



Hartree Centre

Science & Technology Facilities Council

# DualSPHysics Performance on Xeon Phi

Sergi Siso

Application Performance Engineer

IPCC@Hartree, STFC Daresbury Laboratory

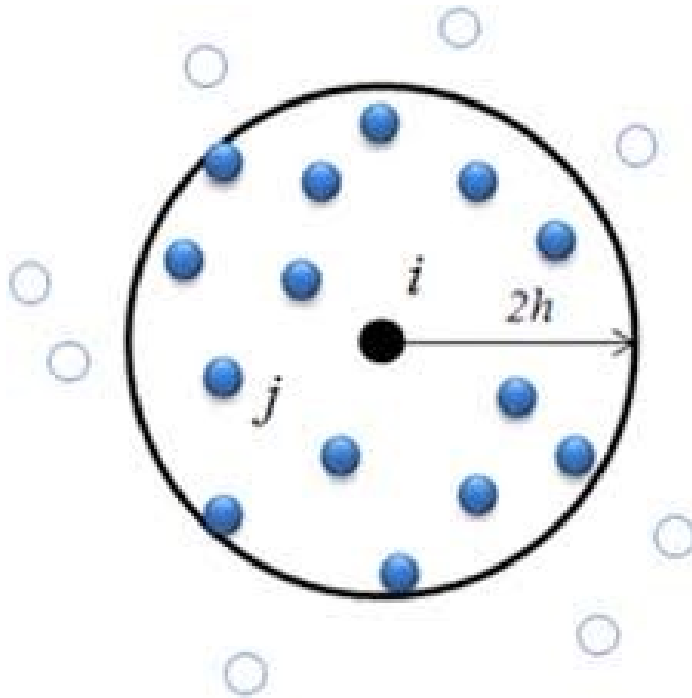
United Kingdom

## DualSPHysics

- **Smoothed Particle Hydrodynamics** numerical model developed to study free-surface flows where Eulerian methods can be difficult to apply, such as waves or impact of dam-breaks on off-shore structures.
- **DualSPHysics** is a set of C++, CUDA and Java codes designed to deal with real-life engineering problems.

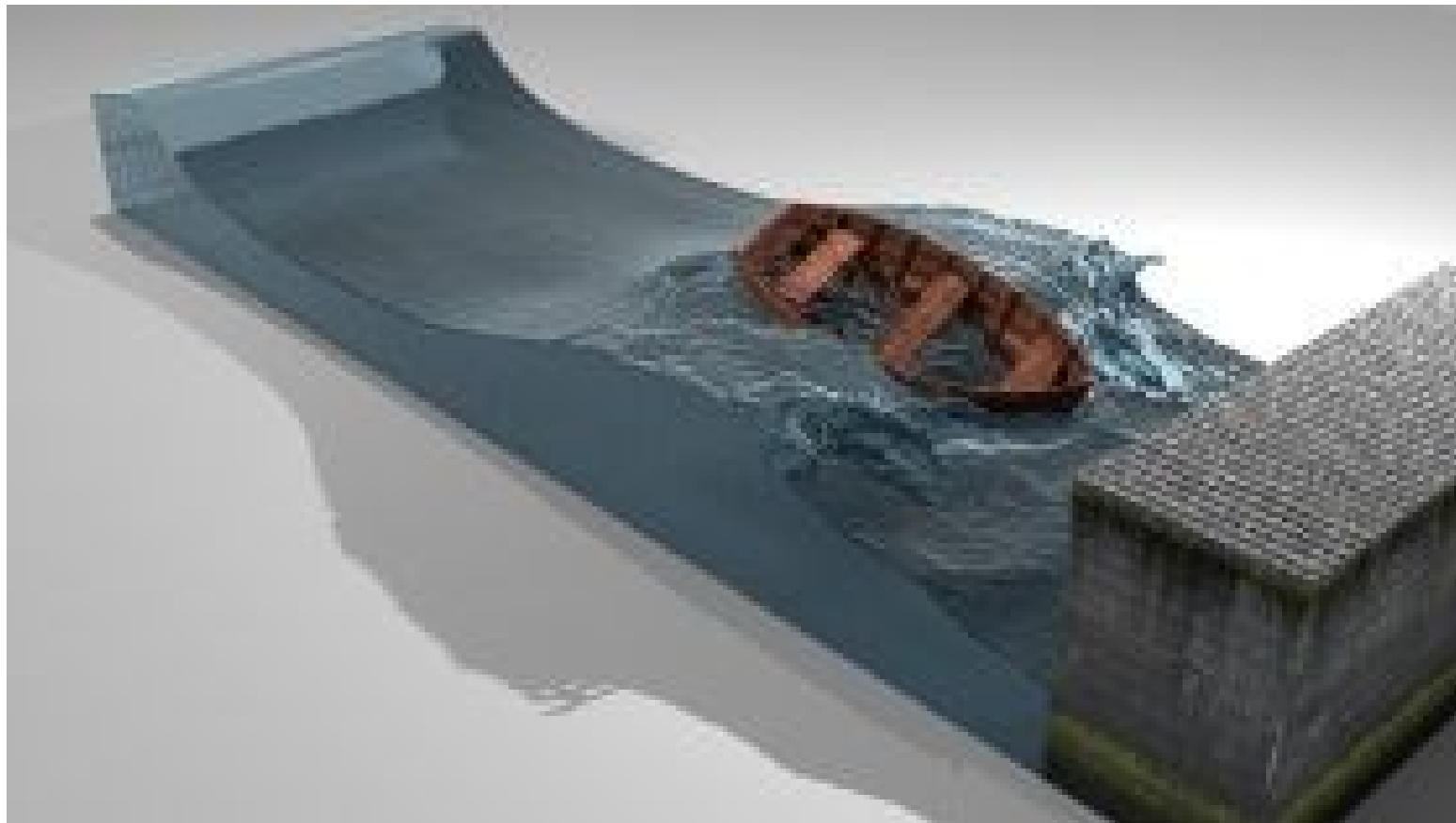


# Smoothed Particle Hydrodynamics

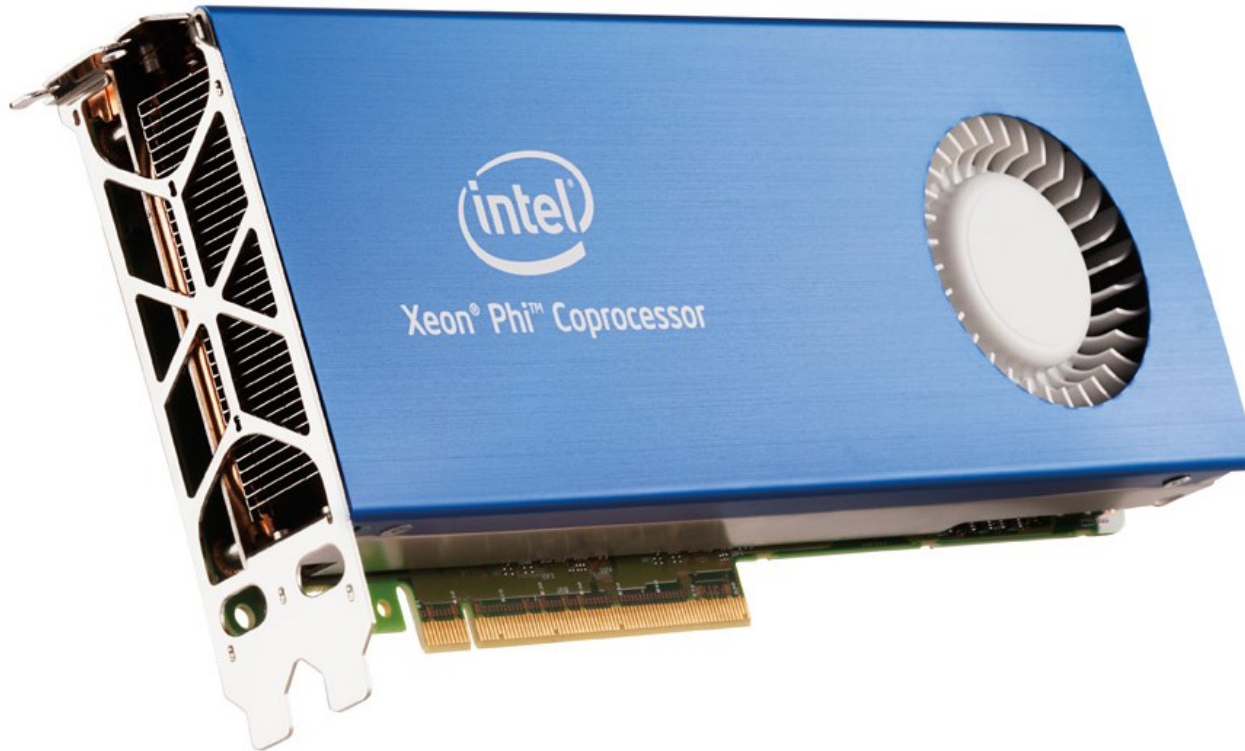


- Divides the fluid into a set of discrete elements.
- Particles have a spatial distance.
- Physical quantity of a particle is obtained through neighbour particles within a range.
- Contributions of each particle to a property are weighted according to their distance from the particle of interest.

# DualSPHysics

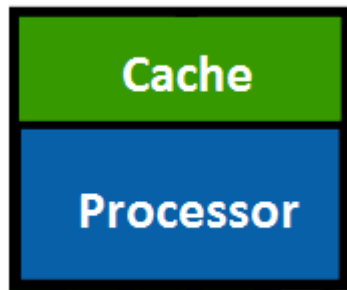


# Intel Xeon Phi



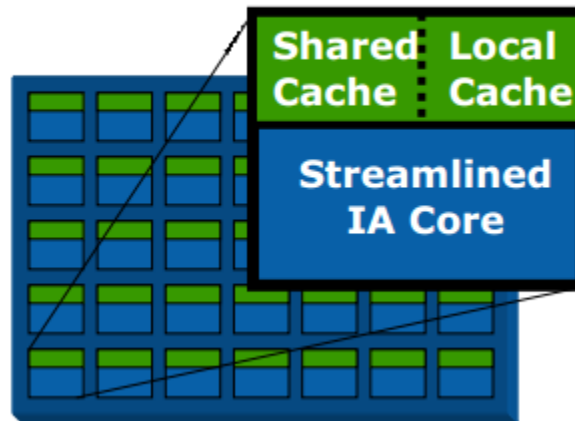
# Xeon Phi: KnC

## Single Core Processor



Long pipelined,  
out-of-order  
execution

## Many-core Processor



Short pipelined,  
cache coherent

## GPU



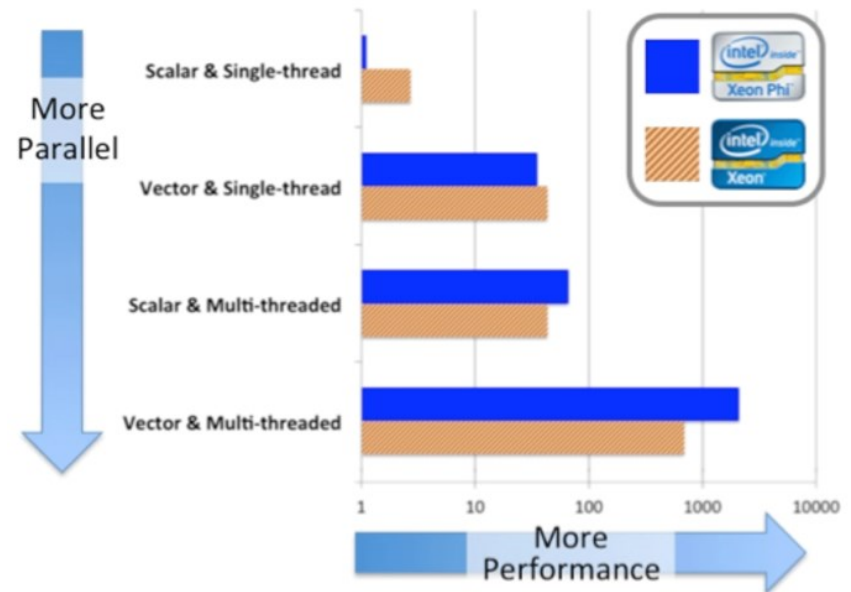
Shared instruction  
control, small cache

Complex serial cores  
Easy to code

Simple parallel cores  
Hard to code

# Xeon Phi: KnC

Xeon Phi 5110P	
Cores	60
Logical cores	240
Frequency	1.053GHz
GFLOPs	2,020
SIMD width	512 Bits
Memory	8GB
Memory B/W	320GB/s



## Initial investigation OpenMP version



Data Sorting and Reordering done serially, this gave better performance on CPU but did not take advantage of Phi resources

Damebreak 150k	2 Xeon E5-2697 (24 threads)	KnC (120 threads)
MakeSort	0.35	7.97
SortData	1.42	11.85
Preforces	0.56	2.22
Forces	33.92	22.12
Compute Step	0.13	0.44
Total Simulation	36.78	47.29

Force Computation was vectorized in Phi with compute efficiency 5.  
Not vectorized in CPU.



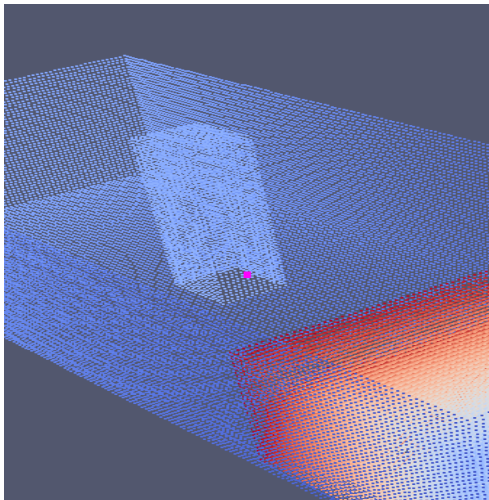
# Force Computation Vectorization

 Removes conditionals and type conversions from the inner loop and creates specialized implementations.  Not all of them Vectorized! But we just consider Verlet with Artificial Viscosity which it is.

```
template<typename T, bool lammps, T_pDeltaSph tdelta>
void JSphCpu::InteractionForcesFluid(...) {
    ...
    #pragma simd reduction(+:acep1x,acep1y,acep1z)
    for(int p1=int(pinit);p1<pfin;p1++){
        for(unsigned p2=pini;p2<pfin;p2++){
            drx=float(posp1.x-pos[p2].x); dry= ...
            const float rr2=drx*drx+dry*dry+drz*drz;
            if(rr2<=Fourh2 && rr2>=1e-18f){
                ... Masked Computation ...
            }
        }
    }
}
```

# Force Computation Arithmetic Optimizations

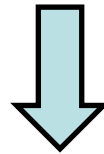
- Common sub-expressions elimination
- We can reduce the Floating Point IEEE restrictions...
  - Specially beneficial on the Phi architecture
  - Compilation flags: `-no-prec-div -fp-model fast=2`
  - 20% improvement on Phi Force Computation
- ... and the results are still good



## Sort Optimization: first attempt

Sorting the data in the CPU (latency optimized) and computing the forces in the Phi (bandwidth-compute optimized) was tried.

```
#ifdef MICOFFLOAD
  #pragma offload target(mic) in(pos[0:Np],velrhop[0:Np], press[0:Np],
    dcell[0:Np], beginendcell[0:Np]) inout(ar[0:Np],ace[0:Np])
  nocopy(tau,gradvel,delta) if(lamsps==false)
#endif
```



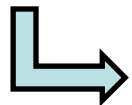
The overhead of transferring the data in each time-step suppress the gains of sorting in the CPU.

**Should find a Phi Native execution Solution!**

## Sort Optimization: second attempt

```
void JCellDivCpuSingle::PreSortFull(...){
    #pragma omp parallel for
    for(unsigned p=0;p<np;p++){
        .. Computes the cell of each particle ..
        #pragma omp atomic
        partsincell[box]++;
    }
}

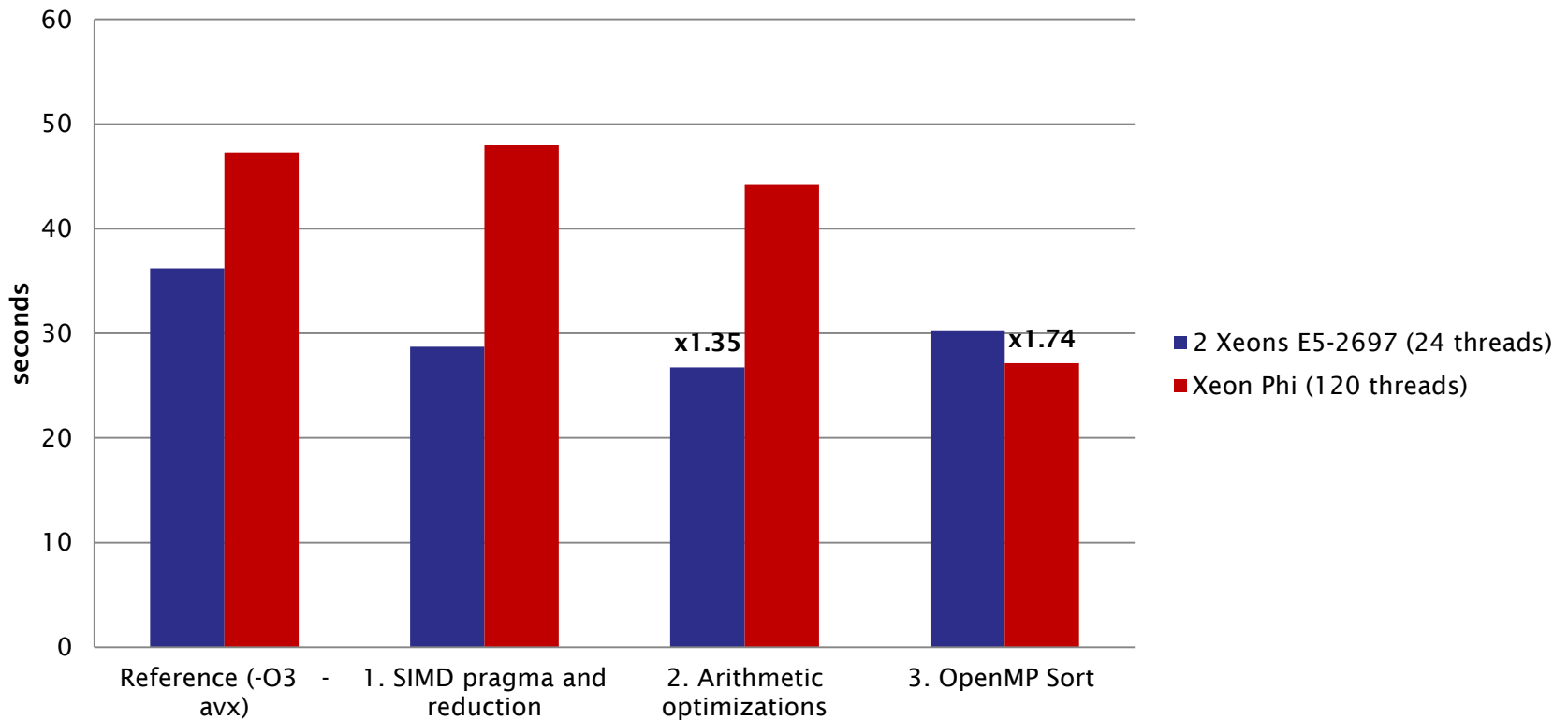
void JCellDivCpuSingle::MakeSortFull(..) // Still serial
void JCellDivCpu::SortArray(tfloat3 *vec){
    #pragma omp parallel for
    for(unsigned p=0;p<Nptot;p++)VSortFloat3[p]=vec[SortPart[p]];
    memcpy(vec,VSortFloat3,sizeof(tfloat3)*Nptot);
}
```



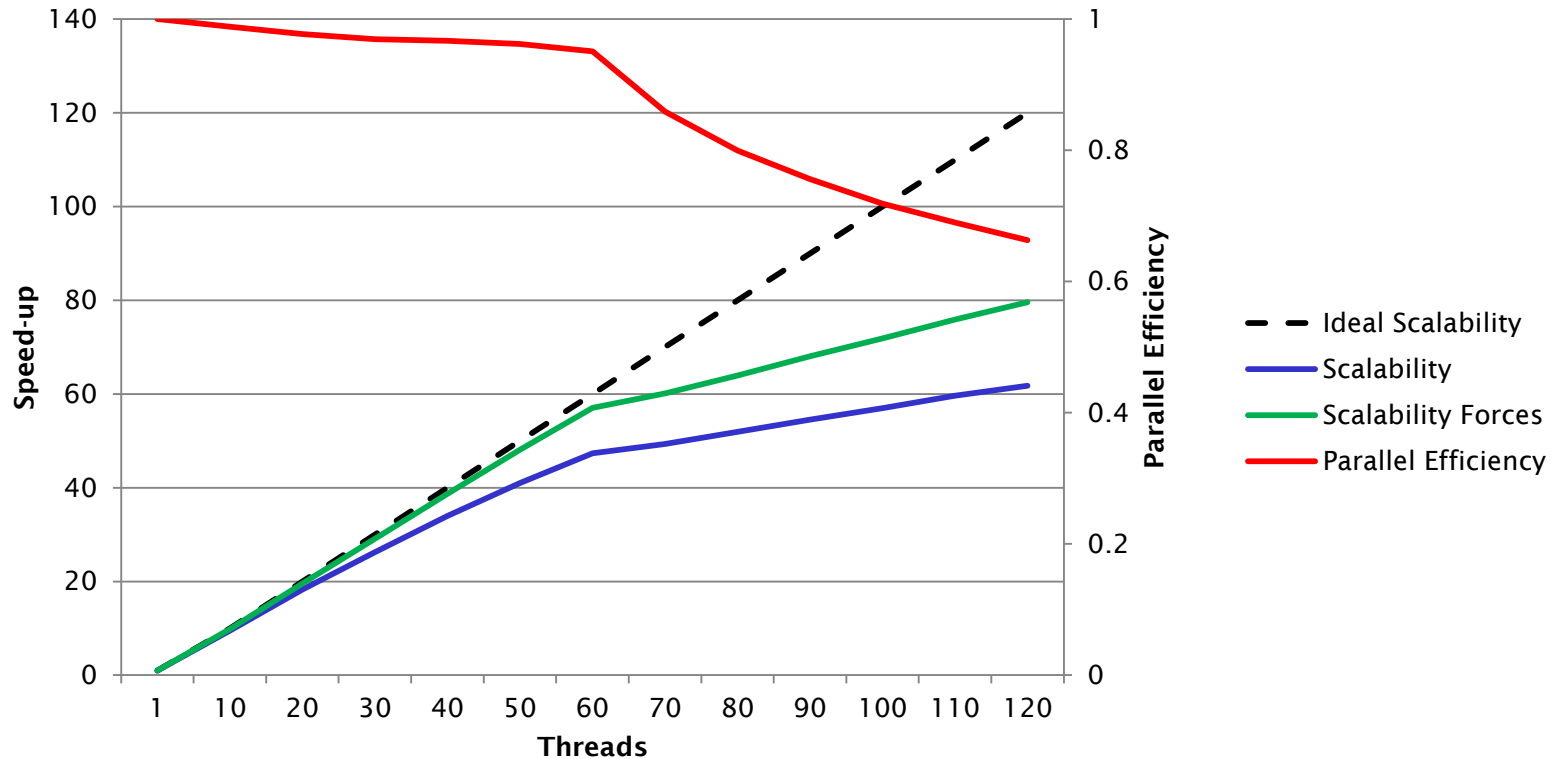
Simple parallel Sort implementation, but enough to reduce it to **11%** of the total simulation time. The focus is again in **the Force Computation (77%)**.

## Total Time

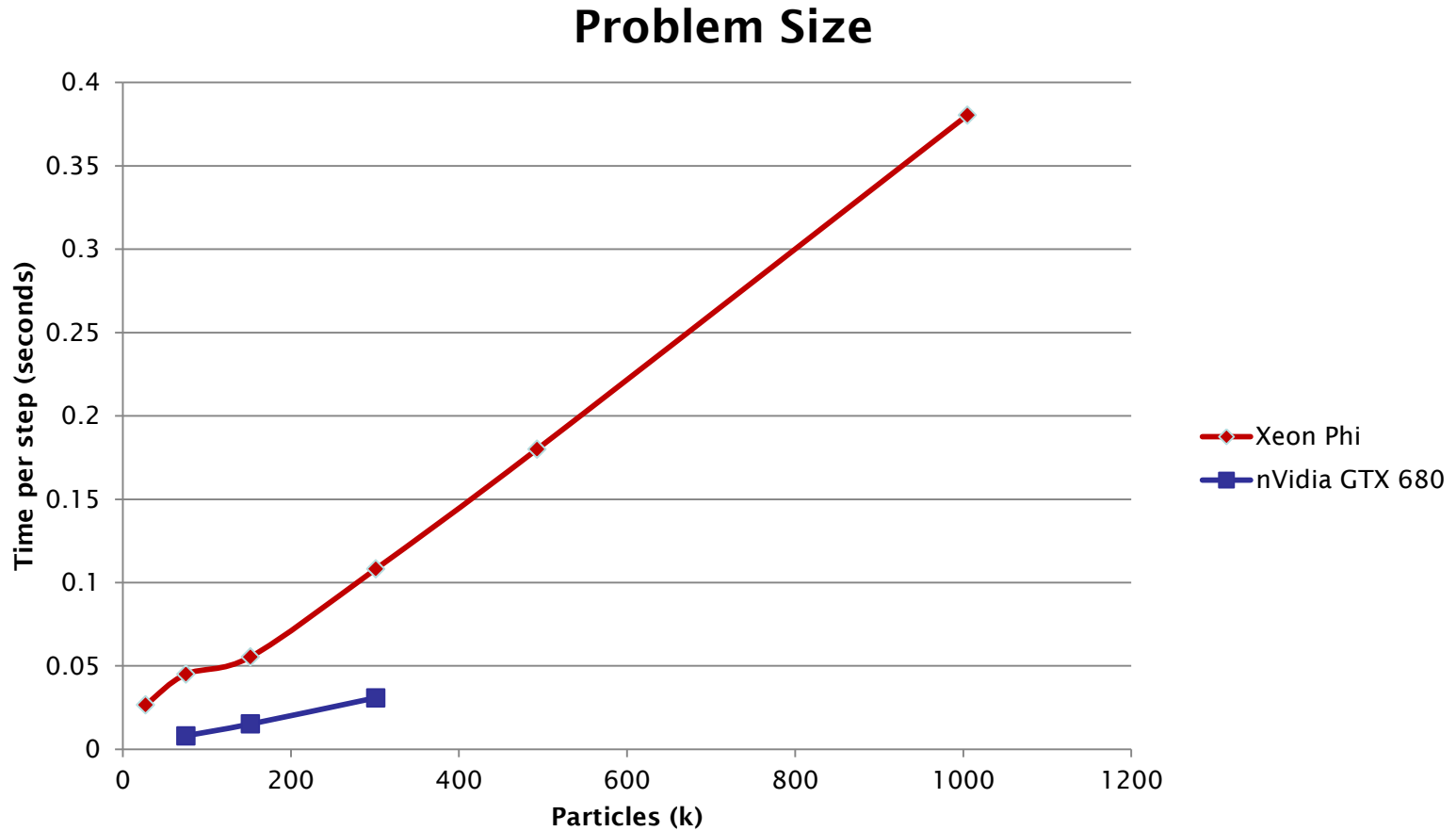
(CaseDambreak3D - 150k particles - Verlet - 0.05 sim real time)



## DualSPHysics Scalability on Phi



Thread scalability is good in both, Xeon and Xeon Phi.

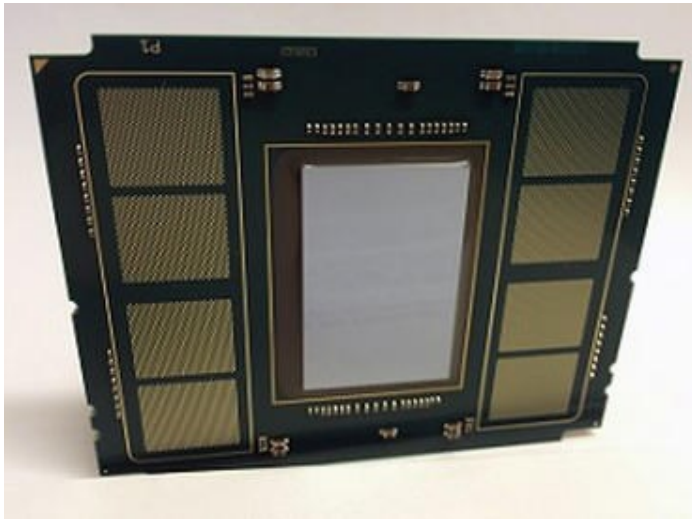


## Future Work

- A better implementation of Sort could be used. The one used in GPU (Thrust Library) seems a good candidate.
- Force updates of the symmetric particle can halve the computation work, but it does not scale due to the threads memory sharing. Memory locality techniques like loop tiling could be investigated.
- Current SIMD vectorization has unaligned load and peeled/remainder loops. Solving that could give significant improvements.
- Compromise between reordering or gathering particles as you need. Data preconditioning is a topic which is currently being highly discussed in IXPUG sessions.



## Future Work



### New Xeon Phi: Knight's Landing

- Host processor chip
- Binary compatible with Xeon
- Single performance x3 vs KnC
- 66-72 cores (1th x core can saturate the fpus -> Less threading!)
- Out of order execution
- MCDRAM (on-package memory)

## Conclusions

- Initial investigation of the DualSPHysics v4 on the Xeon Phi
  - Solved the main issues which caused poor performance.
  - Simple code modifications: OpenMP and SIMD pragma hints to the compiler.
  - CPU version was also improved during the porting
- Resulting Phi performance:
  - Equivalent to 2 Xeon Phi Ivy Bridge with 12 cores each
  - Not as far from GPUs as it use to be, ~3 times worse performance
- Potential for further improvements.