GPU HACKATHON

#gpucesgahack

29 mayo - 1 junio
Santiago de Compostela

TEAM 6

IGNACIO VIDAL FRANCO

Ichthyop

- Ignacio Vidal Franco (ignacio.vfranco@gmail.com)
- Spanish Institute of Oceanography (A Coruña)
- BSc Physics
- MSc Computational Engineering and Design
- Aim: To understand the recruitment of sardine
- Tool: Lagrangian off-line simulation
- Dataset: 1 month hydrodynamic ROMS data (Atlantic coast)

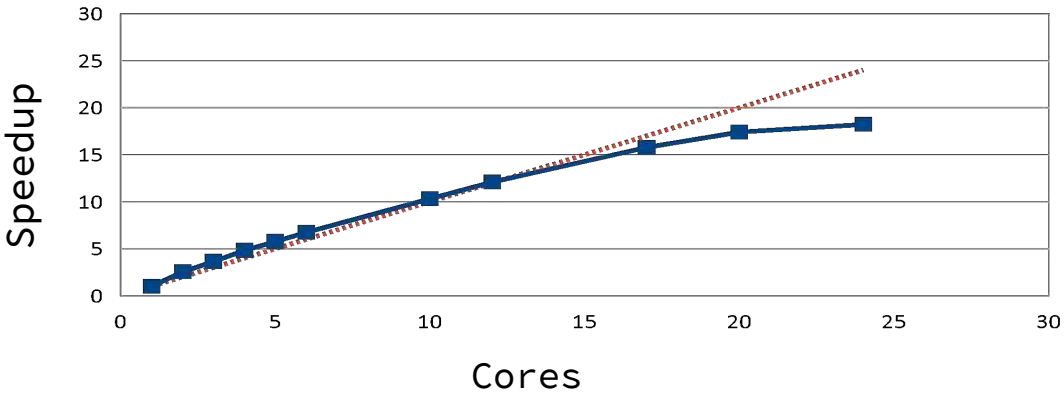
SARDINA

Sardina pilchardus

GPU HACKATHON

Software

- Ichthyop 3.1: A Lagrangian tool for simulating ichthyoplankton dynamics (Philippe Verley)
- Parallelized and tested with Java threads
- Speedup: ~17



Written in Java

GPU HACKATHON

Caveats and expectations

- Current execution time: 1 month is simulated in ~5h (running in Finisterrae 2)
- There is no communication between threads
- There are MPI libraries for Java
- Possibility to escape the JVM and write native code
- Downside: Portability is lost

EXPLORING JAVA NATIVE INTERFACE (JNI)

1. High potential for parallelization
2. Main problem: How to escape the Java Virtual Machine
3. “Hello World” (It worked once… I promise. But no anymore)

```
user@computer:~/ichthyop/ichthyop/classes$ java -jar -Dnp=2
-Djava.library.path=/home/user/ichthyop/ichthyop/classes/ ../dist/ichthyop.jar
Exception in thread "main" java.lang.UnsatisfiedLinkError:
org.previmer.ichthyop.action.NativeVDispAction.sayHello()V
       at org.previmer.ichthyop.action.NativeVDispAction.sayHello(Native Method)
       at org.previmer.ichthyop.ui.IchthyopApp.initLogging(IchthyopApp.java:72)
       at org.previmer.ichthyop.ui.IchthyopApp.main(IchthyopApp.java:98)
```

GPU HACKATHON

EXPLORING JAVA NATIVE INTERFACE (JNI)

1. Finally found the offender:

```
JNIEXPORT void JNICALL Java_org_previmer_ichthyop_action_NativeVDispAction_sayHello
```

vs

```
JNIEXPORT void JNICALL Java**ls**_org_previmer_ichthyop_action_NativeVDispAction_sayHello
```

Outside JVM

```
Hi. I’m written in C!

I’m printParticleCoordinates() X Coordinate: 138,677

Hi. I’m written in C!

I’m printParticleCoordinates() X Coordinate: 139,793

Hi. I’m written in C!

I’m printParticleCoordinates() X Coordinate: 142,534
```

Implementation features

- Main loop

```
public void step() {

    Iterator<IMasterParticle> iter = iterator();
    IMasterParticle particle;

    while (iter.hasNext()) {
        particle = iter.next();
        if (particle.isLiving()) {
            particle.step();
        }
    }
}
```

Outside JVM

- Now I have contiguous chunks of memory (well, I think…)

```
Native. Particle 19968: (140.18,124.965,26.7105)

Native. Particle 19969: (140.05,124.865,16.0581)

Native. Particle 19970: (140.045,124.981,24.0898)

Native. Particle 19971: (138.022,125.247,16.5045)
```

Implementing advectEuler() in C

```
for(i=0;i<nparts;i++) {

        vx[i] = vx[i]*dt;

        vy[i] = vy[i]*dt;

        vz[i] = vz[i]*dt;

        x[i]+=vx[i];

        y[i]+=vy[i];

        z[i]+=vz[i];

    }
```

Does it work correctly? Need to check...

Some results

NATIVE:

```
may 31, 2017 4:35:01 PM
INFO: Step 268 / 105120 - Time 2007/04/18 20:44:30
Native. Entering advectEuler()
Native. Leaving advectEuler()
may 31, 2017 4:35:01 PM
```

Estimated on CPU: ~30 min

JAVA:

Estimated on CPU: ~1h 30min

ANALYSIS

1. Algorithmic features
    a. Lagrangian particle model
2. Parallel design patterns
    a. Fully parallel loops
3. Implementation features
    a. Heavily Object Oriented Java code

GPU HACKATHON

Algorithmic features

- Spatial discretization? Only in input data
  - Finite differences? NO
  - Finite elements? NO
  - Finite volumes? NO
  - Type of elements: triangle, square? NOPE
- Type of solver? Lagrangian particle tracking
  - Direct vs Iterative (convergence criteria)? Iterative solver
  - Adaptive discretization of the domain? Only in input data
  - Sparse/Irregular computations (e.g. sparse matrices -CRS-)? No matrices are used, no BLAS. Only Calculus.
  - Based on solving systems of linear equations? No
- Compute-bounded? Heavily
- Memory-bounded? Hard to tell

Parallel design patterns

- Fully parallel loops? YES!
- Parallel scalar reductions? NO
- Parallel sparse reductions? NO

Pseudo-code

```
while(i<niter) {

    for(n=0;n<nparts;n++) { ← Fully parallel loop! Nparts = ~20000
        displacement_advection   = computeAdvection(n.data);
        displacement_Hdispersion = computeHdispersion(n.data);
        displacement_Vdispersion = computeVdispersion(n.data); ← That's my hotspot!
        …

        totaldisplacement =
            displacement_advection + displacement_Hdispersion +
            displacement_Vdispersion + …

        particleposition[n] += totaldisplacement;
    }
}
```

Serial optimisation

- Profiler shows that computeVdispersion() takes ~ 50% of processing power
- There is a lot of room for optimisation and deletion of redundant code that wastes computation time

Implementation features

- Shape of the data structure?
  - Pointer-based vs Array-based?
  - Recursive data structs (e.g. linked-list, tree)?
  - Array of structs (AoS) vs Struct of Arrays (AoS)? Arrays of Objects
- Moreover:
  - Object oriented code obfuscates the algorithm and data structures
  - JVM is a black box
  - Inherited classes that have objects that call methods that call methods that return objects that call methods that return arrays...

Roadmap (2 months)

- GPGPU parallelization is unfeasible within the allocated time window. This would imply a mix of Java and C and a major transformation in the code: From arrays of structs to structs of arrays
- Code is already parallel, implemented with Java threads
- Best effort/performance ratio: Hot-spot optimisation
- Goal: Simulation time under 3h with the same problem size (20000 particles). Current simulation time is 5h.
- Implementing the optimised non-GPU hotspot in C would be an option for further work

GPU HACKATHON

Conclusions

- Project time is restricted
- Potential for parallelization is high
- Heavily object oriented code style is not good for HPC (that's not because of Java. That's because the coding paradigm)
- Native implementation?
  - Seamless (or at least, not traumatic) blending of accelerated HPC code with current implementation
  - Need to talk with code maintainers in order to discuss effective approach
- Going native takes a big effort and obfuscates the code even more → It's better to start from scratch, using procedural programming (C)
- Code for Science!
- Code for HPC and hardware, not for humans

GPU HACKATHON