LULESH 2.0 Updates and Changes

Ian Karlin Jeff Keasler Rob Neely

Lawrence Livermore National Laboratory

`http://codesign.llnl.gov/lulesh`
LLNL-TR-641973

Contents

1 Introduction

The Livermore Unstructured Lagrange Explicit Shock Hydrodynamics (LULESH) proxy application [1] is being developed as part of the NNSA Advanced Scientific Computing (ASC) National Security Applications (NSapp) Co-design Project [6] at Lawrence Livermore National Laboratory (LLNL). Originally developed as one of five challenge problems for the DARPA UHPC program, it has since evolved and has received widespread use in Department of Energy (DOE) research programs as a mini-app representative of simplified 3D Lagrangian hydrodynamics on an unstructured mesh. This prior work includes ports to a number of research and production programming models [5], and additions to its functionality and tuning for various architectures [3, 4].

However, proxy-apps are meant to evolve over time based on changes in the modeled application, hardware and to capture additional application features [7]. Staying true to this evolutionary model we have updated LULESH to version 2.0 to both add features that make it more representative as a proxy application, as well as features to increase the usability of the mini-app.

The changes include easier usage of the proxy by allowing serial, OpenMP, MPI and MPI plus OpenMP versions to be compiled from a single source. We also have changed some of the loop structures to better reflect real applications and test compilers, added parallel I/O, and introduced optional load imbalance. Finally, we made some cosmetic changes to the code and brought a number of options including a visualization dump to the command line.

The rest of this document describes the updates to LULESH 2.0. First we focus on physics related changes. Then we discuss code refactoring changes including the I/O and viz capabilities. Finally, we describe the command line options, suggested use cases and constraints on optimization and other changes that are not spelled out in previous documentation. We do not repeat information found in the previous documentation as this document is meant to be a supplement to previous documentation [1, 4].

2 Physics Implementation Changes

There are three physics related changes in LULESH 2.0 described in the following sections. These include:

- The concept of multiple *regions*
- Dynamic timestep calculation
- Determining the end runtime of the problem

2.1 Multi-Region Problems

Hydrodynamics codes typically simulate problems containing multiple materials, which are then mapped onto *regions*, or subsets of the mesh. For purposes of LULESH the distinction between materials and regions is important, as LULESH is still hardcoded to solve the single-material Sedov blast wave problem, but now does so using multiple variable-sized regions, all of which model the same ideal gas material. In other words, there is still a single material, but it is coded *as-if* it supported multiple materials.

This change to use multiple regions allowed us to:

- Add array indirections to key kernels (e.g. `EvalEOSForElems` that are common in unstructured hydrodynamics applications.

- Dynamically allocate region-length memory chunks, instead of all memory being large domain-sized allocations, and
- The ability to artificially add minor load imbalance caused by different materials taking different amounts of calculation of the Equation of State.

The regions are implemented by an outer loop over the regions and an inner loop over the elements in a region as shown below. Each region computes the same material model that was found in the original LULESH. However, the regions are of different sizes and have a different (artificially introduced) computational cost.

```
for(int r = 0; r < domain.numReg(); ++r) {
  for(int i = 0; i < domain.regElemSize[r]; ++i) {
    int elemNum = domain.regElemlist[i] ;
    [...]
  }
}
```

Hydrodynamics codes that use the Arbitrary Lagrangian Eulalrian (ALE) formulation - a common style of hydrodynamics at LLNL - mixed zones and non-unit-stride accesses occur within a region. Since LULESH is a pure Lagrangian code we do not capture the mixed elements, however, we do capture the noncontiguous access patterns by setting up bins within a region of various sizes based on code measurements. The bin sizes and frequency each is chosen is shown in Table 1. The details of how elements in the mesh are assigned to regions is explained in comments in `lulesh-init.cc`.

Bin Size	**bin frequency**
1 - 15	77.3%
16 - 31	16.4%
32 - 63	3.3%
64 - 127	0.4%
128 - 255	0.4%
256 - 511	0.3%
512 - 2048	1.9%

Tab. 1. Bin Size Distribution

The difference in computation cost is introduced by adding additional cost to some of the regions in the `EvalEOSForElems` function by simply repeating the calculation multiple times. 50% of the regions have no additional cost. 45% have a additional cost determined by a command line parameter. The final 5% have an additional cost that is ten times that of the of the regions whose cost is specified by the command line argument. This sort of variance is typical of the imbalance of costs in evaluating material properties for various equations of state and/or strength models.

Finally, the reference implementation for multi-region loops puts OpenMP parallelism on the inner loop over non-contiguous elements within a region. However, parallelism could be put on either or both loops with the best parallelization strategy depending on machine and problem parameters. Introducing regions should increase the importance of low OpenMP overheads on manycore platforms, warp divergence and SIMT length on GPUs, and demonstrates one of the challenges of finding parallelism when some regions have a small number of elements in them.

2.2 Courant Constraint Enabled by Default

In LULESH 1.0 the serial and OpenMP version used the courant condition as a time constraint by default and all versions using MPI used a fixed time step. In LULESH 2.0 we have now reenabled the courant in all versions for consistency. Also, by using the courant condition, we add another common parallel communication pattern to LULESH - the AllReduce reduction to compute the minimum timestep across domains.

Finally, to guarantee correctness and prevent mesh tangling the initial timestep is set based on mesh resolution. Since this changes the number of timesteps to solution a new set of correctness reference values are shown in Section 4.3.

2.3 Shock Ends at about 1.0 cm

In the LULESH 1.0, the amount of energy deposited at the origin cell was constant, and as an extensive quantity meant that less energy was being imparted in problems where the element size was small. Thus, when differing numbers of cells were used in the simulation the energy concentration was changed and the shock at simulation end time was in different locations. To fix this the energy is now scaled with the mesh resolution so the shock is about 1.0 cm from the origin when the simulation ends.

3 Code Refactoring and Other Computer Science Changes

LULESH 2.0 includes code refactoring changes to separate the various elements of LULESH each into their own source file. These changes also added new functionality and updated LULESH to better reflect coding styles and loop structures currently found in some of LLNL's production applications.

3.1 Visualization and Parallel I/O

In LULESH 2.0 we add support to write visualization files for verification. Large parallel problems use "poor mans parallel I/O" (or *Multiple independent files*) from the silo library [2], and the number of file pieces is settable by the user, ranging from one-per-MPI task, to a single file, to any number inclusive. The resulting output can be read by Visit [8] to visualize the simulation and to see the distribution of regions in the domain. Both Silo and Visit are freely available for those interested in this optional capability, but are not required to work with, nor are they distributed with LULESH 2.0.

3.2 Multiple Files

LULESH 2.0 now consists of six separate source code files. However we expect most users to only need to analyze and modify `lulesh.cc`, which retains the numerics which are timed by default. The other five files contain the support infrastructure for the new visualization and parallel I/O capabilities and other non-physics support infrastructure such as command line options, MPI communication, initialization, and general utilities. The contents of each of these files is described below.

- `lulesh.cc` retains most of the numerics, and most of the code of interest to the majority of users.
- `lulesh-init.cc` contains a cleaned up constructor and some setup code.

- `lulesh-par.cc` contains all the MPI parallelism routines.
- `lulesh-util.cc` has a few miscellaneous routines - namely a command line switch parser and timer output.
- `lulesh-viz.cc` has the ability to write a parallel ”poor mans I/O” multi-part silo file.
- `lulesh.h` has the C++ class definition for *domain*.

3.3 Miscellaneous Minor Changes

Various versions of LULESH 1.0 all had different class definitions and mixes of C and C++. In particular, the serial C++ version of 1.0 used very different data structure abstractions than did the widely distributed MPI version. This made it difficult to implement and evaluate changes when moving from the serial version to parallel versions.

In LULESH 2.0, a single source code base supports serial, OpenMP, MPI, and MPI+OpenMP versions with compile-time flags. We have added more C++ features found in the LULESH 1.0 serial version, such as `stl:vectors`. The code is also now a mix of C and C++, which is representative of the source mixture found in many LLNL production applications with multi-decade lifetimes. The biggest advantage of these changes is that it is now possible to change data structure representations in all versions with a few lines of code modification in `lulesh.h` (see section 4.2 for more details).

We have also altered the loop structure of loops that iterate within a single element. In the LULESH 1.0 many of these were hand unrolled, whereas now there is a mix of loop structures - including short loop iterations over the 8 nodes of an element. These changes were put in the test compiler’s ability to optimize code written this way, while also adding to the readability and maintainability of the code. Often loops within an element occur in the most computationally intense part of hydrodynamics codes and getting good vectorization on modern hardware is important to their performance. As compilers have improved, these smaller loops with static loop bounds and stack allocated variables are often vectorized resulting in better performance than the unrolled loops. The combination of added readability with potentially improved compiler optimizations is a win-win.

4 Updated Compile and Usage Options

In this section we describe how to use various new features of LULESH 2.0. We also suggest use cases that we believe are interesting, as well as constraints that must be honored in order for LULESH to obtain maximum representation as a proxy application.

4.1 Compiling and Command Line Arguments

There is a default `Makefile` with LULESH 2.0. Using it will compile your code with MPI enabled, and without any mesh visualization. OpenMP support is enabled when the code is compiled with the appropriate OpenMP flag for your compiler (e.g. -fopenmp). These compiler flags will automatically define the symbol `_OPENMP` which enables runtime calls to the OpenMP API.

To build other versions of interest the following flags may also be changed.

- `-DUSE_MPI=<int>` is set to 0 to turn off MPI, and 1 to turn it on.
- `-DVIZ_MESH` compiles in the ability to output visualization files. To use this functionality you need to link to hdf5 and silo. An example of how to do this on LLNL machines is included in the Makefile, as are references to find downloadable versions of these libraries.

LULESH 2.0 has a wide variety of command line arguments available to the user. These include:

- `-b <int>` sets the amount of load imbalance between regions. For this option 0 is no imbalance and imbalance increases with higher inputs. The default is 1.
- `-c <int>` sets the cost of more expensive regions. Half the regions always have no extra cost. 45% have an extra cost for the EOS equal to the number entered and 5% have an extra cost of 10 times the extra cost entered.
- `-f <int>` sets the number of file parts that are output by the I/O routine. The default is the (number of processor plus 10) divided by 9.
- `-i <int>` sets the number of iterations of the simulation to run. By default the simulation is run for 0.01 seconds of simulated time.
- `-r <int>` sets the number of regions to run for each domain in the problem. The default is 11.
- `-s <int>` sets the number of elements of cube mesh along one side in a single domain. The number of domains is always equal to the number of MPI ranks. The default is 30.
- `-p` prints out simulation progress at each timestep.
- `-v` plots the regions in viz files at the end of the simulation. To use this you must have compiled with Silo.
- `-h` prints out the help message.

4.2 Suggested Usage

In this section we describe representative use cases of LULESH from past experience on multiple generations of hardware. These are provided as a good starting place for your exploration, but should not be taken as gospel. Often various useful lessons can only be learned about how codes interact with hardware by varying problem size greatly or combining features in new and interesting ways. To that end we have also provided what we consider reasonable defaults as a good starting place for users.

At LLNL, hydrodynamics is often run as part of a multi-physics simulation. For those simulations the number of elements can range anywhere from about 2,000 to 100,000 per GB of memory, with the upper end of that range limited to simple problems (like those represented by LULESH) in a pure hydrodynamics (single physics) simulation.

The number of regions in problems can vary depending on the problem being modeled, and is at least as large as the number of distinct materials in the problem. Most problems have 5 to 20 regions, but problems with up to 100 are not uncommon. Likewise, load balance of regions is very problem dependent. We believe we have provided reasonable defaults, but changing of these values to stress various systems and to see how different degrees of load imbalance effects your computer system, programming model or other interest is encouraged.

Adjusting the the load balance is done through the interplay of the -b and -c command line options. The -b option will change the relative weight of regions within a domain. The larger the value entered the greater disparity between the region with the most elements in a domain and the region with the fewest. Since each domain will have a different mix of regions the higher this value the more imbalance there will be between domains. The -c option will increase the relative imbalance between regions within a domain and also increase the imbalance between domains as long as -b is not zero. We suggest limiting -b to no greater than three or four as increasing it much further could result in the *EvalEOSForElems* becoming more than half of the overall runtime. To turn off load imbalance entirely (the default in LULESH 1.x), use `-c 0`.

In addition, the LULESH 2.0 distribution contains an additional header file named `lulesh_tuple.h`. This is an alternative header file that can replace `lulesh.h`, and demonstrates how changes limited to the header file can alter the layout of the data structures without requiring changes to the computational kernels, and potentially provide a more ideal mapping of data onto the underlying memory subsystem. This is done through use of C++ abstractions and inlined functions. By utilizing `lulesh_tuple.h`, one can quickly change from a struct of arrays to an array of structs data structure with a few minor edits.

4.3 Correctness Checking

Due to changes in how the initial timestep is set and the initial energy is deposited the correctness values in [4] are no longer correct for LULESH 2.0. The rest of the methodology to test correctness of an implementation is unchanged so we only provide an updated set of energy and iteration values in Table 2.

Size	Iteration Count	Final Origin Energy
5^3	72	7.853665e+03
10^3	231	2.720531e+04
30^3	932	2.025075e+05
45^3	1477	4.234875e+05
50^3	1662	5.124778e+05
70^3	2402	9.417145e+05
90^3	3145	1.482403e+06

Tab. 2. Common on node sizes used for correctness checking and timing studies of LULESH

However, it is also possible to reproduce LULESH 1.0 correctness with LULESH 2.0. With the following changes in lulesh-init.cc. Both changes occur at the bottom of the domain constructor.

- Change the initialization of scale from the current formula to 1.
- Change the initialization of deltatime() from the current formula to Real_t(1.0e-7).

Note that if a large mesh is run with deltatime set as it was in LULESH 1.0 the mesh will tangle and the simulation will fail.

5 Acknowledgments

Funding for this work came from the following sources. LULESH is primarily supported by the *LLNL ASC NSapp Co-design Project*, performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LULESH is based upon prior work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0001.

References

[1] Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory. Technical Report LLNL-TR-490254, Lawrence Livermore National Laboratory.

[2] Silo user's guide version 4.8. Technical Report LLNL-SM-453191, Lawrence Livermore National Laboratory, August 2010.

[3] I. Karlin, J. Mcgraw, E. Gallarado, J. Keasler, E. A. Leon, and B. Still. Memory and parallelism tuning exploration using the lulesh proxy application, November 2012.

[4] Ian Karlin, Abhinav Bhatele, Bradford L. Chamberlain, Jonathan Cohen, Zachary Devito, Maya Gokhale, Riyaz Haque, Rich Hornung, Jeff Keasler, Dan Laney, Edward Luke, Scott Lloyd, Jim McGraw, Rob Neely, David Richards, Martin Schulz, Charle H. Still, Felix Wang, and Daniel Wong. Lulesh programming model and performance ports overview. Technical Report LLNL-TR-608824, Lawrence Livermore National Laboratory, December 2012.

[5] Ian Karlin, Abhinav Bhatele, Jeff Keasler, Bradford L. Chamberlain, Jonathan Cohen, Zachary DeVito, Riyaz Haque, Dan Laney, Edward Luke, Felix Wang, David Richards, Martin Schulz, and Charles Still. Exploring traditional and emerging parallel programming models using a proxy application. In *27th IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2013)*, Boston, USA, May 2013.

[6] R. Neely, M. Heroux, and S. Swaminarayan. National security applications co-design: A framework for an asc tri-lab project. Technical report, LLNL, SNL and LANL, 2012.

[7] R. Neely, M. Heroux, and S. Swaminarayan. Asc co-design proxy app strategy. Technical report, LLNL, SNL and LANL, 2013.

[8] Visit web site. https://wci.llnl.gov/codes/visit/.