Codee Training Series

April 26-27, 2022

Shift Left Performance

Automated Code inspection for Performance

Optimizing the Performance of the LULESHmk

Goals:

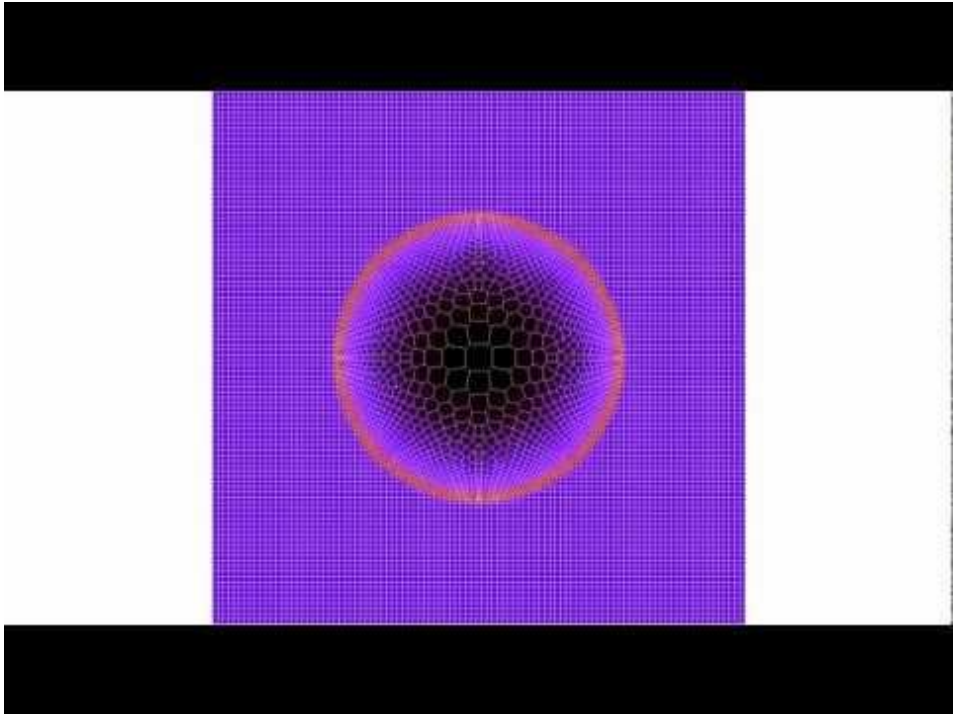
- Produce an OpenACC version of LULESHmk for GPUs
- Build & run an OpenMP code on the GPU
- Build & run an OpenMP code on the GPU

CORAL Benchmarks: LULESH

Livermore **U**nstructured **L**agrange
Explicit **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 journey towards GPU in this workshop: LULESKmk

		Challenges of GPU acceleration addressed in introductory course			Other GPU programming challenges to be addressed in next advanced course			
		Find opportunities for offloading	Optimize memory layout for data transfers	Identify defects in data transfers	Exploit massive parallelism through loop nest collapsing	Minimize data transfers across consecutive loop nests	Minimize data transfers through convergence loops	Identify auxiliary functions to be offloaded
Example codes used in this introductory course	**PI**	X	-	-	-	-	-	-
	MATMUL	X	X	X	X	X	-	-
	LULESHmk	X	X	X	X	X	X	X
	HEAT	X	-	-	-	X	X	-
	Your code!	Probably all of these challenges apply, and even more!						

The source code of LULESHmk Function main()

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

    Real_t locDom_e[NUM_ELEMS];  // Check if 900 is enouch = nx*nx with opt_nx=30 below
    Real_t locDom_m_dxx[NUM_ELEMS];
    Real_t locDom_m_dyy[NUM_ELEMS];
    Real_t locDom_m_dzz[NUM_ELEMS];
    Real_t vnew[NUM_ELEMS];
    for(int i=0; i<NUM_ELEMS; ++i) {
        locDom_e[i] = i + 1.0;
        locDom_m_dxx[i] = i + 1.0;
        locDom_m_dyy[i] = i + 1.0;
        locDom_m_dzz[i] = i + 1.0;
        vnew[i] = i + 1.0;
    }

    Index_t locDom_m_nodelist[MAX_NODELIST];
    for(int i=0; i<MAX_NODELIST; ++i) {
        locDom_m_nodelist[i] = i % NUM_NODES;  // Indirections are bounded to NUM_NODES
    }

    Real_t locDom_f[NUM_NODES];     // Added in the miniapp for verification purposes
    Real_t locDom_m_fx[NUM_NODES];  // LULESH compute force at each node mesh, so 27000 on x
    Real_t locDom_m_fy[NUM_NODES];  // and 27000 on y
    Real_t locDom_m_fz[NUM_NODES];  // and 27000 on z
    for(int i=0; i<NUM_NODES; ++i) {
        locDom_f[i] = 2.0;
        locDom_m_fx[i] = 2.0;
        locDom_m_fy[i] = 3.0;
        locDom_m_fz[i] = 4.0;
    }

    . . .
    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
```

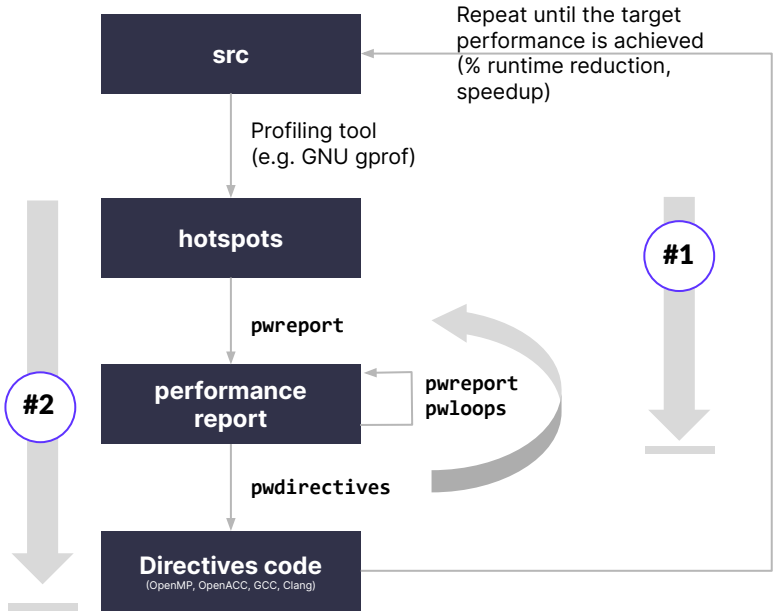
The source code of LULESHmk
Hotspot function *CalcFBHourglassForceForElems()*

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


void CalcFBHourglassForceForElems( Index_t numElem, . . . ) {

   // FUNCTION: Calculates the Flanagan-Belytschko anti-hourglass force.
   . . .
   // Compute the hourglass modes

   for(Index_t i2=0;i2<numElem;++i2){
      Real_t hgfx[8], hgfy[8], hgfz[8] ;

      CalcElemFBHourglassForce(i2, gamma, hgfx, hgfy, hgfz);
      Index_t n0si2 = domain_m_nodelist[(8)*i2+0];
      Index_t n1si2 = domain_m_nodelist[(8)*i2+1];
      . . .
      Index_t n7si2 = domain_m_nodelist[(8)*i2+7];

         domain_m_fx[n0si2] += hgfx[0];
         domain_m_fy[n0si2] += hgfy[0];
         domain_m_fz[n0si2] += hgfz[0];

         . . .

         domain_m_fx[n7si2] += hgfx[7];
         domain_m_fy[n7si2] += hgfy[7];
         domain_m_fz[n7si2] += hgfz[7];

   }
}
```

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!

A systematic, predictable approach to performance optimization with Codee



#1 **Get the performance optimization report for the whole code base**

#2 **Create performance-optimized code for the hotspot automatically**

The same steps used for the simple codes of PI and MATMUL are applicable to other codes like LULESHmk, with a bigger size and a higher complexity.

1: Produce the entry-level report for default #actions (`pwreport --evaluation`)

```
$ pwreport --evaluation luleshmk.c
Target     Lines of code Analyzed lines Analysis time # actions Effort Cost    Profiling
---------- ------------- -------------- ------------- --------- ------ ------- ---------
luleshmk.c 443           207            162 ms        27        190 h  6217€   n/a

ACTIONS PER STAGE OF THE PERFORMANCE OPTIMIZATION ROADMAP
Target     Serial scalar Serial control Serial memory Vectorization Multithreading Offloading
---------- ------------- -------------- ------------- ------------- -------------- ----------
luleshmk.c 0             3              2             22            n/a            n/a

Target : analyzed directory or source code file
Lines of code : total lines of code found in the target (computed the same way as the sloccount tool)
Analyzed lines : relevant lines of code successfully analyzed
Analysis time : time required to analyze the target
# actions : total actionable items (opportunities, recommendations, defects and remarks) detected
Effort : estimated number of hours it would take to carry out all actions (serial scalar, serial control, serial memory, vectorization,
multithreading and offloading with 1, 2, 4, 8, 12 and 16 hours respectively)
Cost : estimated cost in euros to carry out all the actions, paying the average salary of 56,286€/year for a professional C/C++ developer
working 1720 hours per year
Profiling : estimation of overall execution time required by this target

SUGGESTIONS
  You can specify multiple inputs which will be displayed as multiple rows (ie. targets) in the table, eg:
        pwreport --evaluation some/other/dir luleshmk.c

  Use --actions to find out details about the detected actions:
        pwreport --actions luleshmk.c

  Multithreading and offloading actions are filtered by default. Use --include-tags to enable them:
        pwreport --include-tags all luleshmk.c

  You can focus on a specific optimization type by filtering by its tag (serial-scalar, serial-control, serial-memory, vectorization,
multithreading, offloading), eg.:
        pwreport --actions --include-tags serial-scalar luleshmk.c

1 file successfully analyzed and 0 failures in 162 ms
```

By default multithreading and offloading are disabled in Codee.

Rationale: Codee forces the user to explicitly enable multithreading and offloading capabilities to avoid common errors resulting from a misconfigured software environment (eg. lack of an OpenMP compiler with offload)

2: Produce the entry-level report for ALL #actions (`pwreport --evaluation --include-tags all`)

```
$ pwreport --evaluation --include-tags all luleshmk.c
Target     Lines of code Analyzed lines Analysis time # actions Effort Cost     Profiling
---------- ------------- -------------- ------------- --------- ------ -------- ---------
luleshmk.c 443           207            153 ms        49        498 h  16296€   n/a

ACTIONS PER STAGE OF THE PERFORMANCE OPTIMIZATION ROADMAP
Target     Serial scalar Serial control Serial memory Vectorization Multithreading Offloading
---------- ------------- -------------- ------------- -------------- -------------- ----------
luleshmk.c 0             3              2             22             11             11

Target : analyzed directory or source code file
Lines of code : total lines of code found in the target (computed the same way as the sloccount tool)
Analyzed lines : relevant lines of code successfully analyzed
Analysis time : time required to analyze the target
# actions : total actionable items (opportunities, recommendations, defects and remarks) detected
Effort : estimated number of hours it would take to carry out all actions (serial scalar, serial control, serial memory, vectorization,
multithreading and offloading with 1, 2, 4, 8, 12 and 16 hours respectively)
Cost : estimated cost in euros to carry out all the actions, paying the average salary of 56,286€/year for a professional C/C++ developer
working 1720 hours per year
Profiling : estimation of overall execution time required by this target

SUGGESTIONS
  You can specify multiple inputs which will be displayed as multiple rows (ie. targets) in the table, eg:
        pwreport --evaluation some/other/dir --include-tags all luleshmk.c

  Use --actions to find out details about the detected actions:
        pwreport --actions --include-tags all luleshmk.c

  You can focus on a specific optimization type by filtering by its tag (serial-scalar, serial-control, serial-memory, vectorization,
multithreading, offloading), eg.:
        pwreport --actions --include-tags serial-scalar luleshmk.c

1 file successfully analyzed and 0 failures in 153 ms
```

By enabling ALL actions in the report now identifies 11 offload opportunity

3: Produce the report of ALL #actions per type of loops (`pwreport --evaluation --include-tags all --level 2`)

```
$ pwreport --evaluation --level 2 --include-tags all luleshmk.c
Target     Lines of code Analyzed lines Analysis time # actions Effort Cost     Profiling
---------- ------------- -------------- ------------- --------- ------ -------- ---------
luleshmk.c 443           207            152 ms        49        498 h  16296€   n/a

ACTIONS PER STAGE OF THE PERFORMANCE OPTIMIZATION ROADMAP
Target     Serial scalar Serial control Serial memory Vectorization Multithreading Offloading
---------- ------------- -------------- ------------- ------------- -------------- ----------
luleshmk.c 0             3              2             22            11             11

ACTIONS PER LOOP PER STAGE OF THE PERFORMANCE OPTIMIZATION ROADMAP
Difficulty No. Loops Serial scalar Serial control Serial memory Vectorization Multithreading Offloading
---------- --------- ------------- -------------- ------------- ------------- -------------- ----------
Low        15        0             0              1             20            9              9
Medium     4         0             0              1             2             2              2
High       0         0             0              0             0             0              0

Target : analyzed directory or source code file
Lines of code : total lines of code found in the target (computed the same way as the sloccount tool)
Analyzed lines : relevant lines of code successfully analyzed
Analysis time : time required to analyze the target
# actions : total actionable items (opportunities, recommendations, defects and remarks) detected
Effort : estimated number of hours it would take to carry out all actions (serial scalar, serial control, serial memory, vectorization, multithreading and offloading with
1, 2, 4, 8, 12 and 16 hours respectively)
Cost : estimated cost in euros to carry out all the actions, paying the average salary of 56,286€/year for a professional C/C++ developer working 1720 hours per year
Profiling : estimation of overall execution time required by this target

SUGGESTIONS
  You can specify multiple inputs which will be displayed as multiple rows (ie. targets) in the table, eg:
        pwreport --evaluation some/other/dir --include-tags all luleshmk.c

  Use --actions to find out details about the detected actions:
        pwreport --actions --include-tags all luleshmk.c

  You can focus on a specific optimization type by filtering by its tag (serial-scalar, serial-control, serial-memory, vectorization, multithreading, offloading), eg.:
        pwreport --actions --include-tags serial-scalar luleshmk.c

1 file successfully analyzed and 0 failures in 152 ms
```

By increasing the details of the report, the tool reports that ALL the actions are identified in the scope of loops that have LOW difficulty from the performance optimization viewpoint

4: Produce the Codee Actions Report for the target function (`pwreport --actions`)

```
$ pwreport --actions --include-tags all luleshmk.c:CalcFBHourglassForceForElems
ACTIONS REPORT

  FUNCTION BEGIN at luleshmk.c:CalcFBHourglassForceForElems:82:1
    82: void CalcFBHourglassForceForElems( Index_t numElem,

    LOOP BEGIN at luleshmk.c:CalcFBHourglassForceForElems:132:4
      132:    for(Index_t i2=0;i2<numElem;++i2){

      [PWR021] luleshmk.c:132:4 extract temporary computations to a vectorizable loop
      [PWR034] luleshmk.c:132:4 avoid strided array access for variable 'domain_m_nodelist' to improve performance

      [OPP001] luleshmk.c:132:4 is a multi-threading opportunity
      [OPP003] luleshmk.c:132:4 is an offload opportunity
    LOOP END
  FUNCTION END

CODE COVERAGE
  Analyzable files:            1 / 1     (100.00 %)
  Analyzable functions:        1 / 1     (100.00 %)
  Analyzable loops:            1 / 1     (100.00 %)
  Parallelized SLOCs:          0 / 71    (  0.00 %)

METRICS SUMMARY
  Total recommendations:         2
  Total opportunities:           2
  Total defects:                 0
  Total remarks:                 0

SUGGESTIONS

  Use --level 0|1|2 to get more details, e.g:
        pwreport --level 2 --actions --include-tags all luleshmk.c:CalcFBHourglassForceForElems

  2 recommendations were found in your code, get more information with pwreport:
        pwreport --actions --include-tags pwr luleshmk.c:CalcFBHourglassForceForElems

  2 opportunities for parallelization were found in your code, get more information with pwloops:
        pwloops luleshmk.c:CalcFBHourglassForceForElems

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

Each action is reported in the scope of the corresponding loop:
- multithreading (loop:132 OPP001)
- offloading (loop:132 OPP003)

5: Produce the detailed actions for the target function (`pwreport --actions --level 2`)

```
$ pwreport --actions --level 2 --include-tags all luleshmk.c:CalcFBHourglassForceForElems
ACTIONS REPORT

  FUNCTION BEGIN at luleshmk.c:CalcFBHourglassForceForElems:82:1
    82: void CalcFBHourglassForceForElems( Index_t numElem,

    LOOP BEGIN at luleshmk.c:CalcFBHourglassForceForElems:132:4
    . . .

      [OPP003] luleshmk.c:132:4 is an offload opportunity
        Compute patterns:
          - 'sparse' over the variable 'domain_m_fz'
          - 'sparse' over the variable 'domain_m_fy'
          - 'sparse' over the variable 'domain_m_fx'

        SUGGESTION: use pwloops to get more details or pwdirectives to generate directives:
          pwloops luleshmk.c:CalcFBHourglassForceForElems:132:4
          pwdirectives --omp offload luleshmk.c:CalcFBHourglassForceForElems:132:4 --in-place
          pwdirectives --acc luleshmk.c:CalcFBHourglassForceForElems:132:4 --in-place

        More information on: https://www.appentra.com/knowledge/opportunities

    LOOP END
  FUNCTION END
. . .
```

By enabling the detailed report for OPP003 (offload opportunity) you obtain suggestions to invoke pwdirectives for automatic annotation of the source code with OpenMP and OpenACC offload directives

(note: source code edited "in-place" by default")

6: Annotate the code for GPU + OpenACC (`pwdirectives --acc`)

```
$ pwdirectives --acc luleshmk.c:CalcFBHourglassForceForElems: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'
Successfully created luleshmk_acc.c

Minimum software stack requirements: OpenACC rsion 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*)

7: Add missing information to the OpenACC code manually

```
$ cat luleshmk_acc.c
. . .
void CalcFBHourglassForceForElems( Index_t numElem . . .
   // FUNCTION: Calculates the Flanagan-Belytschko anti-hourglass force.
   . . .
   // Compute the hourglass modes

   #pragma acc data copyin(domain_m_nodelist[:], gamma[0][0:8], numElem) copy(domain_m_fx[:], domain_m_fy[:], domain_m_fz[:])
   {
   #pragma acc parallel
   {
   #pragma acc loop
   for(Index_t i2=0;i2<numElem;++i2){
      Real_t hgfx[8], hgfy[8], hgfz[8] ;

      CalcElemFBHourglassForce(i2, gamma, hgfx, hgfy, hgfz);
      Index_t n0si2 = domain_m_nodelist[(8)*i2+0];
      Index_t n1si2 = domain_m_nodelist[(8)*i2+1];
      . . .
      Index_t n7si2 = domain_m_nodelist[(8)*i2+7];

         #pragma acc atomic update
         domain_m_fx[n0si2] += hgfx[0];
         #pragma acc atomic update
         domain_m_fy[n0si2] += hgfy[0];
         #pragma acc atomic update
         domain_m_fz[n0si2] += hgfz[0];

         . . .

         #pragma acc atomic update
         domain_m_fx[n7si2] += hgfx[7];
         #pragma acc atomic update
         domain_m_fy[n7si2] += hgfy[7];
         #pragma acc atomic update
         domain_m_fz[n7si2] += hgfz[7];
    }
    } // end parallel
    } // end data
}
```

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]

8: Understanding the detailed output of the OpenACC compiler

```
$ nvc -acc -Minfo -fast -gpu=cc80 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:
     75, FMA (fused multiply-add) instruction(s) generated
    133, Generating copy(domain_m_fz[:27000]) [if not already present]
         Generating copyin(domain_m_nodelist[:240000],gamma[:1][:],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
              Generated vector simd code for the loop containing reductions
```

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

7: 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     =        0.90 (s)
Grind time (us/z/c)  = 0.035675541 (per dom)  (0.035675541
overall)
FOM              =  28030.409 (z/s)


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

```
$ nvc -acc -fast -gpu=cc80 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     =        1.20 (s)
Grind time (us/z/c)  = 0.047847129 (per dom)  (0.047847129 overall)
FOM              =  20899.895 (z/s)


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

LULESHmk code runs correctly on the GPU @perlmutter using OpenACC offload
But runs slower because further optimizations are required (e.g. minimize data transfers)

Final remarks 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/

codee

www.codee.com

info@codee.com

Subscribe: codee.com/newsletter/

USA - Spain

codee_com

company/codee-com/