# Codee Training Series

April 25-26, 2023



**Shift Left Performance**

Automated Code inspection for Performance

April 24th, 2023

# Optimizing the Performance of the LULESHmk

**Goals:**

- Produce an OpenACC version of LULESHmk for GPUs
- Build & run an OpenACC code on the GPU

# First consideration about using Codee at NERSC

- First, remember to load the Codee module
  ```
  $ module load codee
  ```
- The flag --help lists all the options available in the Codee command-line tools
  ```
  $ pwreport --help
  $ pwloops --help
  $ pwdirectives --help
  ```
- You can run Codee command-line tools on the login nodes (no need to run them on the compute nodes)
- Build and run the example codes on the compute nodes using the batch scripts
  - Scripts tuned to use the appropriate reservations: *codee_day1*, *codee_day2*
- Remember to check the open catalog of rules for performance optimization:

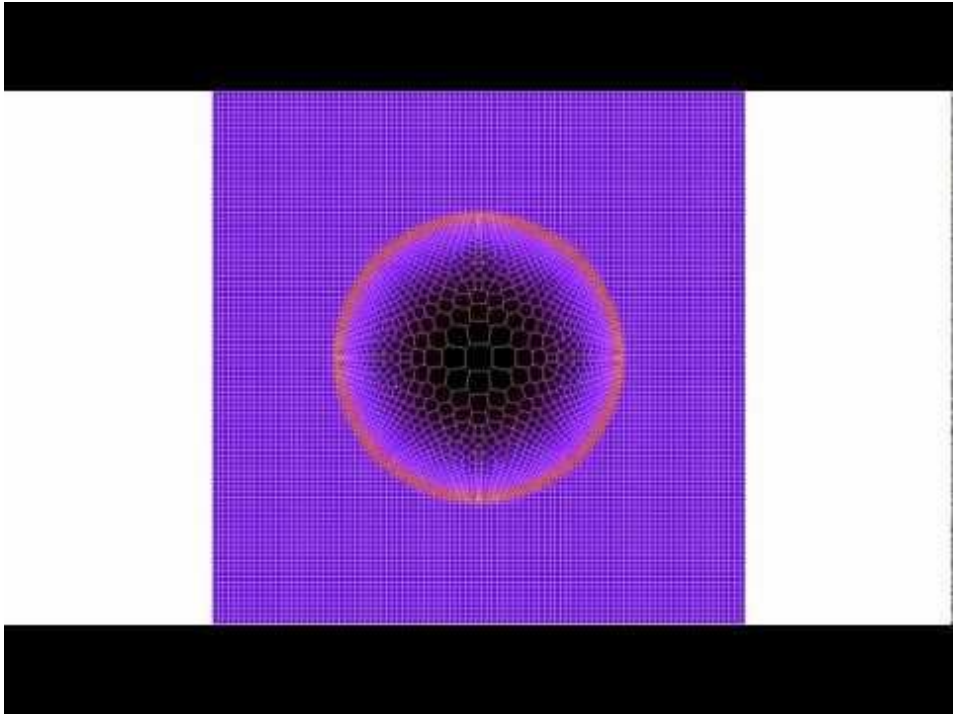
https://www.codee.com/knowledge/

# CORAL Benchmarks: LULESH

**L**ivermore **U**nstructured **L**agrange
**E**xplicit **S**hock **H**ydrodynamics

Part of a Physics Simulation software (ALE3D)

Models the propagation of a Sedov blast wave using Lagrangian hydrodynamics



# Profiling of LULESHmk

```
$ gcc -pg -o luleshmk luleshmk.c -lm
$ ./luleshmk
$ gprof ./luleshmk
```

Note: we use GCC for a quicker profiling using the GPROF profiling tool, which reports the functions that consumes most of the runtime.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 52.65      8.22     8.22 223680000     0.00     0.00  CalcElemFBHourglassForce_workload
 16.54     10.81     2.58      932     0.00     0.00  ApplyMaterialPropertiesForElems_workload
 13.27     12.88     2.07 27960000     0.00     0.00  CalcElemVelocityGradient_workload
 10.64     14.54     1.66 27960000     0.00     0.00  CalcElemFBHourglassForce
  5.00     15.32     0.78      932     0.00     0.01  CalcFBHourglassForceForElems
  1.15     15.51     0.18        1     0.18    15.64  luleshmk
...
```

The hotspot covers >50% of the runtime.

Altogether the first 3 hotspots cover up to >90% of the total runtime.

# How to verify correctness of LULESHmk ?

```
$ ./luleshmk
Run completed:
   Problem size        =  30
   MPI tasks           =  1
   Iteration count     =  932
   Final Origin Energy = 9.330000e+02
   Testing Plane 0 of Energy Array on rank 0:
        MaxAbsDiff   = 8.178369e+06
        TotalAbsDiff = 1.267647e+09
        MaxRelDiff   = 9.665601e-01

Elapsed time         =      25.34 (s)
Grind time (us/z/c)  =  1.0068203 (per dom)  ( 1.0068203 overall)
FOM                  =  993.22589 (z/s)

numNodes   = 27000
numElems   = 30000
checksum_f = 3.28901e+11
checksum_e = 4.37594e+12
```

# The source code of LULESHmk Function main()

```
  362    int main(int argc, char *argv[]) {
    . . .

  381             Real_t locDom_e[NUM_ELEMS];  // Check if 900 is enouch = nx*nx with opt_nx=30 below
  382             Real_t locDom_m_dxx[NUM_ELEMS];
  383             Real_t locDom_m_dyy[NUM_ELEMS];
  384             Real_t locDom_m_dzz[NUM_ELEMS];
  385             Real_t vnew[NUM_ELEMS];
  386             for(int i=0; i<NUM_ELEMS; ++i) {
  387             locDom_e[i] = i + 1.0;
  388             locDom_m_dxx[i] = i + 1.0;
  389             locDom_m_dyy[i] = i + 1.0;
  390             locDom_m_dzz[i] = i + 1.0;
  391             vnew[i] = i + 1.0;
  392             }
  393
  394             Index_t locDom_m_nodelist[MAX_NODELIST];
  395             for(int i=0; i<MAX_NODELIST; ++i) {
  396             locDom_m_nodelist[i] = i % NUM_NODES;  // Indirections are bounded to NUM_NODES
  397             }
  398
  399             Real_t locDom_f[NUM_NODES];   // Added in the miniapp for verification purposes
  400             Real_t locDom_m_fx[NUM_NODES];  // LULESH compute force at each node mesh, so 27000 on x
  401             Real_t locDom_m_fy[NUM_NODES];  // and 27000 on y
  402             Real_t locDom_m_fz[NUM_NODES];  // and 27000 on z
  403             for(int i=0; i<NUM_NODES; ++i) {
  404             locDom_f[i] = 2.0;
  405             locDom_m_fx[i] = 2.0;
  406             locDom_m_fy[i] = 3.0;
  407             locDom_m_fz[i] = 4.0;
  408             }
    . . .
  429             luleshmk(p.param_tol, . . . );
    . . .
}
```

SIZE AND COMPLEXITY OF LULESHmk

```
Number of files:           1
Number of functions:      15
Number of loops:          19
Number of lines of code: 443
```

# The source code of LULESHmk
## Hotspot function *CalcFBHourglassForceForElems()*

```
 68    void CalcElemFBHourglassForce(Index_t i2,. . . ) {
 74     for(Index_t i = 0; i < 8; i++) {
 75               double T = CalcElemFBHourglassForce_workload( WORKLOAD_CalcElemFBHourglassForce );
 76               hgfx[i] = gamma[0][0] * T;
 77               hgfy[i] = gamma[0][1] * T;
 78               hgfz[i] = gamma[0][2] * T;
 79     }
 80    }


 82    void CalcFBHourglassForceForElems( Index_t numElem,. . . ) {
    // FUNCTION: Calculates the Flanagan-Belytschko anti-hourglass force.
    . . .
130    /*        compute the hourglass modes */
131
132       for(Index_t i2=0;i2<numElem;++i2){
133               Real_t hgfx[8], hgfy[8], hgfz[8] ;
134
135               CalcElemFBHourglassForce(i2, gamma, hgfx, hgfy, hgfz);
    . . .
139               Index_t n0si2 = domain_m_nodelist[(8)*i2+0];
140               Index_t n1si2 = domain_m_nodelist[(8)*i2+1];
    . . .
146               Index_t n7si2 = domain_m_nodelist[(8)*i2+7];
147
148               domain_m_fx[n0si2] += hgfx[0];
149               domain_m_fy[n0si2] += hgfy[0];
150               domain_m_fz[n0si2] += hgfz[0];
     . . .
176               domain_m_fx[n7si2] += hgfx[7];
177               domain_m_fy[n7si2] += hgfy[7];
178               domain_m_fz[n7si2] += hgfz[7];
179       }
180    }
```

Important note 1:
The loop body contains functions calls!
Even nested function calls!

Important note 2:
The loop body contains reductions on arrays, with indirections on read/write access to the arrays!

# Produce Codee Checks Report and follow suggestions!

```
$ pwreport --checks --verbose luleshmk.c:CalcFBHourglassForceForElems --include-tags gpu -- -lm -fast
Compiler flags: -lm -fast

[C] target compiler: <none> (Compiler Agnostic Mode)

CHECKS REPORT

luleshmk.c:132:4 [PWR034]: avoid strided array access for variable 'domain_m_nodelist' to improve performance
  Strided array access:
    139:       Index_t n0si2 = domain_m_nodelist[(8)*i2+0];
    140:       Index_t n1si2 = domain_m_nodelist[(8)*i2+1];
    141:       Index_t n2si2 = domain_m_nodelist[(8)*i2+2];
    142:       Index_t n3si2 = domain_m_nodelist[(8)*i2+3];
    143:       Index_t n4si2 = domain_m_nodelist[(8)*i2+4];
    144:       Index_t n5si2 = domain_m_nodelist[(8)*i2+5];
    145:       Index_t n6si2 = domain_m_nodelist[(8)*i2+6];
    146:       Index_t n7si2 = domain_m_nodelist[(8)*i2+7];
  Suggestion: consider using techniques like loop fusion, loop interchange, loop tiling or changing the data layout to avoid
non-sequential access to variable 'domain_m_nodelist'.
  Documentation: https://www.codee.com/knowledge/pwr034

luleshmk.c:132:4 [PWR057]: consider applying offloading parallelism to sparse reduction loop
  Suggestion: use pwdirectives to automatically optimize the code
  Documentation: https://www.codee.com/knowledge/pwr057
  AutoFix:
    * Using OpenMP (recommended):
        pwdirectives --offload omp-teams --in-place luleshmk.c:132:4 -- -lm -fast
    * Using OpenAcc:
        pwdirectives --offload acc --in-place luleshmk.c:132:4 -- -lm -fast

SUGGESTIONS

  More details on the defects, recommendations and more in the Knowledge Base:
        https://www.codee.com/knowledge/

1 file successfully analyzed and 0 failures in 50 ms
```

Filter Codee output and focus on checks targeting GPU only

# Generate code optimized for GPU with OpenACC directives

```
$ pwdirectives --offload acc luleshmk.c:132:4 -o luleshmk_acc.c
Results for file 'luleshmk.c':
  Successfully parallelized loop at 'luleshmk.c:CalcFBHourglassForceForElems:132:4' [using offloading without teams]:
      [INFO] luleshmk.c:132:4 Parallel sparse reduction pattern identified for variable 'domain_m_fz' with associative, commutative operator '+'
      [INFO] luleshmk.c:132:4 Parallel sparse reduction pattern identified for variable 'domain_m_fy' with associative, commutative operator '+'
      [INFO] luleshmk.c:132:4 Parallel sparse reduction pattern identified for variable 'domain_m_fx' with associative, commutative operator '+'
      [INFO] luleshmk.c:132:4 Available parallelization strategies for variable 'domain_m_fz'
      [INFO] luleshmk.c:132:4   #1 OpenACC atomic access (* implemented)
      [INFO] luleshmk.c:132:4 Available parallelization strategies for variable 'domain_m_fy'
      [INFO] luleshmk.c:132:4   #1 OpenACC atomic access (* implemented)
      [INFO] luleshmk.c:132:4 Available parallelization strategies for variable 'domain_m_fx'
      [INFO] luleshmk.c:132:4   #1 OpenACC atomic access (* implemented)
      [INFO] luleshmk.c:132:4 Parallel region defined by OpenACC directive 'parallel'
      [INFO] luleshmk.c:132:4 Loop parallelized with OpenACC directive 'loop'
      [INFO] luleshmk.c:132:4 Complete access range for variables: 'domain_m_nodelist', 'domain_m_fz', 'domain_m_fy', 'domain_m_fx'
      [INFO] luleshmk.c:132:4 Data region for host-device data transfers defined by OpenACC directive 'data'
      [INFO] luleshmk.c:132:4 Make sure there is no aliasing among variables: domain_m_fz, domain_m_fy, domain_m_fx
Successfully created luleshmk_acc.c

Minimum software stack requirements: OpenACC version 2.0 with offloading capabilities
```

Codee produces OpenACC data transfer directives with incomplete access ranges for array *domain_m_fz* (similar for *domain_m_fy*, *domain_m_fx*, *domain_m_nodelist*)

Note: Programmer must specify access ranges manually

Codee reports *sparse reductions* (i.e. reduction on arrays with read/write indirections) and guarantees correctness of the OpenACC code through *atomic* protection on *domain_m_fz* (similar for *domain_m_fy*, *domain_m_fx*)

# Add missing information to the OpenACC code manually

```
$ cat -n luleshmk_acc.c
. . .
    82    void CalcFBHourglassForceForElems( Index_t numElem,
    // FUNCTION: Calculates the Flanagan-Belytschko anti-hourglass force.
   . . .
   130    /*    compute the hourglass modes */
   131
   132       #pragma acc data copyin(domain_m_nodelist[:], gamma[0:4][0:8], numElem) copy(domain_m_fx[:], domain_m_fy[:], domain_m_fz[:])
   133       #pragma acc parallel
   134       #pragma acc loop
   135       for(Index_t i2=0;i2<numElem;++i2){
   136          Real_t hgfx[8], hgfy[8], hgfz[8] ;
   . . .
   138          CalcElemFBHourglassForce(i2, gamma, hgfx, hgfy, hgfz);
   . . .
   142          Index_t n0si2 = domain_m_nodelist[(8)*i2+0];
   143          Index_t n1si2 = domain_m_nodelist[(8)*i2+1];
        . . .
   149          Index_t n7si2 = domain_m_nodelist[(8)*i2+7];
   150
   151             #pragma acc atomic update
   152             domain_m_fx[n0si2] += hgfx[0];
   153             #pragma acc atomic update
   154             domain_m_fy[n0si2] += hgfy[0];
   155             #pragma acc atomic update
   156             domain_m_fz[n0si2] += hgfz[0];
        . . .
   200             #pragma acc atomic update
   201             domain_m_fx[n7si2] += hgfx[7];
   202             #pragma acc atomic update
   203             domain_m_fy[n7si2] += hgfy[7];
   204             #pragma acc atomic update
   205             domain_m_fz[n7si2] += hgfz[7];
   206       }
   207    }
```

The OpenACC directives generated by Codee include incomplete array ranges in the data transfers. The programmer must do the following replacements:

domain_m_nodelist[:] → domain_m_nodelist[0:MAX_NODELIST]
domain_m_fx[:] → domain_m_fx[0:NUM_NODES]
domain_m_fy[:] →domain_m_fy[0:NUM_NODES]
domain_m_fz[:] →domain_m_fz[0:NUM_NODES]

# Verify the OpenACC version of LULESHmk with Codee!

```
$ pwreport --checks --verbose luleshmk_acc_fix.c:CalcFBHourglassForceForElems --include-tags gpu -- -lm -fast
Compiler flags: -lm -fast

[C] target compiler: <none> (Compiler Agnostic Mode)

CHECKS REPORT

luleshmk_acc_fix.c:138:7 [PWR027]: consider annotating function with Acc 'routine'
  Suggestion: annotate the definition of function 'CalcElemFBHourglassForce' at line 68 with '#pragma acc routine', using 'seq',
'vector', 'worker' or 'gang' to specify the required level of parallelism
  Documentation: https://www.codee.com/knowledge/pwr027

luleshmk_acc_fix.c:135:4 [PWR034]: avoid strided array access for variable 'domain_m_nodelist' to improve performance
  Strided array access:
    142:       Index_t n0si2 = domain_m_nodelist[(8)*i2+0];
    143:       Index_t n1si2 = domain_m_nodelist[(8)*i2+1];
    144:       Index_t n2si2 = domain_m_nodelist[(8)*i2+2];
    145:       Index_t n3si2 = domain_m_nodelist[(8)*i2+3];
    146:       Index_t n4si2 = domain_m_nodelist[(8)*i2+4];
    147:       Index_t n5si2 = domain_m_nodelist[(8)*i2+5];
    148:       Index_t n6si2 = domain_m_nodelist[(8)*i2+6];
    149:       Index_t n7si2 = domain_m_nodelist[(8)*i2+7];
  Suggestion: consider using techniques like loop fusion, loop interchange, loop tiling or changing the data layout to avoid
non-sequential access to variable 'domain_m_nodelist'.
  Documentation: https://www.codee.com/knowledge/pwr034

SUGGESTIONS

  74 lines of code in parallel regions were found in your code, get more information with pwreport:
       pwreport --parallelization --verbose luleshmk_acc_fix.c:CalcFBHourglassForceForElems --include-tags gpu -- -lm -fast

  More details on the defects, recommendations and more in the Knowledge Base:
       https://www.codee.com/knowledge/

1 file successfully analyzed and 0 failures in 50 ms
```

Verify the OpenACC-enabled source code of LULESHmk with Codee *pwreport --checks* and discover that it points out functions that need to be explicitly offloaded to the GPU with OpenACC pragma "*routine seq*".

Note: See in the next slide that *nvc* adds the pragma *routine* implicitly, but annotating the source code explicitly helps creating code that is more portable to other environments and compilers.

# Understanding the detailed output of the OpenACC compiler

```
$ nvc -fast -acc -target=gpu -Minfo=all luleshmk_acc.c -o luleshmk_acc
CalcElemFBHourglassForce_workload:
     58, Generating implicit acc routine seq
         Generating acc routine seq
         Generating NVIDIA GPU code
     61, Generated vector simd code for the loop containing reductions
     63, FMA (fused multiply-add) instruction(s) generated
CalcElemFBHourglassForce:
     62, FMA (fused multiply-add) instruction(s) generated
     73, Generating implicit acc routine seq
         Generating acc routine seq
         Generating NVIDIA GPU code
     75, CalcElemFBHourglassForce_workload inlined, size=8 (inline) file luleshmk_acc.c (58)
          61, Generated vector simd code for the loop containing reductions
CalcFBHourglassForceForElems:
    133, Generating copy(domain_m_fz[:27000]) [if not already present]
         Generating copyin(domain_m_nodelist[:240000],gamma[:][:],numElem) [if not already present]
         Generating copy(domain_m_fx[:27000],domain_m_fy[:27000]) [if not already present]
    135, Generating NVIDIA GPU code
         74, #pragma acc loop seq
         75, #pragma acc loop seq
        137, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
    135, Local memory used for hgfz,hgfy,hgfx
    137, Loop not fused: no successor loop
    140, CalcElemFBHourglassForce inlined, size=21 (inline) file luleshmk_acc.c (73)
          74, Loop is parallelizable
          75, CalcElemFBHourglassForce_workload inlined, size=8 (inline) file luleshmk_acc.c (58)
          75, Loop is parallelizable
```

Important note 1:
- The nvc compiler offloads the loop:137
- It manages the functions called in the loop
- Automatically adds #pragma acc routine seq
- Programmer not responsible for that

Hotspot loop offloaded to the GPU along with the corresponding data to do the computations correctly

# Benchmarking on Perlmutter @NERSC using Nvidia toolchain

```
$ nvc luleshmk.c -o luleshmk

$ ./luleshmk
- Configuring the test...
- Executing the test...
- Verifying the test...
Run completed:
   Problem size        =  30
   MPI tasks           =  1
   Iteration count     =  932
   Final Origin Energy = 9.330000e+02
   Testing Plane 0 of Energy Array on rank 0:
        MaxAbsDiff   = 8.178369e+06
        TotalAbsDiff = 1.267647e+09
        MaxRelDiff   = 9.665601e-01


Elapsed time         =      18.43 (s)
Grind time (us/z/c)  = 0.73245649 (per dom)  (0.73245649 overall)
FOM                  =  1365.2688 (z/s)


numNodes    = 27000
numElems    = 30000
checksum_f = 3.28901e+11
checksum_e = 4.37594e+12
```

```
$ nvc -acc -target=gpu luleshmk_acc.c -o luleshmk_acc

$ ./luleshmk_acc
- Configuring the test...
- Executing the test...
- Verifying the test...
Run completed:
   Problem size        =  30
   MPI tasks           =  1
   Iteration count     =  932
   Final Origin Energy = 9.330000e+02
   Testing Plane 0 of Energy Array on rank 0:
        MaxAbsDiff   = 8.178369e+06
        TotalAbsDiff = 1.267647e+09
        MaxRelDiff   = 9.665601e-01


Elapsed time         =       4.22 (s)
Grind time (us/z/c)  = 0.16770062 (per dom)  (0.16770062 overall)
FOM                  =  5963.0071 (z/s)


numNodes    = 27000
numElems    = 30000
checksum_f = 3.28901e+11
checksum_e = 4.37594e+12
```

OpenACC offload yields speedup 4.37x

LULESHmk code runs correctly on the GPU @perlmutter using OpenACC offload.
Runs faster with default optimization flags,
and note that further optimizations are still possible (e.g. minimize data transfers)

# Benchmarking on Perlmutter @NERSC using Nvidia toolchain

```
$ nvc -fast luleshmk.c -o luleshmk

$ ./luleshmk
- Configuring the test...
- Executing the test...
- Verifying the test...
Run completed:
   Problem size          =  30
   MPI tasks             =  1
   Iteration count       =  932
   Final Origin Energy = 9.330000e+02
   Testing Plane 0 of Energy Array on rank 0:
        MaxAbsDiff   = 8.178369e+06
        TotalAbsDiff = 1.267647e+09
        MaxRelDiff   = 9.665601e-01


Elapsed time         =       0.96 (s)
Grind time (us/z/c)  = 0.038327006 (per dom)  (0.038327006 overall)
FOM                  =  26091.263 (z/s)


numNodes    = 27000
numElems    = 30000
checksum_f = 3.28901e+11
checksum_e = 4.37594e+12
```

```
$ nvc -fast -acc -target=gpu luleshmk_acc.c -o luleshmk_acc

$ ./luleshmk_acc
- Configuring the test...
- Executing the test...
- Verifying the test...
Run completed:
   Problem size        =  30
   MPI tasks           =  1
   Iteration count     =  932
   Final Origin Energy = 9.330000e+02
   Testing Plane 0 of Energy Array on rank 0:
        MaxAbsDiff   = 8.178369e+06
        TotalAbsDiff = 1.267647e+09
        MaxRelDiff   = 9.665601e-01


Elapsed time         =       1.34 (s)
Grind time (us/z/c)  = 0.05319738 (per dom)  (0.05319738 overall)
FOM                  =  18797.918 (z/s)


numNodes    = 27000
numElems    = 30000
checksum_f = 3.28901e+11
checksum_e = 4.37594e+12
```

OpenACC offload yields speedup 0.72x

LULESHmk runs correctly but slower on the GPU @perlmutter using OpenACC offload when the maximum optimization level *-fast* is enabled in the Nvidia compiler.

codee

**www.codee.com**

info@codee.com

Subscribe: codee.com/newsletter/

USA - Spain

codee_com

company/codee-com/