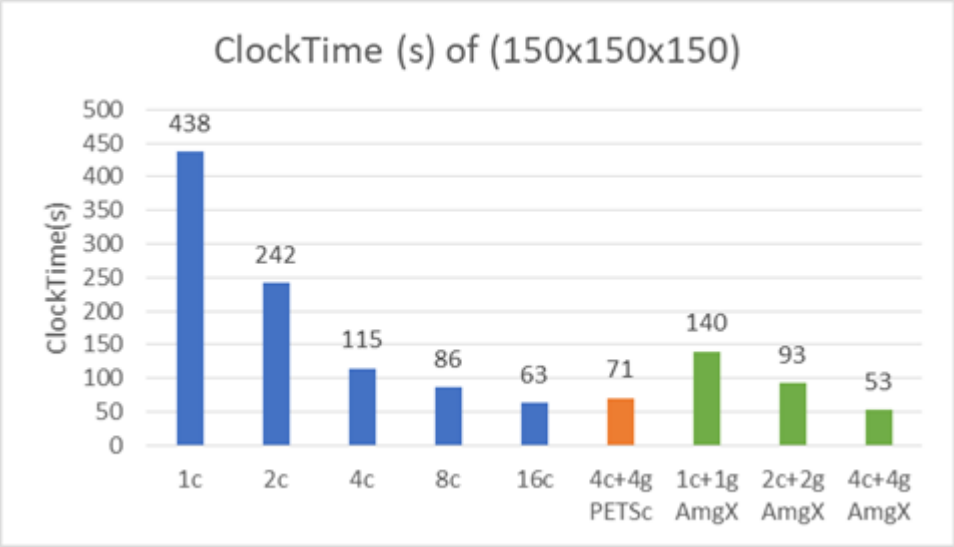
  In CPU only case, 1,2,4,8 CPUs are used. In PETSc, 1 CPU and only 1 GPU are used because about 3.2 GB of GPU memory (58 % of tatal GPU memory) is used. Otherwise, AmgX case, 1,2,4,8 CPUs are used with 1 GPU.

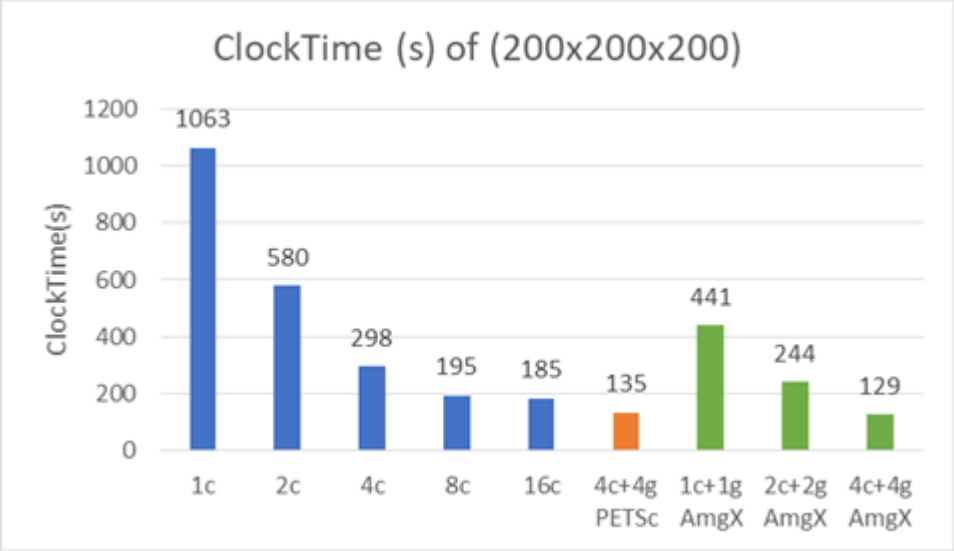


  As shown in the figure, PETSc with 1 GPU and 1 CPU is about 1.4 times faster than 1 CPU. But, AmgX cases are not faster than PETSc case.

- 3.375M (150x150x150) and 8M(200x200x200) cases

  In CPU only cases, 1, 2, 4, 8, 16 CPUs are used. PETSc with 4 CPU with 4 GPU is compared. 1 CPU with 1 GPU, 2 CPUs with 2 GPUs cases diverged and 8M case only completed with 4 CPUs and 4 GPUs. AmgX is run on 1, 2, 4 CPUs with each 1, 2, 4 GPUs.





  In both size problems, GPU cases show faster than CPUs.

Concluding Remarks

This document reports the results of testing the OpenFOAM GPU acceleration with the PETSc library and AmgX, which are external solvers that can run on GPU systems.

By using PETSc and AmgX, OpenFOAM GPU acceleration can be achieved with less code modification. However, the GPU acceleration does not show much speedup because of the additional overhead. One reason is the conversion between the LDU matrix and the CSR matrix, which are the matrix formats used by OpenFOAM and the external solvers, respectively. Another reason is the difference in the convergence rate between the sparse solvers in PETSc, AmgX, and the standard solver in OpenFOAM, which may affect the performance and the stability of the GPU acceleration.

A limitation of the GPU acceleration is the GPU memory size, which restricts the size and the complexity of the problems that can be solved on the GPU. AmgXWrapper with PETSc is more memory optimized than PETSc GPU, as it features implicit data transfer when there are mismatched numbers of CPU cores and GPU devices in a computing node.

The AMD GPU with ROCm support will be tested for the OpenFOAM GPU acceleration. The performance optimization for fitting the problem size and the GPU memory size will also be checked. Moreover, other applications, such as simpleFoam, will be used to validate the benefit of the GPU acceleration. Furthermore, the standard CPU solvers and the GPU solvers will be compared in terms of the computational cost.

References

[1] PETSc(Portable Extensible Toolkit for Scientific Computation), https://petsc.org/release/

[2] petsc4Foam, https://develop.openfoam.com/modules/external-solver

[3] HAO LI et el., Insertion of PETSc in the OpenFOAM Framework, August 2017, ACM Transactions on Modeling and Performance Evaluation of Computing Systems 2(3):1-19, DOI:10.1145/3098821

[4] Matt Martineau et el., AMGX GPU SOLVER DEVELOPMENTS FOR OPENFOAM, 2020, https://wiki.openfoam.com/images/a/a4/OpenFOAM_2020_NVIDIA_Martineau.pdf

[5] S. Zampini et el., GPU accelerated OpenFOAM simulations with PETSc4FOAM, 2020, 8th OpenFOAM Conference 2020, https://wiki.openfoam.com/images/c/cd/OpenFOAM_2020_KAUST_Zampini.pdf

[6] Missing header file for amgxwrapper, https://develop.openfoam.com/modules/external-solver/-/issues/25