GPU HACKATHON

#gpucesgahack

29 mayo - 1 junio
Santiago de Compostela

TEAM 5

Daniel Otero Oubiña
Belén Torrente Torrente
Jaime González Cuevas

NICE TO MEET YOU :-)

A Coruña

OUR GOALS

Turn a case study into a success story.

Validate the Parallware Trainer against real applications.

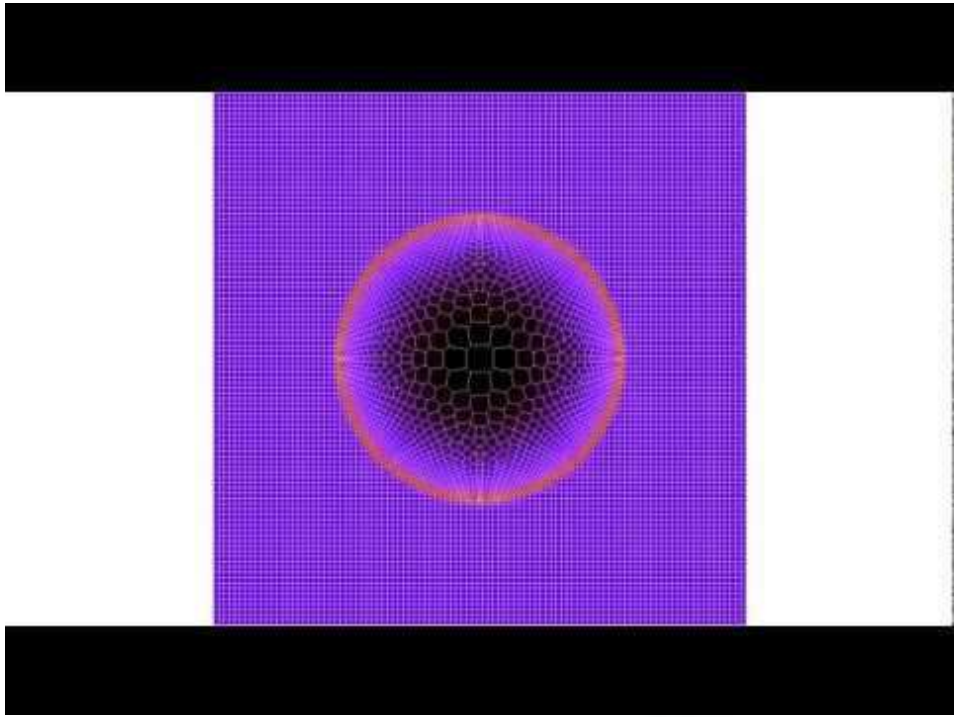
We have chosen LULESH.

LULESH: WHAT IS IT?

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



GPU HACKATHON

LULESH: WHY?

Reference benchmark in HPC facilities (part of CORAL)

Request from multiple institutions (ORNL, Livermore…)

An excellent real-life study case.

HACKATHON ROADMAP

LULESH parallel implementation as our top performance reference

Measure the scalability of the code

Study how the different parallel approaches behave (OpenMP and OpenACC mainly)

ANALYSIS (DRAFT)

1. Algorithmic features
2. Parallel design patterns
3. Implementation features

Algorithmic features

- Spatial discretization

  Main data structure: Domain

    Struct with all the 3D nodal data:

      coordinates, velocities, accelerations, forces, mass, connectivities, strain, energy, pressure, viscosity…

  Uses finite elements

Algorithmic features

- Type of solver
  - No convergence criteria: fixed number of iterations according to input / domain size.
  - Stencil pattern to calculate forces

- Compute-bounded and Memory-bounded

Parallel design patterns

- Three hotspots:
  - ApplyMaterialPropertiesForElems
    - Very complex calls, needs further analysis
  - CalcFBHourglassForceForElems
    - Parallel sparse reduction ← will look at this one first
  - CalcKinematicsForElems
    - Fully parallel

Implementation features

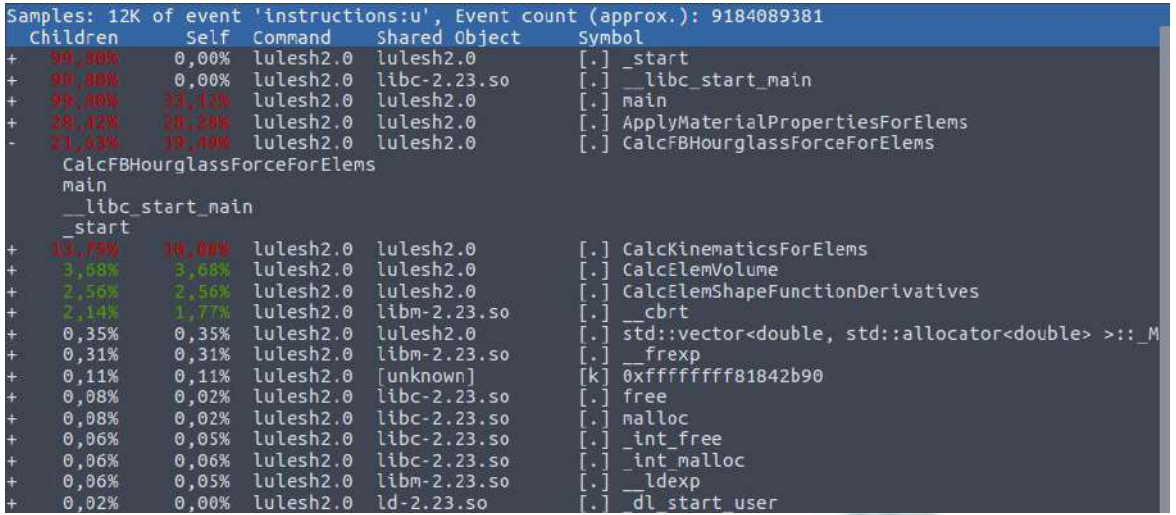
- Shape of the data structure:
  - Struct of arrays -> Array-based
    - Matrix represented as array
  - NO recursive data structs

Profiling

Used *perf* tool from Linux kernel

```
$ perf record -g ./<executable>

$ perf report -g graph
```



```
Samples: 12K of event 'instructions:u', Event count (approx.): 9184089381
  Children      Self  Command    Shared Object      Symbol
+   99,80%     0,00%  lulesh2.0  lulesh2.0          [.] _start
+   99,80%     0,00%  lulesh2.0  libc-2.23.so       [.] __libc_start_main
+   99,80%    34,15%  lulesh2.0  lulesh2.0          [.] main
+   28,42%    28,28%  lulesh2.0  lulesh2.0          [.] ApplyMaterialPropertiesForElems
-   21,63%    19,49%  lulesh2.0  lulesh2.0          [.] CalcFBHourglassForceForElems
     CalcFBHourglassForceForElems
     main
     __libc_start_main
     _start
+   11,73%    10,80%  lulesh2.0  lulesh2.0          [.] CalcKinematicsForElems
+    3,68%     3,68%  lulesh2.0  lulesh2.0          [.] CalcElemVolume
+    2,56%     2,56%  lulesh2.0  lulesh2.0          [.] CalcElemShapeFunctionDerivatives
+    2,14%     1,77%  lulesh2.0  libm-2.23.so       [.] __cbrt
+    0,35%     0,35%  lulesh2.0  lulesh2.0          [.] std::vector<double, std::allocator<double> >::_M
+    0,31%     0,31%  lulesh2.0  libm-2.23.so       [.] __frexp
+    0,11%     0,11%  lulesh2.0  [unknown]          [k] 0xffffffff81842b90
+    0,08%     0,02%  lulesh2.0  libc-2.23.so       [.] free
+    0,08%     0,02%  lulesh2.0  libc-2.23.so       [.] malloc
+    0,06%     0,05%  lulesh2.0  libc-2.23.so       [.] _int_free
+    0,06%     0,06%  lulesh2.0  libc-2.23.so       [.] _int_malloc
+    0,06%     0,05%  lulesh2.0  libm-2.23.so       [.] __ldexp
+    0,02%     0,00%  lulesh2.0  ld-2.23.so         [.] _dl_start_user
```

HourGlass function analysis

```
1 call for iteration

Problem size -> iterations (seconds each call)

    5 -> 72   (0.000120583)

    25 -> 752 (0.01339616)

    40 -> 1294 (0.055384797)
```

HourGlass force calculation pseudocode

```
for each element elem {
  for each neighbour n of elem  {
    hgfx[n] = … // Computation based on neighbour velocities on X axis
    hgfy[n] = … // Computation based on neighbour velocities on Y axis
    hgfz[n] = … // Computation based on neighbour velocities on Z axis
  }

  for each neighbour n of elem {
    fx[n] += hgfx[n];
    fy[n] += hgfy[n];
    fz[n] += hgfz[n];
  }
}
```

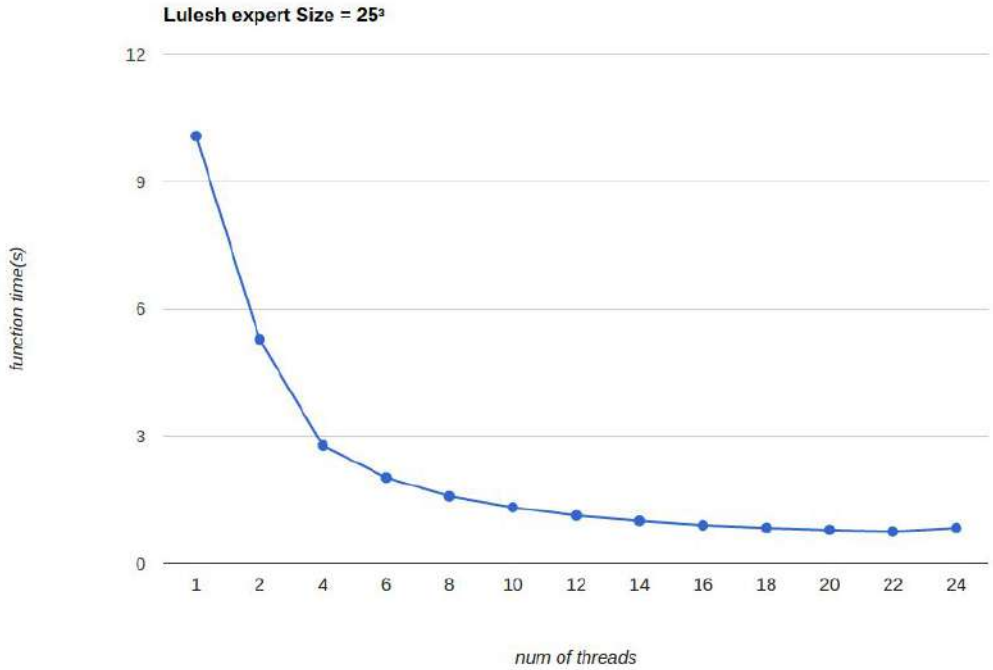
Lulesh reference implementation

Problem size: 15625 elements

Iterations: 752 iterations

Hardware: Intel Xeon E5 2680v3

2x 12-core processor



HourGlass force calculation pseudocode

```
#pragma omp for
for each element elem {
  for each neighbour n of elem  {
    hgfx[n] = … // Computation based on neighbour velocities on X axis
    hgfy[n] = … // Computation based on neighbour velocities on Y axis
    hgfz[n] = … // Computation based on neighbour velocities on Z axis
  }

  for each neighbour n of elem {
    #pragma omp atomic
    fx[n] += hgfx[n];
    #pragma omp atomic
    fy[n] += hgfy[n];
    #pragma omp atomic
    fz[n] += hgfz[n];
 }
}
```

Atomic CPU implementation (OpenMP)

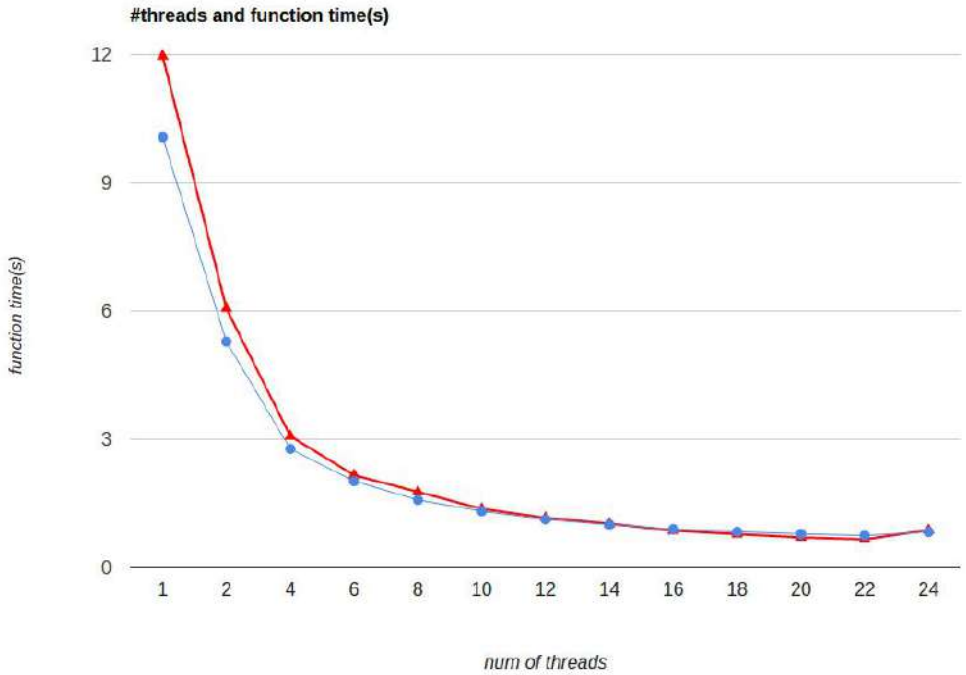
Problem size: 15625 elements

Iterations: 752 iterations

Hardware: Intel Xeon E5 2680v3

2x 12-core processor

~ 97% performance



GPU HACKATHON

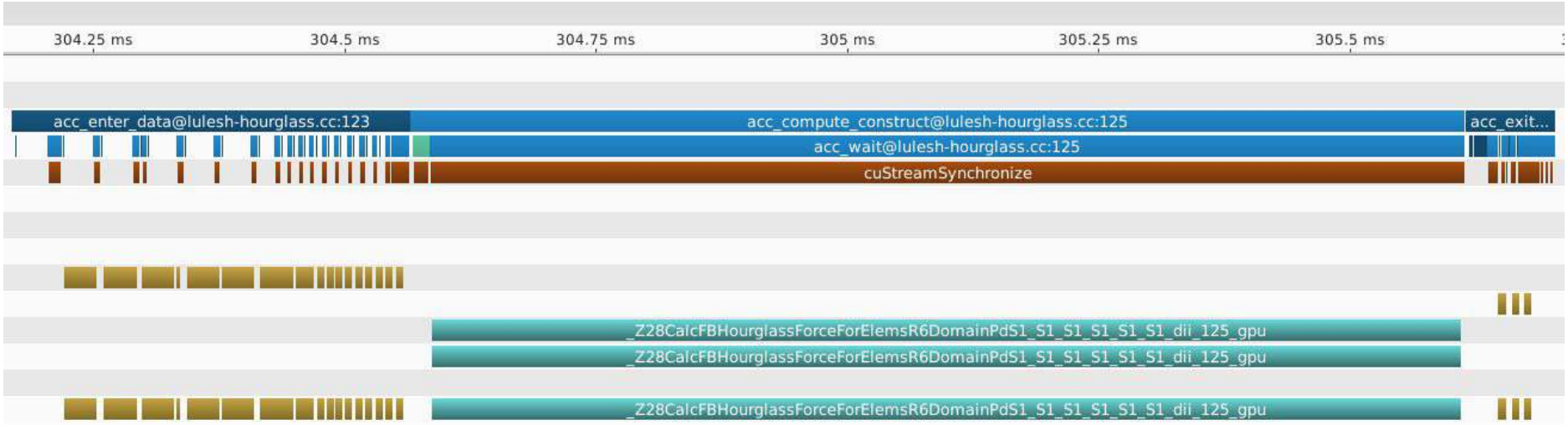
Atomic GPU implementation (OpenACC)

```
#pragma acc parallel loop copyin(...) copy(fx[],fy[],fz[])
for each element elem {
  for each neighbour n of elem  {
    hgfx[n] = … // Computation based on neighbour velocities on X axis
    hgfy[n] = … // Computation based on neighbour velocities on Y axis
    hgfz[n] = … // Computation based on neighbour velocities on Z axis
  }

  for each neighbour n of elem {
    #pragma acc atomic
    fx[n] += hgfx[n];
    #pragma acc atomic
    fy[n] += hgfy[n];
    #pragma acc atomic
    fz[n] += hgfz[n];
 }
}
```

OpenACC atomic profiling

Profiling for one iteration using PgProf



Enter data (396 µs)

Compute (1.049 ms)

Copy to CPU (89 µs)

Not too bad compute-to-transfer ratio

TODO List

- Measure execution times for OpenACC implementation
- Further study whether LULESH is compute-bounded or memory-bounded
  - Memory usage estimations
  - Tests with larger problem size
- Compare OpenMP and OpenACC versions
- Study remaining hotspot functions