Massively Parallel Computing for Incompressible Smoothed Particle Hydrodynamics

Xiaohu Guo [a], Benedict D. Rogers [b], Peter. K. Stansby [b], Mike Ashworth [a]

[a] *Application Performance Engineering Group, Scientific Computing Department, Science and Technology Facilities Council, Daresbury Laboratory, Warrington WA4 4AD UK*

[b] *School of Mechanical, Aerospace and Civil Engineering, University of Manchester, Manchester, M13 9PL, UK*

Summary of the Project Progress

The incompressible smoothed particle hydrodynamics (ISPH) method with projection based pressure correction has been shown to be highly accurate and stable for internal flows.

We have proposed a focused effort to optimize an efficient fluid solver using the incompressible smoothed particle hydrodynamics (ISPH) method for violent free-surface flows on offshore and coastal structures. In ISPH, previous benchmarks showed that the simulation costs are dominated by the neighbour searching algorithm and the pressure Poisson equation(PPE) solver. In parallelisation, a Hilbert space filling curve(HSFC) and the Zoltan[5] package have been used to perform domain decomposition where preservation of the spatial locality is critical for the performance of neighbour searching

The following list highlights the major developments:

- The map kernel which provide the functionality of sending particles and their physical field data blocks to an appropriate partition have been rewritten and optimised with MPI one-sided communications and Zoltan distributed directory utility. The percentage of the map kernel is now reduced to less than 20% for those simulations with particles distributed evenly across the domain. For non-evenly distributed particles, using only non-empty cells will be part of future work.
- The nearest neighbour searching kernel has been rewritten with the preconditioned dynamic vector approach and linked list approach. Several optimisations have also been achieved during modification of the kernels:
  - A generic interface has been added so that it can be called by setting input parameters.
  - The inter-particle distance is always calculated rather than being stored, which has better cache performance and small memory footprint. The code is now 10 times faster when using small number of particles, and up to 3 times faster when using large number of particles.
  - The optimisation of smooth kernel functions to reduce memory footprint.

    - With above two modules, the memory footprint is much smaller than before.

- The cell reordering along the Hilbert space-filling curve have been implemented, but the performance improvements is not as large as expected, The main issue is the overhead of reorder itself. the overall performance improvement by reorder cells is less than 5%.

- The optimisation of the pressure Poison Equation (PPE solver).
    - The new Yale sparse matrix format have been replaced with PETSc CSR format, this has greatly reduced memory footprint.
    - The particles have been renumbered before assembling to the global matrix, as an result, insertions of values to the global matrix become local operations. The performance of the PPE solver have greatly improved. With multi-grid pre-conditioner in PETSc, the code can now scale well up to 8 thousands cores with up to 100 million particles.

Key words: Smoothed Particle Hydrodynamics; Poisson Equation; MPI; Space-filling curve; Sparse Linear Equation; PETSc; Neighbour List Searching

---

1 The ISPH dCSE project

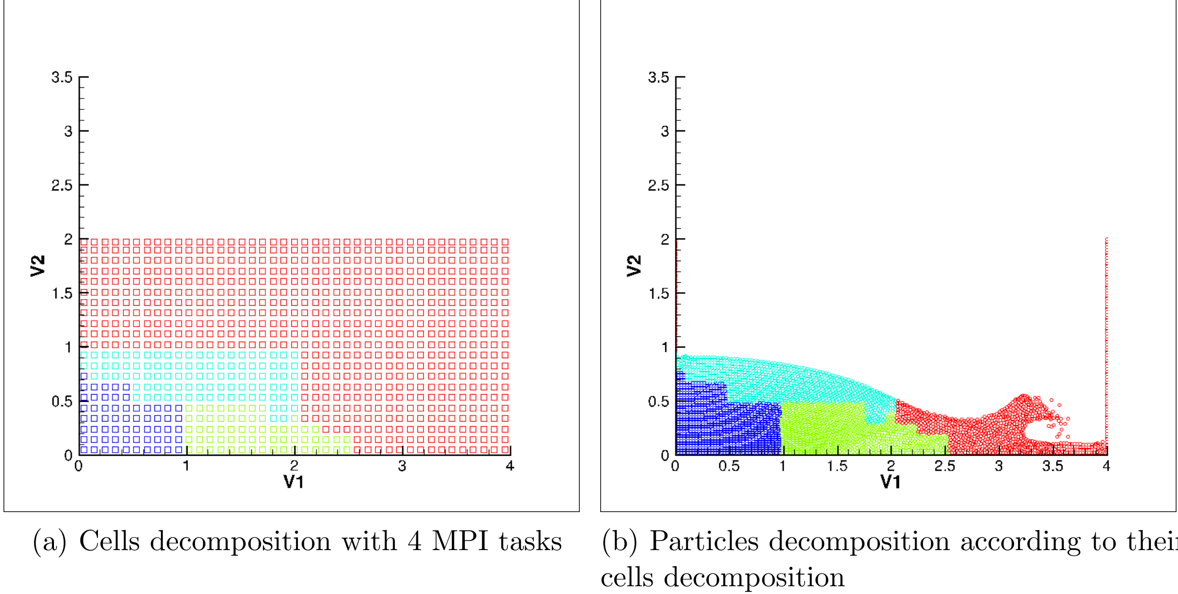
The ISPH software[1][2] has been in development for several years for tackling grand-challenge problems in coastal and offshore engineering, e.g. complex, generally highly nonlinear and distorted, wave motion, which may involve breaking, bore propagation, aeration, structure interaction and violent impact. The original version of the ISPH code is serial. Since August 2010, the current code has been developed jointly as part of an EPSRC-funded project between the Manchester SPH group and Daresbury Application Performance Engineering Group(APEG). The code has been rewritten with Fortran 90 and parallelised with MPI using Zoltan for domain decomposition and dynamic load balancing and PETSc for the sparse linear solver.

The remaining part of this report is organised as follows: In the next section we describe optimization of particle mapping kernel. Section 3 will address the neighbour list searching techniques and optimizations. Section 4 discusses how we optimise pressure Poisson solver with PETSc. Section 5 shows the final performance results, finally Section 6 contains a conclusion and discussion about further work.

2 WP1: Optimization of the particle mapping kernel

ISPH utilizes Zoltan [1] for domain decomposition. Previously, the particles were placed in cells which were then used to construct the neighbour list. Now, we use a Hilbert

[1] http://www.cs.sandia.gov/zoltan/Zoltan.html



(a) Cells decomposition with 4 MPI tasks

(b) Particles decomposition according to their cells decomposition

Fig. 1. Domain decomposition using HSFC with Dam Break test case

space-filling curve to decompose the cells which implicitly defines a data-to-processor assignment(see Fig. 1(a) and Fig. 1(b)). Mapping routines provide the functionality of sending particles and their physical field data blocks to an appropriate partition. Using the CrayPAT profiling on 1024 MPI tasks revealed that the main bottle neck is the use of MPI_GATHERV, consuming about 86% of the total MPI communication time, originating from the mapping functions. One of the main reasons is that this mapping function retains the map of the identification of an individual cell, cell_id, and partitions identification (ID) numbers globally. For highly violent flow cases, such as a dam break, the number of cells is normally much larger than the number of particles. For a large number of particles, storing this map as global shared property is inefficient. We propose two ways to tackle this issue.

2.1 *Optimize particle mapping kernel with MPI one-sided communication*

Mapping routines provide the functionality of sending particles and their physical field data blocks to an appropriate partition. Within each time step, we only update mapping subroutines once. The mappings are saved in cpid (see pseudo code Listing 1)

```
  !! Create window for cell partition id
  call MPI_Win_create(cpid, sizeof_win, disp_int, MPI_INFO_NULL
, MPI_COMM_WORLD, win, ierr)
  !! Create the new datatype for each MPI partition
  call MPI_Type_indexed(ncloca, blocklen, sorted_id(1:ncloca)
-1, MPI_INTEGER, ntype, ierr)
  call MPI_Type_commit(ntype, ierr)
```

```
    call MPI_Win_fence(0, win, ierr)

    do ip = 0, nproc - 1
       target_disp=0
       call MPI_Put(cell_parts, ncloca, MPI_INTEGER, ip,
  target_disp, 1, ntype, win, ierr)
    enddo

    call MPI_Win_fence(0, win, ierr)
    call MPI_Type_free(ntype, ierr)

    call MPI_Win_free(win, ierr)
```

Listing 1: mapping kernel pseudo code

Some notes on the MPI one-sided communication: all address arguments are of MPI defined type **MPI_Aint** in c and integer (kind = MPI_ADDRESS_KIND) in Fortran. MPI_Aint is defined to be an integer of the size needed to hold any valid address on the target architecture. Without using integer (kind=MPI_ADDRESS_KIND) in Fortran will result segmentation fault.

Using MPI derived type for MPI_PUT operation will help to reduce number of MPI_PUT operations per MPI Tasks.

2.2 Zoltan Distributed Directory

For those simulations where the number of cells is several times larger than number of particles, keeping and updating such a global mapping is very expensive. A distributed approach is more appropriate for such a mapping. Here, we propose using the distributed directory utility provided by Zoltan, which employs a rendezvous algorithm[3] to manage the cells locations after data migrations or to make information globally accessible. A distributed directory balances the load (in terms of memory and processing time) and avoids the bottleneck of a centralized mapping design.

However, since the Zoltan library does not offer a Fortran interface for the Distributed Directory, it has been necessary to create a Fortran interface. The following functions in Distributed Directory have now been provided in the Fortran interface:

- **Zoltan_DD_Create**: Allocates memory and initializes the directory
- **Zoltan_DD_Destroy**: Terminate the directory and frees its memory.
- **Zoltan_DD_Update**: Adds or updates GIDs' directory information.
- **Zoltan_DD_Find**: Returns GIDs' information (owner, local ID, etc.)
- **Zoltan_DD_Remove**: Eliminates selected GIDs from the directory.

Algorithm 1. ISPH Domain Decomposition and Dynamic load Balancing Algorithm with Zoltan Distributed Directory

```
Read in the particles data, calculate the domain size
call cell_generation() {constructing cells}
call get_cell_weight() {calculate number of particles in each cells}
call Zoltan_DD_Create(dd, ...) {initialize the directory}
for t = 1 → total_number_of_timesteps do
  call zoltan_partition(change) {Zoltan uses HSFC to decomposition cells, parameter change indicates whether there is partition changes}
  if change then
    call update_local_cells_gid()
    call Zoltan_DD_Update() {update the directory information after partition}
  end if
  call particle_migration()
  call Zoltan_DD_Find() in halo_update(){Find all required cells partition information from the directory}
  performing local calculation
end for
call Zoltan_DD_Destroy() {Terminate the directory and frees its memory}
```

Algorithm 1 shows the implementation details of Zoltan Distributed Directory together with domain decomposition and dynamic load balancing algorithm. We have to use a c pointer in order to pass around the Zoltan Distributed Directory object.

We have experimented the above approach with Dam Break test case using 55 million particles running with 2048 MPI tasks. The CrayPAT profiling results showed that the map kernel now uses under 10% **of the memory???** (see Fig. craypat MPI_Alltoall) with $2k$ cores.

3 WP2:Optimising neighbour list searching module

In general, no matter how fast or how highly parallelized the computing system is, it is evident that simulations with a large number of particles are only possible if an efficient neighbourhood search algorithm is employed. there are two methods typically used to construct neighbour lists: the first method is the cell-linked list (CLL) method which creates a list linked to every cell used to calculate the possible interactions among particles placed in the same and neighbour cells. The second method called a Verlet List (VL) creates a list linked to every particle. The ISPH code uses the CLL method as it requires less memory than VL method. The first step in the creation of a list is similar in both approaches since particles should be initially organized in cells. There are several approaches to allocate particles to cells to construct a neighbour list such as the linked list approach, which has similar performance and memory footprint to the preconditioned dynamic vector [4]. Therefore, we have rewritten

the neighbour list module with the preconditioned dynamic vector approach and linked list approach, Using the linked list approach offers the potential to provide the appropriate data structure for adaptive particle simulations (where particles are split and merge similar to adaptive mesh refinement, AMR) since memory pre-allocation for each particles neighbour list is particularly difficult when not using the linked list approach. The neighbour searching module with a preconditioned dynamic vector can be used for those non-adaptive SPH simulations with a moderate number of particles.

In the original code, in each time step, all particles distances are only calculated once and saved in three arrays, say drx, dry, rr2 in 2-D. The size of each array is about $num_of_particles * num_of_links$ which saves a lot of repeated floating operations, In the current development code, those three arrays are replaced with functions which have reduced memory footprint but have more repeated floating operations. Table 1 compared the serial performance between original code and the current code with 1 core using static water test case. The code is now 10 times faster when using small number of particles, and up to 3 times faster when using large number of particles.

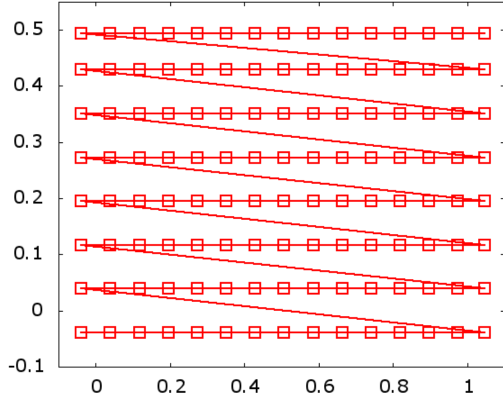
Table 1
The wall time comparison between original version and current version

Number of Particles	1257 (10 timestep)	125250(1 timestep)
The Original version (s)	0.96	191.6
The Current version (s)	0.091	63.76

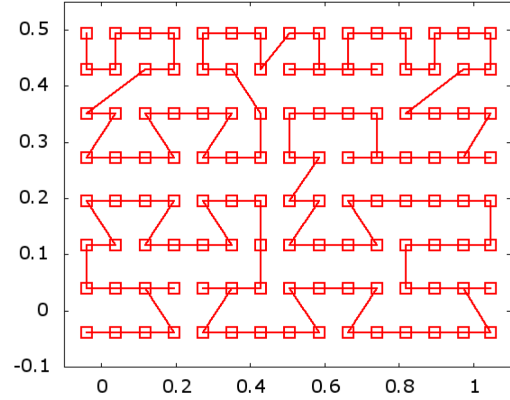
3.1 Reordering Particles

The original order of particles are normally stored following the order of initially given order, eg: wall particles, fluid particles, and mirror particles. The memory layout does not change during the simulation. Reordering particles reorganizes the data memory accesses so that one can load the cache with a small subset of a much larger data set. The idea is then to work on this block of data in cache. By using/reusing this data in cache we reduce the need to go to main memory (which is much slower than access to cache, and this also helps to reduce demands on memory bandwidth).

Since SPH neighbour searching is operated according to cells' neighbours, it is natural to consider ordering particles according to cells order. In the ISPH code, the cells have been ordered on the Space-filling curve(see Fig. 2), therefore the particles will also be reordered according to cells Space-filling curve order. As a result, the memory access will be changed every few timesteps due to reordering and particles moving. And these changes will improve the cache performance since HSFC keeps data locality.



(a) The original order of cells



(b) Cells ordered along the Space-filling curve

Fig. 2. The cells' original order v.s. the cells' HSFC order

4 WP3: Optimisation of pressure Poisson solver

Nearly 47% of the total computation is spent in the pressure Poisson solver in the current ISPH serial version. To solve the pressure Poisson equation efficiently, the PETSc software[6] has been employed within the ISPH code.

4.1 Vector Data type in PETSc

In PETSc, vectors are used to store discrete PDE solutions, right-hand sides for linear systems, etc. In parallel, the vectors can be created with the following functions

- VecCreate(MPI Comm comm,Vec *v);
- VecSetSizes(Vec v, int m, int M);
- VecSetFromOptions(Vec v);

In order to create user specified length of vectors, we replace **VecCreate** with **VecCreateMPI** for parallel or **VecCreateSeq** for serial. We use **VecSetValues** instead of **VecSetValue** to set an array of values into PETSc vector, which has better performance than **VecSetValue**.

4.2 Matrix Data type in PETSc

The serial version of ISPH used the new Yale format of sparse matrix, while the default matrix representation within PETSc is the general sparse AIJ format (also called the Yale sparse matrix format or compressed sparse row format, CSR). The matrix to be solved must be assembled for entry into the PETSc matrix. By default

the internal data representation for the AIJ formats employs zero-based indexing.

PETSc partitions matrices by continuous rows, while in the ISPH code, each row represents all neighbouring particles of a specific particle. The renumbering matrix has to be employed so that each partition can assemble its own matrix.

Suppose we are solving

$$AX = b \tag{1}$$

Let P be any permutation matrix, instead solve linear system 1, we solve the renumbered matrix:

$$PAP^T(PX) = Pb \tag{2}$$

With the HSFC, solving linear system (2) will have smaller bandwidth than directly solving linear system (1), this further helps to boost the performance of the PPE solver.

4.3 Preconditioner and solver

At the beginning of the ISPH project, the default solver option was using Jacobi as the preconditioner (**PCJACOBI**) and Stabilized version of BiConjugate Gradient Squared solver (**KSPBCGS**). With the improving the implementation of PETSc library for matrix solvers, we are now able to use a multi-grid preconditioners such as HYPRE BoomerAMG, PETSc GAMG, which has dramatically improved the convergence, robustness and the execution time. **BE SPECIFIC ...**

5 WP3: Final Benchmarking and Performance Analysis

The wet bed dam break and the static water have been used in this work as the benchmark test cases. The total number of particles used for benchmarking varies from 2 million up to 100 million. We are using UK National HPC platform HECToR, which is Cray XE6 system. It offers a total of 2816 XE6 compute nodes. Each compute node contains two AMD 2.3 GHz 16-core processors giving a total of $90,112$ cores; offering a theoretical peak performance of over 800 Tflops. There is presently 32 GB of main memory available per node, which is shared between its thirty-two cores, the total memory is 90 TB. The processors are connected with a high-bandwidth interconnect using Cray Gemini communication chips. The Gemini chips are arranged on a 3 dimensional torus.

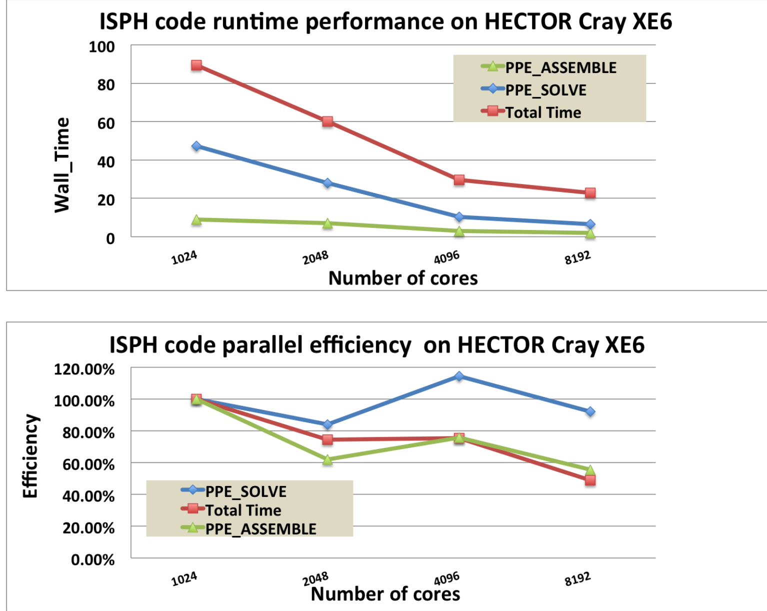


Fig. 3. ISPH Performance Comparison

The speedup S_p and efficiency E_p are obtained with the following formula:

$$S_p = T_1/T_p, \quad E_p = S_p/p \tag{3}$$

where T_1 is the wall time with 1 node, each node comprises 32 AMD Interlagos cores, T_p is the wall time with p nodes($p \geq 1$).

Table 2
Time Spent in Zoltan Partition

cores	32	64	128	256	512	1024
Time(s)	0.0625	0.0879	0.0715	0.0784	0.0952	0.1147

Table 2 gives the total timing of Zoltan domain decomposition with HSFC using 2 million particles with 32 cores up to 1024 cores. For 1024 cores, each time step is around $11s$ while Zoltan partition time is only about 1.01% for 1024 cores. For smaller number of cores, the percentage of Zoltan partition time is much smaller compare with total time.

Fig. 3 shows the ISPH solver run time and parallel efficiency using GMRES with the HYPRE BoomerAMG multi-grid preconditioner where the code has been benchmarked from 1024 to 8192 cores. The time spent in matrix assembly is very small compared with the time required for the solver. The average PPE solver parallel efficiency for ISPH is now well above 85% scaling from 1024 to 8192 cores showing that efficient large-scale ISPH simulations can now be run for real applications.

From Fig. 4, we can see that MPI still accounts for the 41% of the whole ISPH performance, the major cost $MPI_Allreduce$ comes from Zoltan. As there are empty

Fig. 4. CrayPAT Sample Profiling Statistic of ISPH with 2048 MPI Tasks

```
 Samp%  |   Samp  |  Imb.  |  Imb.  |Group
        |         |  Samp  | Samp%  | Function
        |         |        |        |  PE=HIDE
 100.0% | 28609.0 |     -- |     -- |Total
|------------------------------------------------------------------------------
|  40.8% | 11665.1 |     -- |     -- |MPI
||-----------------------------------------------------------------------------
||  22.9% |  6557.1 |  319.9 |   4.7% |MPI_Allreduce
||   9.0% |  2576.2 |   89.8 |   3.4% |mpi_alltoall
||   3.1% |   890.4 |   41.6 |   4.5% |mpi_scan
||   3.0% |   861.9 |  348.1 |  28.8% |mpi_bcast
||   1.2% |   350.6 | 1243.4 |  78.0% |mpi_waitall
||=============================================================================
|  36.7% | 10493.6 |     -- |     -- |ETC
||-----------------------------------------------------------------------------
||  13.1% |  3757.1 |  562.9 |  13.0% |hypre_ParVectorInnerProd
||  11.1% |  3161.4 | 3332.6 |  51.3% |hypre_ParCSRCommHandleDestroy
||   3.8% |  1084.0 | 4242.0 |  79.7% |hypre_BoomerAMGRelax
||   2.0% |   585.0 | 1361.0 |  70.0% |hypre_CSRMatrixMatvec
||   1.0% |   279.7 |  967.3 |  77.6% |PETScParseFortranArgs_Private
||=============================================================================
|  14.7% |  4194.0 |     -- |     -- |USER
||-----------------------------------------------------------------------------
||   4.0% |  1137.0 |  634.0 |  35.8% |__mirror_parts_generator_MOD_generate_mirror
||   1.8% |   523.8 |  895.2 |  63.1% |__cell_base_MOD_construct_cell_linked_list
||   1.5% |   439.3 |   99.7 |  18.5% |MAIN__
||   1.1% |   303.5 |   38.5 |  11.3% |__sph_kernel_MOD_kernel_calculation
||=============================================================================
|   6.7% |  1902.5 |     -- |     -- |PETSC
||-----------------------------------------------------------------------------
||   3.4% |   982.9 |  369.1 |  27.3% |PetscCommDuplicate
||   3.1% |   901.0 |  321.0 |  26.3% |PetscSetDisplay
|==============================================================================
```

cells with Dam Break test case, in future work, we think using non-empty cells as Zoltan objects will greatly reduce number of cells for domain decomposition and dynamic load balancing, and therefore improve the Zoltan performance. The PPE solver is now about 43.4%, the majority costs are within HYPRE BoomerAMG. Kernel calculation is now well under 2% as the result of the neighbour list searching optimization.

We have experimented the particles reordering along the cells' Space-filling curve with particles numbers from 2K to 14K. From Fig. 5, we can see that the performance is varying from 4.4% to 8.5% with different number of particles. We think the performance improvement is less than we expected. The reason is that the overhead of particles reordering. How to reduce the overhead of reordering itself is still being

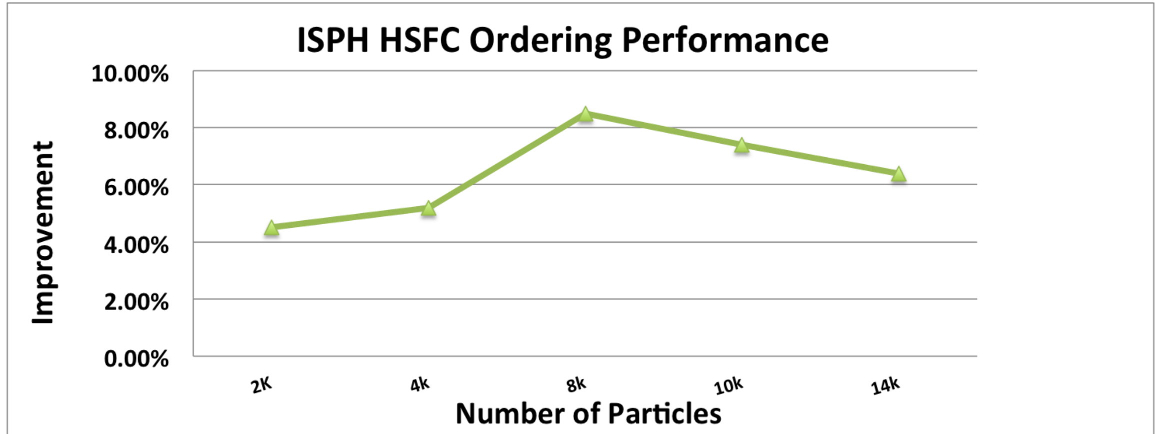


Fig. 5. ISPH total Wall-time improvement when reordered particles according to cells Space-filling curve order

investigated.

6 Summary and Conclusions

The map kernel which provide the functionality of sending particles and their physical field data blocks to an appropriate partition have been rewritten and optimised with MPI one-sided communications and Zoltan distributed directory utility. The percentage of the map kernel is now reduced to less than 20% for those simulations with particles distributed evenly across the domain. For non-evenly distributed particles, we need find a way to just use non-empty cells for the domain decomposition to reduce the cost of Zoltan, this will be part of future work.

The nearest neighbour searching kernel have been rewritten with the preconditioned dynamic vector approach and linked list approach, several optimizations makes significant improvement of the ISPH's serial performance. We have experimented with HSFC ordering, but the performance improvements is not very obvious, The main issue is the overhead of reorder itself. How to reduce the overhead of the ordering itself is ongoing work.

We have improved implementation of using PETSc solving PPE for ISPH, with HYPRE BoomerAMG as precondioner. the code can now scale well up to 8 thousands cores with up to 100 million particles.

We have also observed that I/O becomes the bottleneck when using more than 4k MPI tasks.

Acknowledgements

This project was funded under the HECToR Distributed Computational Science and Engineering (CSE) Service operated by NAG Ltd. HECToR A Research Councils UK High End Computing Service - is the UK's national supercomputing service, managed by EPSRC on behalf of the participating Research Councils. Its mission is to support capability science and engineering in UK academia. The HECToR supercomputers are managed by UoE HPCx Ltd and the CSE Support Service is provided by NAG Ltd. `http://www.hector.ac.uk`

Part of this work was presented at the 8th International SPHERIC SPH Workshop in Trondheim, June 2013.

References

[1] Lind, S.J., Xu, R., Stansby, P.K. and Rogers, B.D. *Incompressible Smoothed Particle Hydrodynamics for free surface flows: A generalised diffusion-based algorithm for stability and validations for impulsive flows and propagating waves*, *J. Comp. Phys*, 231:1499-1523, 2012.

[2] Guo X, Lind SJ, Rogers B D, Stansby P K, Ashworth M. *Efficient Massive Parallelisation for Incompressible Smoothed Particle Hydrodynamics with* 10^8 *Particles.* Proc. 8th International SPHERIC Workshop, Editor J.E. Olsen. pp 397-402. 4-6 June 2013

[3] A. Pinar and B. Hendrickson, *Communication Support for Adaptive Computation*, in Proc. SIAM Parallel Processing 2001

[4] J.M. Dominguez et al. *Neighbour lists in smoothed particle hydrodynamics*, Int. J. Numer. Meth. Fluids, 2010. DOI: 10.1002/fld.2481

[5] Erik Boman, Karen Devine, Lee Ann Fisk, Robert Heaphy, Bruce Hendrickson, Courtenay Vaughan, Umit Catalyurek, Doruk Bozdag, William Mitchell, James Teresco, *Zoltan 3.0: Parallel Partitioning, Load-balancing, and Data Management Services; Developer's Guide*, Sandia National Laboratories, 2007, Tech. Report SAND2007-4749W

[6] PETSc Manual Page, *http://www.mcs.anl.gov/petsc/documentation/index.html*