Efficient Massive Parallelisation for Incompressible Smoothed Particle Hydrodynamics with 10^8 Particles

Xiaohu Guo[a], Steven Lind[b], Benedict D. Rogers[c], Peter K. Stansby[c], Mike Ashworth[a]
[a]Scientific Computing Department, Science and Technology Facilities Council,
Daresbury Laboratory, Warrington, WA4 4AD, UK.
[b]School of Computing, Mathematics and Digital Technology, Manchester Metropolitan University,
Manchester, M1 5GD, UK.
[c] School of Mechanical, Aerospace and Civil Engineering, University of Manchester,
Manchester, M13 9PL, UK

***Abstract*—The incompressible smoothed particle hydrodynamics (ISPH) method with projection based pressure correction has been shown to be highly accurate and stable for internal flows. This paper describes an alternative parallel approach for domain decomposition and dynamic load balancing by using Hilbert space filling curve to decompose the cells with number of particles in each cell as the cells' weight functions. This approach can distribute particles evenly to MPI partitions without losing spatial locality which is critical for neighbour list searching. As a trade-off, the subdomain shapes become irregular. The unstructured communication mechanism has also been introduced to deal with halo exchange. Solving sparse linear equations for pressure Poisson equation is one of the most time consuming parts in ISPH using standard preconditioners and solvers from PETSc[1]. The particles are reordered so that insertions of values to the global matrix become local operations without incurring extra communications, which also have the benefit of reducing the bandwidth of coefficients matrix. The performance analysis and results showed the promising parallel efficiency.**

I. Introduction

Smoothed Particle Hydrodynamics(SPH) codes have been effectively parallelised using domain decomposition methods, implemented with libraries such as MPI for a long time. However, to efficiently maintain geometric locality of particles within processors and deal with dynamic load balancing arising from solving a complex, highly nonlinear and distorted flow is still hot topic. The highly scalable parallel performance relies on a good assignment of particles to processors and grouping physically close particles within a single processor reduces inter-processor communication.

There are several domain decomposition techniques used by the various existing MPI-based SPH fluid simulation models, The standard block partition method has been employed by SPhysics, DualSPhysics [1], which is easy to use, but handles poorly load balancing issues arising from solving highly complex, nonlinear and distorted flow.

[1]PETSc: http://www-unix.mcs.anl.gov/petsc/petsc-2/index.html

The particles' irregular distribution and complex-shaped computational domain prevent using the traditional simple domain decomposition methods with large number of particles due to load balancing issues. PPM [2] and SPH-Flow [3] have a implementation of Recursive Orthogonal Bisection, (ROB), In the ROB algorithm, the computational domain is first divided into two regions by a cutting plane orthogonal to one of the coordinate axes so that half the work load is in each of the sub-regions. The splitting direction is determined by computing in which coordinate direction the set of objects is most elongated, based upon the geometric locations of the objects. The sub-regions are then further divided by recursive application of the same splitting algorithm until the number of sub-regions equals the number of processors. However, the ROB method does not generally produce very fine granularity load balancing since it is based on cutting with a hyperplane.

This paper describes an alternative partitioning and dynamic load balancing approach by using Hilbert Space Filling Curve (HSFC) to decompose the cells with number of particles in each cell as the cells' weight functions. Comparing with ROB, this approach is less demanding computationally than ROB [4] and can distribute particles evenly to MPI partitions with fine granularity, and avoids losing spatial locality which is critical for neighbour list searching. As a trade-off, the subdomain shapes become irregular. The unstructured communication mechanism has also been introduced to deal with halo exchange.

Solving sparse linear equations for the pressure Poisson equation is one of the most time consuming parts in ISPH. The ISPH code uses standard preconditioners and solvers from PETSc. The particles are reordered so that insertions of values to global matrix become local operations without incurring extra communications, which also have a benefit of reducing the bandwidth of coefficients matrix.

This paper is organised as follows, Section II introduces the basic ISPH equations. Then sections III and IV describes the domain decomposition method and dynamic load balancing

method used in this code and how the code deal with various boundary conditions. the section V explains how we solve pressure Poisson equation with PETSc solver and the section VI presents the performance results and analysis. The last section VII are conclusions.

II. Basic incompressible Smoothed Particle Hydrodynamics (SPH) methodology

In the ISPH method, the Navier-Stokes equations in Lagrangian form, shown below, are solved. Incompressibility here is enforced in the projection method by a pressure Poisson equation [5].

$$\nabla \cdot u = 0 \tag{1}$$

$$\frac{du}{dt} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 u + f \tag{2}$$

In SPH, a variable A at a point $\mathbf{r}$ is approximated by a convolution product of the variable A with a smoothing kernel function $\omega_h(|\mathbf{r}-\mathbf{r}'|)$, with a smoothing length h, and is written as

$$A(\mathbf{r}) \approx \int_{\mathbf{\Omega}} \mathbf{A}(\mathbf{r}')\omega_{\mathbf{h}}(|\mathbf{r}-\mathbf{r}'|)\mathbf{dr}' \tag{3}$$

where Ω is the supporting domain. In a discretised format, the interpolation can be written as:

$$A(\mathbf{r_i}) \approx \sum \mathbf{V_j}\mathbf{A}(\mathbf{r_j})\omega_{\mathbf{h}}(\mathbf{r_{ij}}) \tag{4}$$

where V is the particle volume, $\mathbf{r_{ij}}$ is a distance vector between particle i and j. Hereafter $\omega_h(r_i j) = \omega_h(|r_i - r_j|)$ will be simply written as ω_{ij}. In this paper a quintic spline kernel is used for all cases. A smoothing length of $h = 1.3dx$ is typically used, where dx is the initial particle spacing.

The gradient operator for a general variable ϕ is given by:

$$\nabla\phi_i \simeq -\sum_j V_j(\phi_i - \phi_j)\nabla\omega_{ij} \tag{5}$$

This choice of gradient operator is preferred over others due to its increased accuracy when used in combination with the kernel gradient normalisation [6]. The expression for the kernel gradient normalisation is

$$\nabla W_(ij) = \mathbf{L}(\mathbf{r})\nabla\omega_{\mathbf{ij}} \tag{6}$$

where

$$\mathbf{L}(\mathbf{r}) = \begin{pmatrix} \sum V_j(x_j - x)\frac{\partial\omega_{ij}}{\partial x} & \sum V_j(x_j - x)\frac{\partial\omega_{ij}}{\partial y} \\ \sum V_j(y_j - y)\frac{\partial\omega_{ij}}{\partial x} & \sum V_j(y_j - y)\frac{\partial\omega_{ij}}{\partial y} \end{pmatrix}^{-1} \tag{7}$$

With these SPH spatial discretisations, the conservation of mass equation (1) SPH discretised form is:

$$\sum_j Vj(u_i - u_j)\cdot\nabla W_{ij} = 0 \tag{8}$$

While the discretised conservation of momentum equation (2) is,

$$\frac{du_i}{dt} = \frac{1}{\rho}\sum_j V_j(p_j - p_i)\nabla W_{ij} + \sum_j V_j \frac{2\mu\mathbf{r_{ij}}\cdot\nabla\omega_{\mathbf{ij}}}{\left(r_{ij}^2 + \eta^2\right)}\mathbf{u_{ij}} + \mathbf{f_i} \tag{9}$$

The pressure can be obtained from the pressure Poisson equation (PPE), written as:

$$\nabla\cdot\left(\frac{1}{\rho}\nabla p^{n+1}\right)_i = \frac{1}{\Delta t}\nabla\cdot\mathbf{u_i^\star} \tag{10}$$

The viscous term and Laplacian operators generally have two possible discretisations, see Morris et al. [7] and Schwaiger [8], they have both been implemented in the ISPH code [9].

III. Domain Decomposition and dynamic load balancing

Constructing neighbour lists and solving the pressure Poisson equation are two critical points in terms of the high performance of the Incompressible SPH codes. The parallel efficiency of the SPH software applications requires effectively mininizing communications between processors and good load balancing.

With these motivations, the ISPH code utilizes an alternative partitioning and dynamic load balancing approach by using Hilbert Space Filling Curve (HSFC) from Zotlan [10] to decompose the cells with number of particles in each cell as the cells' weight functions. The Zoltan library's toolkit provides various parallel partitioning algorithms, such as simple block partition methods, Recursive Coordinate Bisection(RCB), ROB, HSFC etc, which make ISPH code possible to switch between different partition algorithms for different applications. In the ISPH code, the particles were placed in cells which were then used to construct the neighbour list. We then use a Hilbert space-filling curve to decompose the cells which implicitly defines a data-to-processor assignment. Mapping routines provide the functionality of sending particles and their physical field data blocks to an appropriate partition. With this mechanism, the particles can be distributed evenly to MPI partitions, and in the mean time, without losing spatial locality which is critical for neighbour list searching.

The Zoltan Inverse Hilbert Space-Filling Curve functions [11] map a point in one, two or three dimensions into the interval $[0, 1]$, we assigning a weight for each cell defined as

$$wgts(i) = \frac{number\_of\_particles\_in\_cell\_i}{total\_number\_of\_particles} \tag{11}$$

The Zoltan HSFC partitioning algorithm seeks to divide the interval $[0, 1]$ into P intervals(P partitions) each containing the same weight of cells associated to these intervals by their inverse Hilbert coordinates. N bins are created (where $N > P$) to partition $[0, 1]$. The weights in each bin are summed across all processors. The algorithm sums the bins from left to right until the desired weight for current part interval is achieved.

This results in new partition of P intervals. This process is repeated as needed to improve partitioning tolerance.

Algorithm 1 ISPH Domain Decomposition and Dynamic load Balancing Algorithm

```
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}
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()
  end if
  call particle_migration()
  call halo_update()
  performing local calculation
end for
```

In the Zoltan implementation, The load on each processor is computed as the sum of the weights of objects it is assigned. The imbalance is then computed as the maximum load divided by the average load. An value for **IMBALANCE_TOL** of 1.2 indicates that 20% imbalance is OK; that is, the maximum over the average load should not exceed 1.2. If the tolerance ratio is over 1.2, then re-partitioning occurs and migrate the particles to their belonging partitions. The tolerance is setting in **Zoltan_Partition**

A. Halo Exchange

In the ISPH code, there are two major steps involving halo exchange, the first step is the projection step, the pressure and velocity fields are calculated and the particles' positions are advanced, the halo exchange are required before the pressure solver and a one-time pressure halo update after the pressure solver. At the second step, the particles are then shifted slightly and the hydrodynamic variables are corrected by a Taylor series in order to stabilise the simulations, this requires one halo exchange before the second step start since the entire particles' positions are changed during the projection step.

Due to a time step restriction(the CFL condition where a particle must not move more than fraction of its smooth length in one time step) each particle does not exhibit large spatial shifts and therefore the majority of particles tend to remain in the same cell. This ensures no requirement of re-partition during each time step, but halo exchange must be performed three times in each time step.

As the result of using above domain decomposition algorithm, each partition's sub-domain becomes irregular (see Fig. 1 (a) and (b)). In oder to construct the halo cells, we can't just use their coordinates as we are doing in block partition methods. Instead, we need setup a searching algorithm to identify halo cells, and construct the data structure to save halo information. We have designed non-structured communication plan specially for this type of problem. the detail of algorithm are list below:

Algorithm 2 ISPH Halo Exchange

```
for t = 1 → totalnumberoftimesteps do
  call halo_plan_setup(halo_sends, halo_recvs) {setup halo exchange plan}
  call halo_update_integer() {update halo objects integer ids}
  call halo_update_real() {update the physical eld data blocks to an appropriate partition}
  call halo_plan_destroy(halo_sends, halo_recvs) {destroy halo exchange plan if partition changed}
end for
```

Function **halo_update_** can be called anywhere in the same time step, and **halo_update_real** offers the capability to update both single real data block (needed by pressure halo update) and multiple real data blocks(velocity and pressure can be update together). Using multiple real data blocks **halo_update** has better scalability as all small messages are blocked together to send and receive. We also use non-blocking point-to-point communications instead of **MPI_ALLTOALLV** which will further improve the scalability.

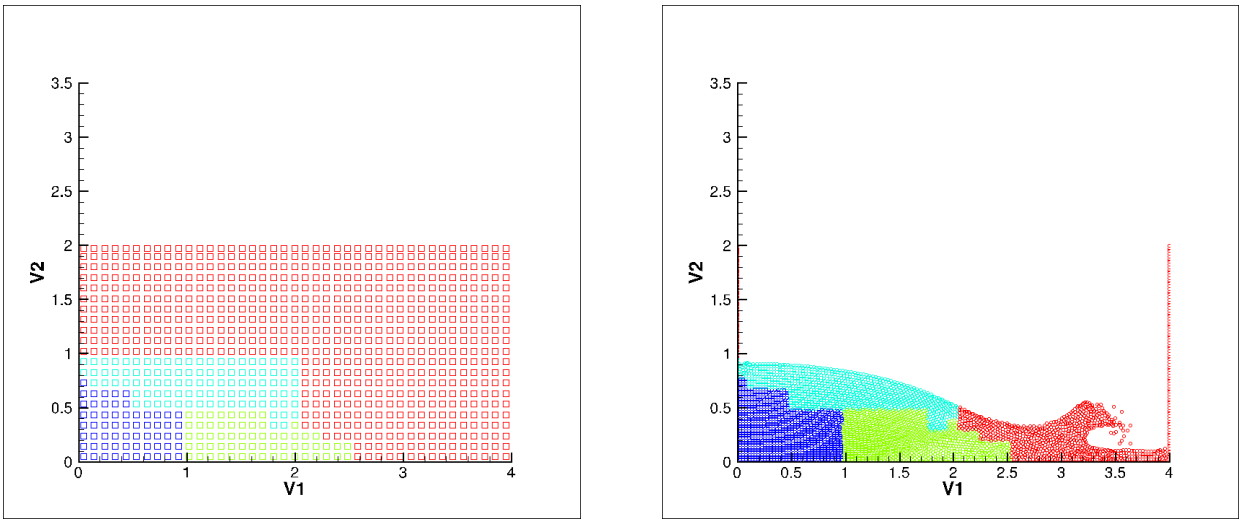
Algorithm 3 ISPH Halo plan setup

```
call halo_cell_sends(halo_sends) {calculate the cells id need to send}
call halo_cell_recvs(halo_recvs) {calculate the cells id need to receive}
call merge_cells {merge owned cells and received halo cells}
```

In each time step, the cells' positions are fixed, therefore the calculation of cells' id that need send and receive communications are only performed once. The halo plan is destroyed if the re-partitioning is needed. As cell's neighbour information are globally accessible (can be identified by their global index), there is no communication required during calculation of cells id to send and receive. In order to match the data between send side and receive side, both the sending side of cells list and receiving side of cells list has to be sequentialized by their global index.

B. Particles Data Management

For non-structured application, the data are saved according to their local index, the problems arise with arrangement of data between owned particles, halo particles and mirror particles. The current approach uses three counters to distinguish three different kind of particles, $nploca$, $nphalo$, $npmirror$, the particles' local index are less than $nploca$ are owned particles, any particles' local index are greater than $nploca$ and less than $nphalo$, are halo particles, the particles' local



(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

index are greater than $nphalo$ and less than $npmirror$ are mirror particles. This arrangement means generating mirror particles becomes a local process and each partition only needs to maintain the particles belonging to it; halo particles are then copied through halo exchange.

Maintaining particles belonging to a partitio is achieved through function $particles\_migration$ which involves two operations, the first operation is compressing the resident particles and the second operation is copying the entering particles. The particles exiting will be copied into the send buffer. As the number of leaving particles and entering particles changed during the simulation, it is not appropriate to preallocate the memory for this situation, the linked list data structure are used here to deal with such situation.

IV. Boundary Conditions

The current ISPH code supports various boundary conditions, such as wall boundary and mirror particle boundary.

A. mirror particles

The number of mirror particles depends on the number of particles near the domain boundaries, the number of mirror particles can change quite dramatically, therefore, it is almost impossible to precisely preallocate memory for mirror particles at each time step using array data structures.

In the serial ISPH version, mirror particles are generated with the following procedure:

- insert all the particles into cells according to original domain size
- identify which particles have mirror particles and generate their mirror particles
- regenerate the mirror cells according to the enlarged domain size
- reinsert the entire particles including mirror particles into cells

This approach has replicated computation of inserting particles into cells. If the same method is used in the parallel code, there are several difficulties have to overcome. First, we have to re-decompose cells; second, particle redistribution is necessary. In order to avoid these difficulties, the mirror particles' generation must be redesigned.

In parallel, we have the following assumption for the mirror particles generation

- mirror cells can be generated without any knowledge of number of mirror particles.
- The mirror particles can be generated independently in each partition.
- We assume the mirror particles have little influence on dynamic load balancing.

Algorithm 4 ISPH mirror particles generation

if have_mirrors = 1 **then**
  enlarge domain size
  generate cells according to enlarged domain size
end if
insert existed particles into cells
call halo_update(){this will make sure each mirror cells have complete mirror particles}
call mark_mirror_cells {mark mirror cells}
loop through all marked cells, all particles within certain distance in the marked cells should have mirror particles
loop all mirror particles, insert into their own cells.

The current method for mirror particles is to treat each partition separately. If we have mirror particles, the mirror cells will be added before time loop, the mirror particles start from $nphalos$. Here we assume the number of mirror particles has little influence of load balancing. Indeed, the mirror particles in the same mirror cells may be in a different partition. The

halo update before mirror particles generation will ensure each mirror cell has the complete list of its own mirror particles.

V. Solving pressure Poisson equation with PETSc

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, the PETSc software [12] has been employed within the ISPH code.

A. 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**.

B. Matrix Data type in PETSc

The serial version of ISPH use 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

$$AX = b \tag{12}$$

Let P be any permutation matrix, instead solve linear system 12, we solve the renumbered matrix:

$$PAP^T\,(PX) = Pb \tag{13}$$

With the HSFC, solving linear system (13) will have smaller bandwidth than directly solving linear system (12), this further helps to boost the performance of the PPE solver.

C. Preconditioner and solver

We are using jacobi as the preconditioner (**PCJACOBI**) and Stabilized version of BiConjugate Gradient Squared solver (**KSPBCGS**). PETSc also offers other preconditioners such as a multigrid preconditioner. The code setup interface can adapt easily to different preconditioners with different solvers, such as using (**PCJACOBI** with **KSPGMRES** etc.

TABLE I
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

VI. Performance Analysis and results

The wet bed dam break and the static water have been used in this paper as the benchmarking test cases. total number of particles used for benchmarking is 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.

The speedup are obtained with the following formula:

$$S_p = T_1/T_p \tag{14}$$

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 I gives the total timing of Zoltan domain decomposition with HSFC using 2 million particles with 32 cores upto 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.

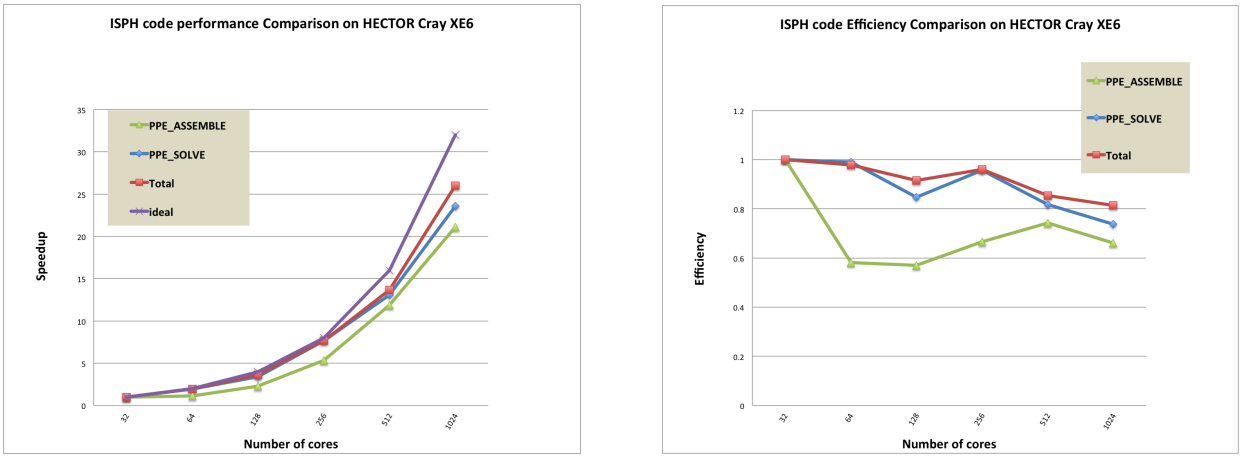
Fig. 2(a) and Fig. 2(b) gives the ISPH code parallel speedup and efficiency up to 1024 cores. We can see that solver performance is better than the matrix assembly. From 32 cores to 64 cores, matrix assembly's efficiency drop quickly because the current code use own new Yale format while PETSc use the Yale format, A matrix-to-matrix copy is necessary which creates a large memory footprint. We are currently examining the use of the Yale format to avoid matrix-to-matrix copy costs and therefore reduce the large memory footprint.

The solver, with 1024 cores, achieves a 73.7% efficiency. the overall efficiency is about 81.3% for 1024 cores which indicates there is still more work to be done in the other sections of the code, such as neighbour list searching and kernel calculations.

Although being a critical element of the overall performance, I/O is not considered in this paper and has been deactivated except for reading input during benchmarking for this paper.

VII. Conclusion

From the above results, we can see that domain decomposition with space filling curve can efficiently deal with irregular



(a) The Speedup (b) The Efficiency

Fig. 2. ISPH Speedup and Efficiency Comparison with static water test case

distributed particles. The method can are perfectly match the nature of non-uniform spatial distribution of SPH particles during simulations, which also offers capability of developing parallel adaptive SPH within an ISPH toolkit.

The code has been benchmarked on the UK Nation Supercomputing Platform HECToR, The percentage of time spent in Zoltan HSFC partition is almost negligible compare with the total time in each time step.

The initial benchmark results show that ISPH code can achieve nearly 81.3% efficiency using 1024 cores though there still more improve space such as, reducing memory footprint, optimisation neighbour list searching and kernel calculations.

ACKNOWLEDGMENT

The work was funded by the EPSRC Grant "*An incompressible smoothed particle hydrodynamics (ISPH) wave basin with structure interaction for fully nonlinear and extreme coastal waves*". Grant Number: EP/H018603/1, EP/H018638/1

The authors would also like to acknowledge the funding support 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.

REFERENCES

[1] Domnguez JM, Crespo AJC and Gmez-Gesteira M. *Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method. Computer Physics Communications*, 184(3): 617-627. 2013.

[2] F. Sbalzarini, J. H. Walther, M. Bergdorf, S. E. Hieber, E. M. Kotsalis, and P. Koumoutsakos. *PPM: A Highly Efficient Parallel Particle-Mesh Library for the Simulation of Continuum Systems*, *Journal of Computational Physics*, 215(2):566-588, 2006.

[3] G.Oger, E. Jacquin, M. Doring, P.-M. Guilcher, R. Dolbeau, P.-L. Cabelguen, L. Bertaux, D. Le Touz, B. Alessandri *Hybrid CPU-GPU acceleration of the 3-D parallel code SPH-Flow*, Proc. 5th International SPHERIC Workshop, Manchester, 394-492, 2010.

[4] John R. Pilkington, Scott B. Baden *Partioning with spacefilling curves*, CSE Technical Report, No: CS94-349,, Department of Computer Science and Engineering, Univeristy of California, San Diego, March, 1994

[5] A.J. Chorin, *Numerical solution of the Navier Stokes equations*, J. Math. Comput. 22 (1968) 745762.

[6] G. Oger, M. Doring, B. Alessandrini, P. Ferrant, *An improved SPH method: towards higher order convergence*, J. Comput. Phys. 225 (2007) 14721492.

[7] J.P. Morris, P.J. Fox, Y. Zhu, *Modelling low Reynolds number incompressible flows using SPH*, J. Comput. Phys. 136 (1997) 214226.

[8] H.F. Schwaiger, *An implicit corrected SPH formulation for thermal diffusion with linear free surface boundary conditions*, Int. J. Numer. Methods Eng. 75 (2008) 647671.

[9] 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.

[10] Erik Boman, Karen Devine, Lee Ann Fisk, Robert Heaphy, Bruce Hendrickson, Vitus Leung, Courtenay Vaughan, Umit Catalyurek, Doruk Bozdag, William Mitchell, *Zoltan home page(http://www.cs.sandia.gov/Zoltan)*, 1999.

[11] 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

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