GPU HACKATHON

#gpucesgahack

29 mayo - 1 junio
Santiago de Compostela

TEAM 1

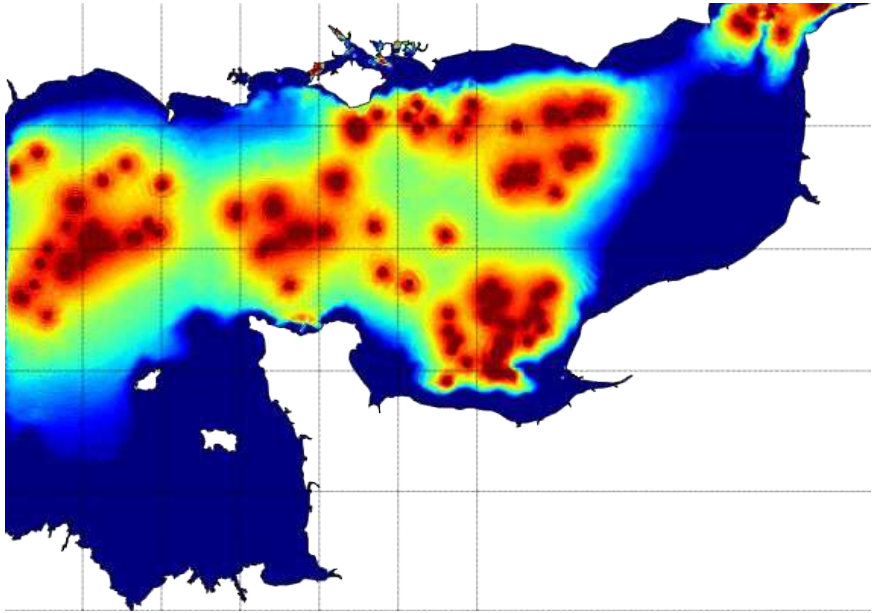
David Santos Domínguez
Ignacio Jáuregui Novo

Universidade Vigo

AtlantTIC

Bellhop 3D – Predicción de acústica submarina

Generar **mapa de ruido submarino** de la ría de Vigo

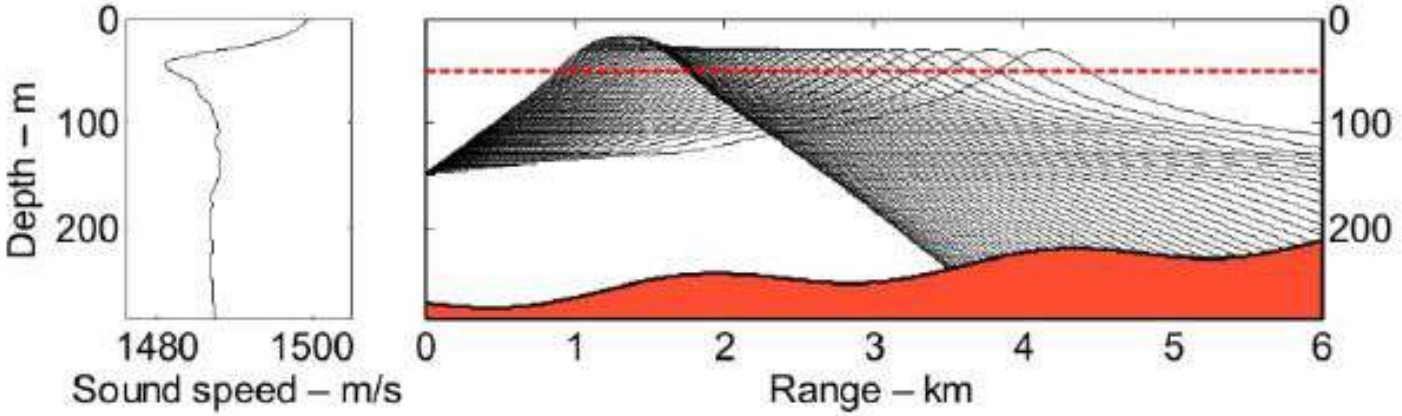


GPU HACKATHON

Bellhop 3D – Predicción de acústica submarina

Predicción acústica (nivel de presión sonora) basada en la técnica de **trazado de rayos**

Código Original de M. Porter.



Bellhop 3D – Predicción de acústica submarina

Aceleración/Paralelización:

- Validación y calibración estadística del método/software requiere de un número muy elevado de simulaciones
- Generar instantáneas en tiempos aceptables desde el punto de vista de experiencia usuario

GPU HACKATHON

Proof-of-concept of parallelism using Matlab

Using 20 cores / 40 threads and Matlab “parfor”
Times reduces linearly approximately

- Need of Montecarlo testing of some input parameters: lots of iterations
- Implementation of the workflow from matlab: matlab just makes independent calls to the Bellhop3D program pre-compiled
- Long simulations for each parameter, not GPU accelerated

ANALYSIS (DRAFT)

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

Algorithmic features

- Spatial discretization
  - Finite Element Ray
  - Number of sound sources. 3D Discretization of sound receivers
  - Beams of multiple rays in each step
- Type of solver
  - Direct for number of rays - First problem size
  - Iterative - ray steps,
    - Stopping criteria depends on: bounding box size, attenuation due to propagation and boundary conditions, MAXN number of iterations
  - Non Adaptive discretization of the domain
  - Dense computations (review sparse matrices?)
  - Based on solving systems of linear equations? Reduced delta matrix formulation?
- Compute-bounded: Yes, 3 levels of parallelism (number of rays, maximum number of steps x ray, size/construction of the beam
- Memory-bounded: Bathymetry file size + maximum number of ray steps

Parallel design patterns

```
 1 Azimut: DO ibeta=1:Nbeta {!FULLY PARRALELL LOOP
 2   Elevation: DO ialpha= 1:Nalpha {!FULLY PARRALELL LOOP
 3         REAL Amplitude=Ray_amplitude_calculation;
 4         ARRAY RaySegments[MaxN];
 5         Stepping: DO istep=1:MaxN OR stopCriteria=1 ! RECURRENCY
 6         {  !Solve differentia ray eq. based on Heun ( Runge-Kutta method)
 7              Struct T=f(params);
 8              RaySegments[istep+1]=f(RaySegments[istep]+f(T);
 9              if(reflection) !CALCULATE REFLECTIONS (check for reflection conditions)
10                   RaySegments[istep+2]=f(RaySegments[istep+1]+f(T);
11         }
12         P[Rtheta,Rdist,Rdepth] = f(RaySegments) !CALCULATE RECEPTOR PRESSURE

13   }
14 }
```

Implementation features

- Shape of the data structure?
  - Array-based
  - Array of structs (AoS)

Lots of variables, difficulties establishing the scope of each variable in Fortran

Problem Size

Depending on:

- Number of rays (azimuth,elevation)
- Number of sources
- Bathymetry size
- Number of rays segments (unpredictable)
- Number of frequencies=nf

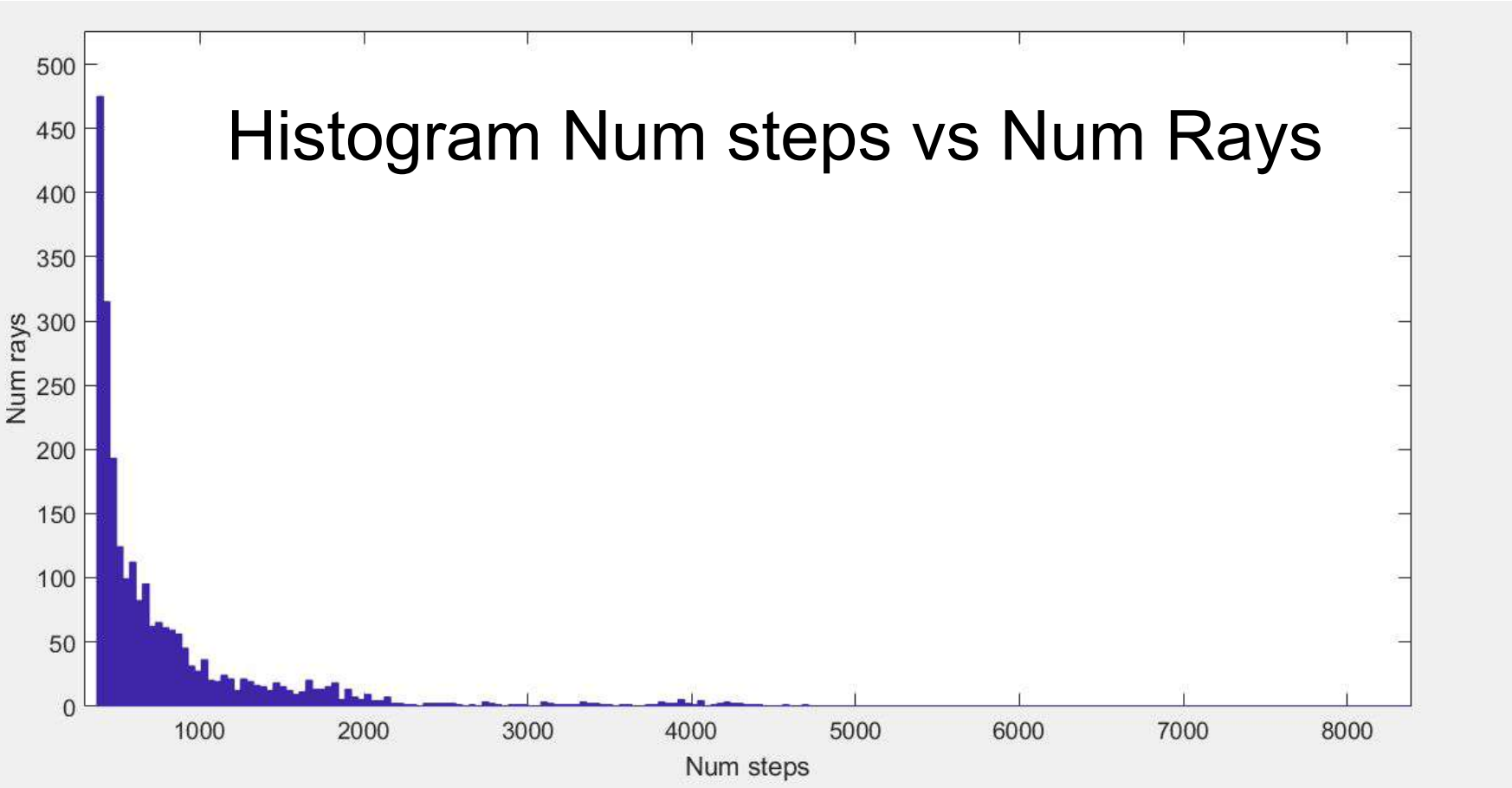
Minimum: 50x50 rays 1 source*nf=14s*nf
Maximum: ~=400*400 rays*Nsources*nf=50min*Nsources*nf
Reasonable: 400x20 should provide good numerical results

Problem size - loop times and segment size - 50x50 1source

```
Mean azimuth loop: 0.23s (50 rays)
Max azimuth loop:  0.55s
Min azimuth loop:  0.09s

Mean segment size: 969 steps
Max segment size: 99997(4 MaxN rays only)
Min segment size: 370

Without 4 MaxN rays  mean:929  max:4700
```



Current work

- Trying to parallelize the most external loop with omp
- The first approach is a failure>the program does not work properly
- Lots of variables with no clear scope > more investigation about the variables and their scope