Summary Version

Results and benchmark tarball are for version 1.0 from the Laghos repo (tag v1.0).

Purpose of Benchmark

Laghos (LAGrangian High-Order Solver) is a miniapp that solves the time-dependent Euler equations of compressible gas dynamics in a moving Lagrangian frame using unstructured high-order finite element spatial discretization and explicit high-order time-stepping.

Laghos is based on the discretization method described in the following article:

> V. Dobrev, Tz. Kolev and R. Rieben
> High-order curvilinear finite element methods for Lagrangian hydrodynamics
> *SIAM Journal on Scientific Computing*, (34) 2012, pp. B606-B641

Laghos captures the basic structure of many other compressible shock hydrocodes, including the BLAST code at Lawrence Livermore National Laboratory. The miniapp is built on top of a general discretization library, MFEM, thus separating the pointwise physics from finite element and meshing concerns.

Laghos is a LLNL ASC co-design mini-app that was developed as part of the CEED software suite, a collection of software benchmarks, miniapps, libraries and APIs for efficient exascale discretizations based on high-order finite element and spectral element methods. See http://github.com/ceed for more information and source code availability.

The CEED research is supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nation's exascale computing imperative.

Characteristics of Benchmark

In each time step, the problem is ultimately formulated as solving a big system of ordinary differential equations (ODE) for the unknown (high-order) velocity, internal energy and mesh nodes (position). The left-hand side of this ODE is controlled by *mass matrices* (one for velocity and one for energy), while the right-hand side is constructed from a *force matrix*. Laghos supports two options for deriving and solving the ODE system, namely the *full assembly* and the *partial assembly* methods. **Partial assembly is the main algorithm of interest for this benchmark.**

Other computational motives in Laghos include the following:

- Support for unstructured meshes, in 2D and 3D, with quadrilateral and hexahedral elements (triangular and tetrahedral elements can also be used, but with the less efficient full assembly option). Serial and parallel mesh refinement options can be set via a command-line flag.
- Explicit time-stepping loop with a variety of time integrator options. Laghos supports Runge-Kutta ODE solvers of orders 1, 2, 3, 4 and 6.
- Continuous and discontinuous high-order finite element discretization spaces of runtime-specified order.
- Moving (high-order) meshes.
- Separation between the assembly and the quadrature point-based computations.
- Point-wise definition of mesh size, time-step estimate and artificial viscosity coefficient.
- Constant-in-time velocity mass operator that is inverted iteratively on each time step. This is an example of an operator that is prepared once (fully or partially assembled), but is applied many times. The application cost is dominant for this operator.
- Time-dependent force matrix that is prepared every time step (fully or partially assembled) and is applied just twice per "assembly". Both the preparation and the application costs are important for this operator.
- Domain-decomposed MPI parallelism.
- Optional in-situ visualization with GLVis and data output for visualization / data analysis with VisIt.

Code Structure

- The file `laghos.cpp` contains the main driver with the time integration loop starting around line 310.
- In each time step, the ODE system of interest is constructed and solved by the class `LagrangianHydroOperator`, defined around line 258 of `laghos.cpp` and implemented in files `laghos_solver.hpp` and `laghos_solver.cpp`.
- All quadrature-based computations are performed in the function `LagrangianHydroOperator::UpdateQuadratureData` in `laghos_solver.cpp`.
- Depending on the chosen option (`-pa` for partial assembly or `-fa` for full assembly), the function `LagrangianHydroOperator::Mult` uses the corresponding method to construct and solve the final ODE system.

- The full assembly computations for all mass matrices are performed by the MFEM library, e.g., classes `MassIntegrator` and `VectorMassIntegrator`. Full assembly of the ODE's right-hand side is performed by utilizing the class `ForceIntegrator` defined in `laghos_assembly.hpp`.
- The partial assembly computations are performed by the classes `ForcePAOperator` and `MassPAOperator` defined in `laghos_assembly.hpp`.
- When partial assembly is used, the main computational kernels are the `Mult*` functions of the classes `MassPAOperator` and `ForcePAOperator` implemented in file `laghos_assembly.cpp`. These functions have specific versions for quadrilateral and hexahedral elements.
- The orders of the velocity and position (continuous kinematic space) and the internal energy (discontinuous thermodynamic space) are given by the `-ov` and `-ot` input parameters, respectively.

Mechanics of Building the Benchmark

Laghos has the following external dependencies:

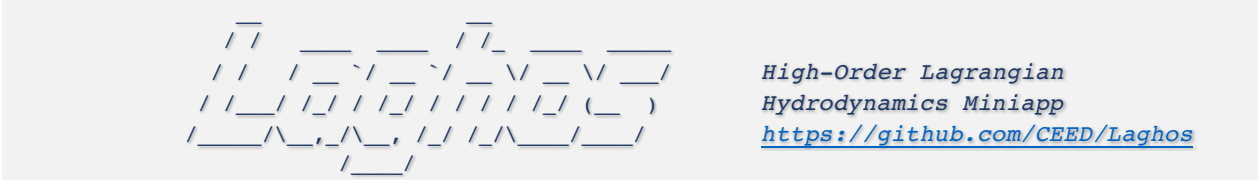
- *hypre*, used for parallel linear algebra, we recommend version 2.10.0b https://computation.llnl.gov/casc/hypre/software.html
- METIS, used for parallel domain decomposition (optional), we recommend version 4.0.3 http://glaros.dtc.umn.edu/gkhome/metis/metis/download
- MFEM, used for (high-order) finite element discretization, its GitHub master branch https://github.com/mfem/mfem

To build the miniapp, first download *hypre* and METIS from the links above and put everything on the same level as Laghos:

```
~> ls
Laghos/ hypre-2.10.0b.tar.gz metis-4.0.tar.gz
```

Build *hypre* (note the `--enable-bigint` option)

```
~> tar -zxvf hypre-2.10.0b.tar.gz
~> cd hypre-2.10.0b/src/
~/hypre-2.10.0b/src>./configure --disable-fortran --enable-bigint
~/hypre-2.10.0b/src> make -j
~/hypre-2.10.0b/src> cd ../..
```



Build METIS:

```
~> tar -zxvf metis-4.0.3.tar.gz
~> cd metis-4.0.3
~/metis-4.0.3> make
~/metis-4.0.3> cd ..
~> ln -s metis-4.0.3 metis-4.0
```

This build is optional, as MFEM can be built without METIS by specifying `MFEM_USE_METIS = NO` below.

Clone and build the parallel version of MFEM starting from the laghos-v1.0 tag:

```
~> git clone git@github.com:mfem/mfem.git ./mfem
~> cd mfem/
~/mfem> git checkout laghos-v1.0
~/mfem> make parallel -j
~/mfem> cd ..
```

See the MFEM building page for additional details.

Build Laghos:

```
~> cd Laghos/
~> make
```

This can be followed by `make test` and `make install` to check and install the build respectively. See `make help` for additional options.

Mechanics of Running the Benchmark

Sedov Blast

The main problem of interest for Laghos is the Sedov blast wave (`-p 1`) with partial assembly option (`-pa`). A sample runs in 3D is:

```
mpirun -np 8 laghos -p 1 -m data/cube01_hex.mesh -rs 2 -tf 0.6
                    -no-vis -pa
```

To partition an initial 3D mesh in a way that results in a perfectly balanced partitioning, with each MPI task having the same number of zones, one needs to specify the correct partitioning (`-pt`) and initial mesh (`-m`) options. The (`-pt`) option specifies the relative ratio between the number of MPI tasks in each of the (x,y,z) directions, as shown in the examples below. Furthermore, the

number of serial refinements (option `-rs`) should be sufficiently high to produce at least one zone per MPI task, before the parallel refinements (option `-rp`) are performed.

Having an initial mesh with 8 zones and 4096 MPI tasks, one should use the -pt 111 -m data/cube01_hex.mesh options:

```
mpirun -np 4096 laghos -p 1 -pt 111 -m data/cube01_hex.mesh
                       -rs 3 -rp 2 -tf 0.6 -no-vis –pa
```

this ensures that after 3 uniform serial refinements, the mesh is 16 x 16 x 16, i.e. there is exactly one element for each of the 4096 MPI tasks (these elements are additionally refined two more times, `-rp 2`, in parallel). If one wants to use 2048 MPI tasks, the command line would be:

```
mpirun -np 2048 laghos -p 1 -pt 221 -m data/cube01_hex.mesh
                       -rs 3 -rp 2 -tf 0.6 -no-vis –pa
```

leading to the same 16 x 16 x 16 serial mesh, but partitioned in 16 x 16 x 8 MPI tasks.

Similarly, for 1536 MPI tasks, one can specify an initial 4 x 3 x 2 mesh with 24 zones, using the following options:

```
mpirun -np 1536 laghos -p 1 -pt 432 -m data/cube_24_hex.mesh
                       -rs 2 -rp 2 -tf 0.6 -no-vis -pa
```

To run on 6144 MPI tasks, an appropriate initial mesh and serial/parallel refinements are specified with:

```
mpirun -np 6144 laghos -p 1 -pt 322 -m data/cube_12_hex.mesh
                       -rs 3 -rp 2 -tf 0.6 -no-vis -pa
```

Verification of Results

To make sure the results are correct, we tabulate reference final iterations (`step`), time steps (`dt`) and energies (`|e|`) for the following runs:

```
mpirun -np 8 laghos -p 1 -m data/square01_quad.mesh -rs 3 -tf 0.8
                    -no-vis –pa

mpirun -np 8 laghos -p 1 -m data/cube01_hex.mesh -rs 2 -tf 0.6
                    -no-vis –pa
```

run	step	dt	e
1	1150	0.002271	46.3055694447
2	561	0.000360	134.0937837800

An implementation is considered valid if the final energy values are all within round-off distance from the above reference values.

Performance Timing and FOM

Each time step in Laghos contains 3 major distinct computations:

1. The inversion of the global kinematic mass matrix (CG H1).
2. The force operator evaluation from degrees of freedom to quadrature points (Forces).
3. The physics kernel in quadrature points (UpdateQuadData).

By default, Laghos is instrumented to report the total execution times and rates, in terms of millions of degrees of freedom per second (megadofs), for each of these computational phases. These rates are reported as three separate figures of merits in the table below, together with a total combined execution rate which is the reportable **Figure of Merit (FOM)** for the benchmark:

nodes	cores	order	velocity dofs/node	FOM_1	FOM_2	FOM_3	FOM
4096	65536	Q2-Q1	788,738	26696.10	11792.90	3469.28	362.56
24576	393216	Q3-Q2	664,705	168497.00	74221.40	16695.60	2072.63

These results were obtained with the following runs on the Vulcan BG/Q machine at LLNL.

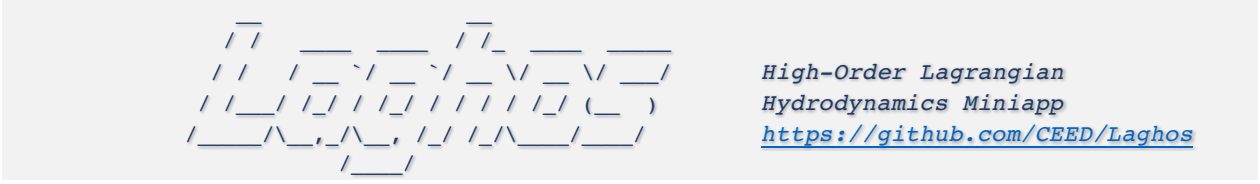
First run (1/24th of Sequoia):

```
srun -n 65536 laghos -pa -p 1 -tf 0.6 -no-vis
                     -pt 211 -m data/cube01_hex.mesh
                     --cg-tol 0 --cg-max-iter 50 --max-steps 2
                     -ok 2 -ot 1 -rs 5 -rp 3
```

Second run (1/4th of Sequoia):

```
srun -n 393216 laghos -pa -p 1 -tf 0.6 -no-vis
                      -pt 322 -m data/cube_12_hex.mesh
                      --cg-tol 0 --cg-max-iter 50 --max-steps 2
                      -ok 3 -ot 2 -rs 5 -rp 3
```

To make the last run 8 times bigger, one can either weak scale by using 8 times as many MPI tasks and increasing the number of serial refinements: `srun -n 3145728 … -rs 6 -rp 3`, or use the same number of MPI tasks but increase the local problem on each of them by doing more parallel refinements: `srun -n 393216 … -rs 5 -rp 4`.



Versions

In addition to the main MPI-based CPU implementation in https://github.com/CEED/Laghos, the following versions of Laghos have been developed:

- A serial version in the serial directory.
- GPU version based on OCCA.
- A RAJA-based version in the raja-dev branch.

Contact

You can reach the Laghos team by emailing laghos@llnl.gov or by leaving a comment in the issue tracker.

Copyright

The following copyright applies to each file in the CEED software suite, unless otherwise stated in the file:

> Copyright © 2017, Lawrence Livermore National Security, LLC. Produced at the Lawrence Livermore National Laboratory. LLNL-CODE-734707. All Rights reserved.

See files LICENSE and NOTICE for details.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, LLNL-SM-742945.