,

Developing highly scalable 3-D incompressible SPH

Xiaohu Guo[a], Benedict D. Rogers[b]

[a]*Application Performance Engineering Group,*
Scientific Computing Department,
Science and Technology Facilities Council,
Daresbury Laboratory, Warrington WA4 4AD UK
[b]*SPH Group,*
School of Mechanical, Aerospace and Civil Engineering,
University of Manchester, Manchester, M13 9PL, UK

Abstract

Recently, accurate incompressible SPH (ISPH) codes have become more popular now that the numerical stability can be ensured, particularly for free-surface flows. The parallel efficiency of the SPH software applications requires effectively minimising communications between processors and good load balancing. Constructing neighbour list and solving pressure Poisson equation are two critical points in terms of the high performance of parallelised ISPH codes.

With these motivations, this project proposed to implement an alternative partitioning and dynamic load balancing approach 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 consequence, the sub-domain shapes become irregular. Unstructured communication has also been introduced to deal with complicated patterns of MPI communication between the sub-domains. Solving sparse linear equations for pressure Poisson equation is one of the most time consuming parts in ISPH, ISPH uses multigrid preconditioners and solvers from the open-source library PETSc. The particles have been reordered so that insertions of values into the global matrix become local operations without incurring extra communications, which also has the benefit of reducing the bandwidth of coefficients matrix. We demonstrated that domain decomposition with a space filling curve can efficiently deal with irregularly distributed particles. The method can 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 performance analysis and results show the promising efficiency with 10K+ cores.

The following list highlights the major developments:

- Domain decomposition and dynamic load balancing kernels with 3D Hilbert space filling curve method using Zoltan Distributed directory utilities.
- Using bounding box approach to reduce the memory footprint of partition mapping kernel and also reduced the overhead of Hilbert SFC itself because the number of rules of traversing along the SFC has increased from 4 in 2-D to 12 rules in 3-D, which may cause large overheads in SFC domain decomposition if the number of domain decomposition objects are huge.
- In the halo exchange kernels, we have optimised nbormap() kernel so that it can automatically switch between 2D and 3D(number of neighbour cells are different between 2D and 3D).
- All the above optimisations have been verified and validated with still water and 3D dry dam break test cases with up to 100 million particles
- The neighbour list searching kernels using preconditioned dynamic vector approach and the linked list approach have been implemented. These implementations particular solved the added difficulties in parallel like the memory operations for the unknown number of particles due to particles movement and dynamic generated particles for mirror boundary conditions.

- The particles reordering subroutines has been implemented according to the cells space filling curve order. This current offers 5% $\sim$ 8% performance improvements though further optimisation need to be done to reduce the overhead of space filling curve ordering.
- The new Yale sparse matrix format have been replaced with PETSc CSR format(AIJ and MPIAIJ) which reduces the memory footprint and avoids the expensive memory copy.
- The matrix assembly kernels using Morris and Schwaiger have been re-engineered to reduce accesses to PETSc memory space and SPH kernel calculations, eg: fusion loops of matrix assembly and right hand side which can reduce expensive redundant SPH smooth kernel calculations.
- The parallel I/O has been implemented using the H5hut. The implementation based on the unreleased version of H5hut-1.99-15rc1 provided by Dr. Achim Gsell from PSI, Switzerland.
  - This implementation has resolved the poor performance of I/O which dominated by dumping large amount of data generated with the large scale simulations which has to be run over many time steps.
  - The output file using H5hut can also be directly loaded in parallel by visualization tools like VisIt and ParaView which greatly facilitated the post-processing.

Keywords: Incompressbile SPH; Space Filling Curve; PETSc; Zoltan ; MPI;

---

1. The ISPH eCSE project

The major challenges caused by the increasing scale and complexity of the current petascale and the future exascale systems is imposing a strong evolutionary pressure on numerical algorithms and software ecosystem. Smoothed Particle Hydrodynamics(SPH) codes have been effectively parallelised using domain decomposition methods, implemented with libraries such as MPI for a long time. However, efficiently maintaining geometric locality of particles within processors with dynamic load balancing arising from solving a complex, highly nonlinear and distorted flow still requires substantial improvement.

The ISPH eCSE project mainly comprised of four work packages(refered to as WP1, WP2, WP3, WP4). WP1 is optimization the 3-D domain decomposition kernel and halo exchange kernel modifications from 2-D version of ISPH, WP2 is optimising 3-D neighbour list searching kernels, WP3 is optimisation of 3-D pressure Poisson solver. WP4 is Parallel I/O with H5hut.

The remaining part of this report is organised as follows: In the next section we describe the optimisation on the domain decomposition and dynamic load balancing methods and then the Section 3 describes the basic data structures of neighbour list searching and their optimisation. The Section 4 explains how we solve pressure Poisson equation with PETSc solver and the Section 5 presents the implementation of parallel I/O with H5hut. Finally, Section 7 are conclusions and the future work.

2. WP1: Optimisation the 3-D domain decomposition kernel and halo exchange kernel modifications from 2-D version of ISPH

Following from the 2-D optimisation for ISPH[5], for 3-D ISPH, we have continued to use the Hilbert Space Filling Curve(HSFC) method for domain decomposition and dynamic load balancing ISPH. The basic idea of domain decomposition along SFCs is to reduce a multidimensional domain decomposition problem to a one-dimensional splitting problem. This holds for both 2-D and 3-D domain decomposition. Fig. 1 shows the advantage of using SFC method to perform domain decomposition for 3-D complex fluid, the method we use here does not require extra effort to deal with trade-off between dynamic load balancing and data locality.

However, the number of rules of traversing along the SFC has increased from 4 in 2-D to 12 rules in 3-D, which may cause large overheads in SFC domain decomposition if the number of domain decomposition objects are huge. For those simulations where the number of cells have the same order magnitude with the number of particles, keeping

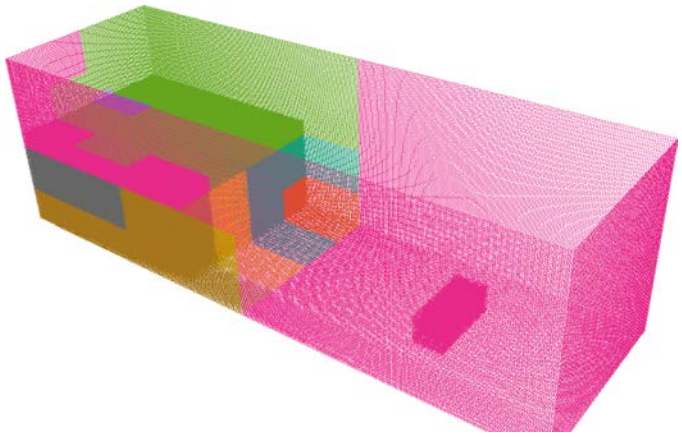


Figure 1: 3-D domain decomposition with HSFC with 8 MPI Tasks, different colours represent different partitions

and updating such a global mapping is very expensive. A minimal memory footprint approach is required for such a mapping. Hence, we employ a bounding box approach to manage the cells' locations after data migrations or to make information globally accessible. Alternatively, this bounding box approach can further reduce the memory footprint and improve MPI efficiency of mapping kernels. The number of bounding boxes will be determined by number of cells and number of partitions, hence, the size of bounding box should be small compared with the number of cells. Once the number of bounding boxes is fixed, we then use the number of particles in each bounding box as its weight function, and using SFC to decompose bounding box. This should be able to balance automatically the number of particles in each partition and also reduce the memory footprint of mapping kernels. The extra added benefit is keeping the run-time cost of the SFC domain decomposition kernel to minimum.

As the data layout used in 2-D and 3-D ISPH are very similar, the parallel data management layer can be shared between 2-D and 3-D. Though the number of neighbour cells increased from 8 in 2-D to 26 in 3-D, we can still use the common function *nbormap* (see **Pseudocode. 1**) to calculate each cell's neighbour cell's id once given the cell's global id. The parameter *nbors* can be switched between 2-D ($full2d\_nbors$) and 3-D ($full3d\_nbors$) which can be set at the beginning of the simulation. Although this approach has greatly facilitated the parallel data management for both 2-D and 3-D, it has one drawback. The **Pseudocode 1** only covers the situation that the cell with all the neighbouring indices exist and are actually neighbours. It cannot cover the two boundary cases, e.g.

- The returned list of **nbormap()** are in bounds but not an actual neighbours. This is harmless because none of the particles in those cells lie within the smoothing kernel cutoff. It wastes some computation but not enough to justify complicating the code to avoid those cells.
- The neighbour ids in the returned list of **nbormap()** are out of bounds. This causes illegal memory access. Fortunately, this is easy to detect and handle. If the returned neighbour id is outside the valid range of cell ids, the code skips that id and furthermore all subsequent neighbour ids will also be invalid (because the table orders them that way), so there is no need to check any subsequent ids in the list.

Pseudocode 1: Neighbour Cells Map Function

```
!! gid: cells global id
function nbormap(gid)
  integer(ikind), intent(in) :: gid
  !! nbors value could be full3d_nbors or full2d_nbors
  integer(ikind), dimension(nbors) :: nbormap
  !! total number of cells in xy plane
  integer(ikind) :: pxy

  !! ncx: number of cell in x direction
  !! ncy: number of cell in y direction
```

```
    pxy=ncx*ncy

    !! the bottom plane
    !! SW cell
    nbormap(1)=gid-ncx-1-pxy
    !! South cell
    nbormap(2)=gid-ncx-pxy
      ... ...
    !! NE cell
    nbormap(9)=gid+ncx+1-pxy

    !! the middle plane
    !! SW cell
    nbormap(10)=gid-ncx-1
      ... ...
    nbormap(17)=gid+ncx+1

    !! the top plane
    !! SW cell
    nbormap(18)=gid-ncx-1+pxy
      ... ...
    !! NE cell
    nbormap(26)=gid+ncx+1+pxy

end function nbormap
```

3. WP2: Optimising 3-D neighbour list searching kernels

3.1. Basic ISPH algorithm analysis

The basic algorithm of ISPH can be summarised in the Algorithm 1:

Algorithm 1: ISPH Basic Algorithm Description

Read in the particles data, calculate the domain size
Distribute particles with HSFC method.
In each partition
for $t = 1 \rightarrow total\_number\_of\_timesteps$ **do**
Step 1: calculate viscosity term: $\nu\nabla^2\mathbf{u_i^n}$ {**Loop 1**}
Step 2: $\mathbf{u_i^*} = \mathbf{u_i^n} + \left(\nu\nabla^2\mathbf{u_i^n} + \mathbf{F_i^n}\right)\mathbf{\Delta t}$, Calculate $\mathbf{u_i^*}$ without pressure gradient.
Step 3: $\nabla\cdot\left(\frac{1}{\rho}\nabla p^{n+1}\right)_i = \frac{1}{\Delta t}\nabla\cdot\mathbf{u_i^\star}$, Solve pressure Poisson equation to get pressure P_i^{n+1} {**Loop 2**}
Step 4: Get pressure gradient ∇p_i^{n+1} {**Loop 3**}
Step 5: $\mathbf{u_i^{n+1}} = \mathbf{u_i^*} - \frac{\mathbf{\Delta t}}{\rho}\nabla\mathbf{p_i^{n+1}}$, Calculate velocity $\mathbf{u_i^{n+1}}$
Step 6: $\mathbf{r_i^{n+1}} = \mathbf{r_i^n} + \mathbf{\Delta t}\left(\frac{\mathbf{u_i^{n+1}}+\mathbf{u_i^n}}{2}\right)$, Particles positions are centred in time
Step 7: $\delta\mathbf{r_s} = -\mathbf{D'}\nabla\mathbf{C}$. Shift the particle by Ficks law {**Loop 4**}
end for

There are four major loops involve using particles neighbour lists(see Algorithm 1 Loop 1-4), where Loop 1, Loop 2 and Loop 3 are using the same neighbour lists. Due to data dependencies, we cannot merge these loops together into one loop. And all these loops involve calculation of smoothing kernel gradients, which can be saved for reuse. However, we have found that performing redundant calculation of smoothing kernel gradients are not expensive enough to save for reuse in the rest of loops. Even without particles reordering, the single core run-time costs of performing redundant calculation of smoothing kernel gradient are almost the same as the approach of saving and reusing approach. This may change when using different smoothing kernels, such as Gaussian smoothing kernel which requires expensive intrinsic function calculation and the wider particles neighbour lists.

3.2. *Efficient Neighbour list searching*

In general SPH software applications, 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.

Based on the run-time performance, we use cell linked list (CLL) approach [4][10]. The first step is the creation of a CLL list, all particles should be initially organised 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].

In parallel, the added complication is the number of particles in each domain is unknown due to:

- dynamic load balancing for particles movement
- boundary conditions, e.g.: mirror particles, which will be generated according to the fluid particles in each domain

The memory allocation could be very tricky. However, the dynamic vector approach can easily tackle these issues because of the precalculation step.

3.3. *Reordering Particles*

The original order of particles are 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 reorganises the data memory accesses so that one can load into 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 ISPH3D neighbour searching is operated according to cells' neighbours, it is natural to consider ordering particles according to cells order. In the ISPH3D code, the cells have been ordered on the Space-filling curve, 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. However reorder cells along the HSFC has overhead due to large number of cells, this currently only be done everything 10 timestep and the reordering current only offers $5\% \sim 8\%$ performance improvements due reordering overhead. Further optimization need to be done to reduce the overhead of space filling curve ordering.

4. WP3: Optimisation of 3-D pressure Poisson solver

For the pressure Poisson equation, there are two different methods have been implemented and can be switched on depending on different applications. Both the left and right hand side of ISPH pressure Poisson equation requires calculation of kernel gradients, therefore they can be put in the same neighbour loop to reduce the costs of floating point operations.

In ISPH3D, we employ PETSc software[11] to solve the sparse linear system. While the majority of implementation are still the same as 2-D[5], we have paid particular attention to reduce accesses to PETSc memory space. For example, the boundary conditions for the free surface flow of pressure Poisson equation, this has to be done before the left hand side matrix and right hand side vector are assembled.

PETSc partitions matrices by continuous rows in parallel. 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. The assembly matrix has therefore been implemented row by row, the accesses to PETSc memory space have been reduced to minimal to reduce the overhead of matrix assembly stage.

5. WP4: Parallel I/O with H5Hut

SPH simulations running on large high-performance computing systems over many time steps can generate an huge amount of data for post-processing and analysis. Achieving high-performance I/O for this data, effectively managing it on disk, and interfacing it with analysis and visualization tools can be challenging, especially for domain scientists who do not have I/O and data management expertise. The H5Hut[14] library is an implementation of several data models for particle-based simulations that encapsulates the complexity of parallel HDF5 and is simple to use, yet does not compromise performance.

H5Hut is tuned for writing collectively from all processors to a single, shared file. Although collective I/O performance is typically (but not always) lower than that of file-per- processor, having a shared file simplifies scientific workflows in which simulation data needs to be analyzed or visualized. In this scenario, the file-per-processor approach leads to data management headaches because large collections of files are unwieldy to manage from a file system standpoint. On a parallel file system like Lustre, even the ls utility will break when presented with tens of thousands of files, and performance begins to degrade with this number of files because of contention at the metadata server. Often a post-processing step is necessary to refactor file-per-processor data into a format that is readable by the analysis tool. In contrast, H5Hut files can be directly loaded in parallel by visualization tools like VisIt and ParaView.

The implementation based on the unreleased version of H5Hut-1.99-15rc1 provided Dr. Achim Gsell from PSI, Switzerland.

6. Performance Improvement and Analysis

The 3-D dam break with a dry bed has been used in this paper as the benchmark test case. Fig. 2 shows the pressure contours at $t = 0.00, 0.34, 0.58, 1.26$. The total number of particles used for the benchmark is around 100 million. The sparse linear solver is using multigrid preconditioner HYPRE BoomerAMG[12] and Krylov subspace method GMRES. The multigrid preconditioner HYPRE BoomerAMG has two phases, the first phase is setup phase including selection of coarse grids, creation for the interpolation operators, and the representation of the fine grid matrix operator on each coarse grid. The second phase is the solving phase containing matrix-vector multiply and the smoothing operators.

We are using the UK National HPC platform ARCHER, which is Cray XC30 system. ARCHER compute nodes contain two 2.7 GHz, 12-core E5-2697 v2 (Ivy Bridge) series processors. Each of the cores in these processors can support 2 hardware threads (Hyper-threads). Within the node, the two processors are connected by two QuickPath Interconnect (QPI) links. Standard compute nodes on ARCHER have 64 GB of memory shared between the two processors. The memory is arranged in a non-uniform access (NUMA) form: each 12-core processor is a single NUMA region with local memory of 32 GB. Access to the local memory by cores within a NUMA region has a lower latency than accessing memory on the other NUMA region. There are 4544 standard memory nodes (12 groups, 109,056 cores) and 376 high memory nodes (1 group, 9,024 cores) on ARCHER giving a total of 4920 compute nodes (13 groups, 118,080 cores), providing a total of 2.55 Petaflops of theoretical peak performance.

The efficiency is obtained with the following formula:

$$S_p = \frac{T_1}{T_p * p} * 100.00\% \tag{1}$$

where T_1 is the wall time with 1 node, each node comprises 24 Intel E5-2697 cores, T_p is the wall time with p nodes ($p \geq 1$). S_p is the statistic generally used to show the code scalability.

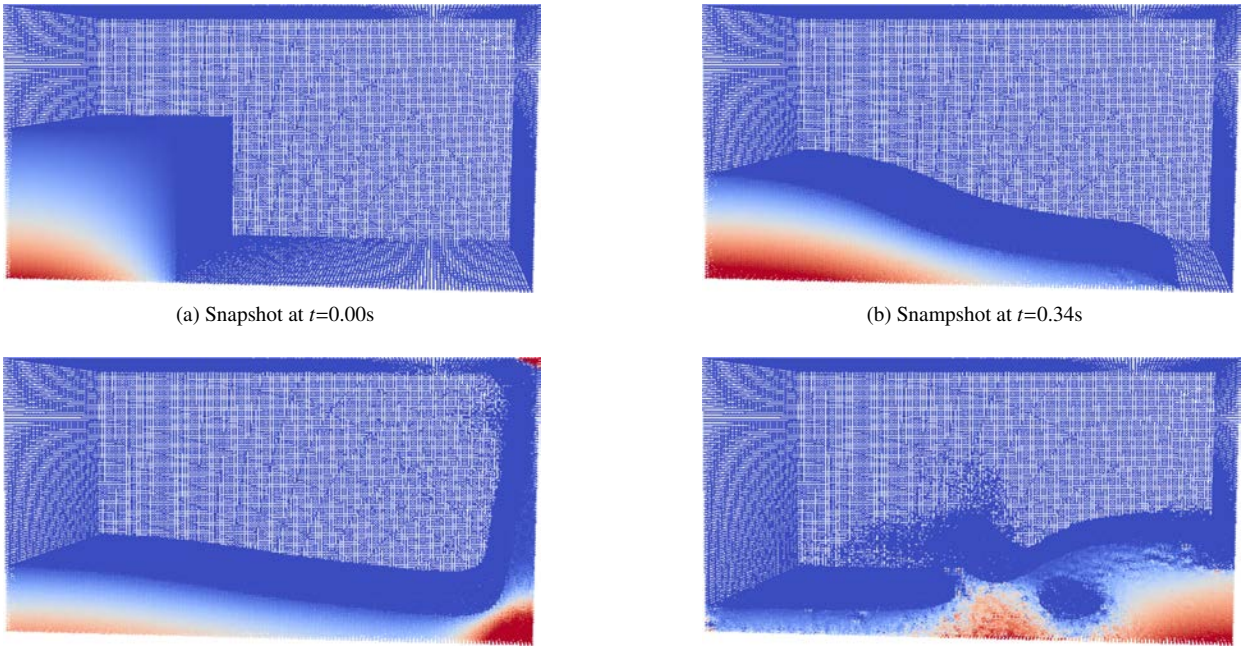
Fig. 3a shows the I/O wall time running 10 timesteps, with 97.8 million fluid particles, the total amount of data per timestep is around 4.1GB, we have experiment with 32-512 archer nodes. The I/O keeps relative constant low costs with H5Hut I/O.

Fig. 3 shows the overall performance of the ISPH3D running 10 time steps, Fig. 3a gives the wall time of each components/kernels in the ISPH3D. Fig. 3b gives the efficiency of each components/kernels. The efficiency defined in Equation 1. The KSPSOLVE gives performance of GMRES, PCSETUP and PCAPPLY gives statistics of BoomerAMG's setup phase and solve phase, MatMult is matrix vector multiplication, MatAssemble is the cost of calculation of pressure Poisson equation matrix coefficients, the right hand side and assembly to the PETSc memory space for

later solving. SPH_KERS are those kernels mentioned in Algorithm 1 from step 1 to step 7 except step 3 (pressure solver).

In the Fig. 3a, we can see the majority of the walltime is now spent solving the pressure Poisson equation. The main costs in the solver are KSPSOLVE (using GMRES) and PCApply which is BoomerAMG's solver phase. The other SPH related kernels account very small percentage compared with pressure solver.

In the Fig. 3b shows the scalabilities of each kernel. Firstly, we can see the ISPH kernels are now able to scale linearly. With up to 12288 cores, it can still achieve 99.04% efficiency. In the PPE solver part, With 12288 cores, KSPSOLVE and PCAPPLY can still reach more than 60% efficiency. However, the scalability of those relatively low cost parts, like PCSETUP, MatAssemble becomes worse (only 20% efficiency with 12K cores) when using large number of cores. They are now become the bottleneck for the scalibility of ISPH3D. These issues need to be further investigated.



(a) Snapshot at t=0.00s

(b) Snampshot at t=0.34s

(c) Snampshot at t=0.58s

(d) Snampshot at t=1.26s

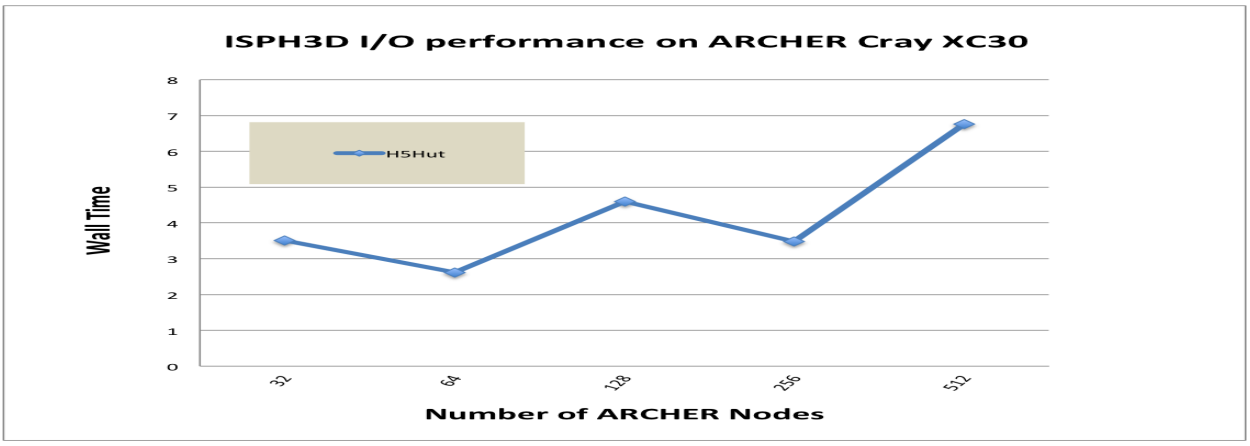
Figure 2: The Snapshots for the 3-D Dam break dry bed test case at $t = 0.00s, 0.34s, 0.58s, 1.26s$

7. Summary and Conclusions

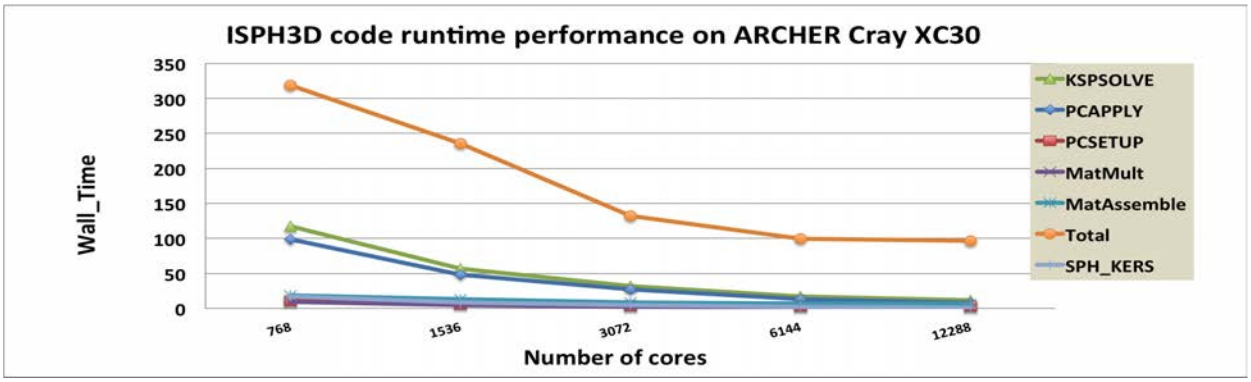
We have investigated the software data structure and algorithms carefully in order to achieve strong scalability in large-scale systems. In order to reduce memory footprint for mapping particles into different MPI partitions, we proposed to use bounding box instead of cells themselves as partition objects. This offers rewards of small memory footprint and reducing run-time overhead of HSFC partition kernels. We also demonstrated that domain decomposition with space filling curve can efficiently deal with irregular distributed particles like 3-D Dam break with obstacles.

We have analysised the ISPH algorithm in detail, some of those loops can be merged together in order to reduce extra floating point operations and reducing memory footprint in the meantime, however this may cause issue of software modularity.

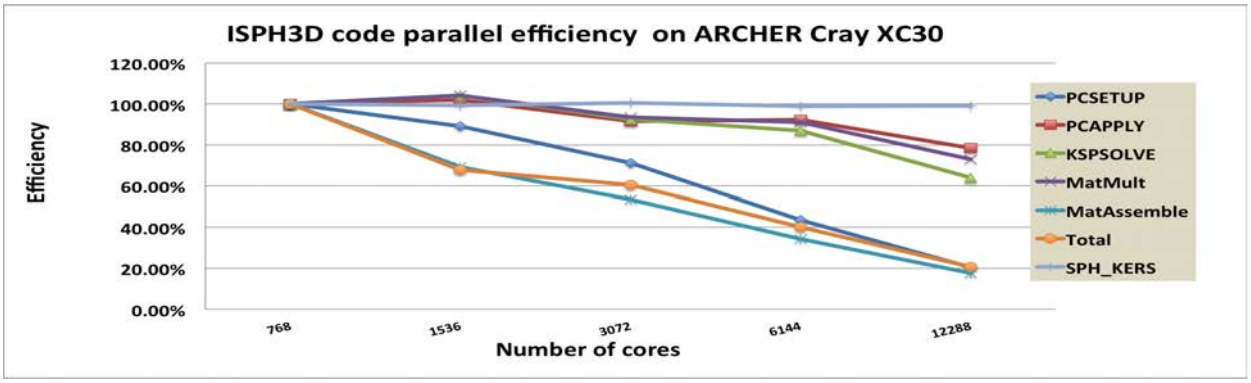
For 3-D simulations, the main cost is now the pressure Poisson solver. We have experimented with multigrid preconditioner HYPRE BoomerAMG with GMRES. The bottlenecks are the setup phase of preconditioner and the PETSc matrix assembly. Further investigation is needed to be made in order to clarify the unscalable parts in PCSETUP and MatAssemble.



(a) ISPH3D I/O Performance Analysis



(a) ISPH3D Walltime Analysis



(b) ISPH3D Parallel Efficiency Analysis

Figure 3: Strong scaling results for the whole ISPH3D using up to 512 XC30 nodes (12288 cores).

The code has been benchmarked on the UK National Supercomputing Platform ARCHER, where we have presented the results of memory efficient implementation for 3-D incompressible SPH, using more than 10,000 cores for the first time to the best of the authors' knowledge. We use the 3-D dam break with a dry bed as the benchmark test case. The final performance analysis and results using around 100 million particles showed the promising efficiency with 12K cores.

Acknowledgment

The authors would also like to acknowledge the funding support under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk). The authors would also like to thank the EPCC eCSE support team for their help throughout this work

References

[1] Jack Dongarra, Jeffrey Hittinger, John Bell, Luis Chacon, Robert Falgout, Michael Heroux, Paul Hovland, Esmond Ng, Clayton Webster, and Stefan Wild. *Applied Mathematics Research for Exascale Computing*, *Report*, U.S. Department of Energy, ASCR, March 2014. URL:http://science.energy.gov/ /media/ascr/pdf/research/am/docs/EMWGreport.pdf

[2] R. Xu, P.K. Stansby, D. Laurence, *Accuracy and stability in incompressible SPH (ISPH) based on the projection method and a new approach*, *J. Comput. Phys*. 228 (2009) 67036725.

[3] 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*, (2012) **231**:1499-1523.

[4] Domnguez, J. M., Crespo, A. J. C., Gmez-Gesteira, M. and Marongiu, J. C., *Neighbour lists in smoothed particle hydrodynamics*. Int. J. Numer. Meth. Fluids, 67: 2026–2042. 2011. doi: 10.1002/fld.2481

[5] Guo Xiaohu, Steven Lind, Benedict D. Rogers, Peter K. Stansby and Mike, Ashworth. *Efficient Massive Parallelisation for Incompressible Smoothed Particle Hydrodynamics (ISPH) with* 10^8 *particles*. In: The 8th SPHERIC Workshop. Vol. 1. 2013, pp. 397402.

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

[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] 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.

[10] P. Gonnet, *Efficient and scalable algorithms for smoothed particle hy- drodynamics on hybrid shared/distributed-memory architectures*, SIAM Journal on Scientific Computing, vol. 37, no. 1, 2015, pp. C95C121.

[11] PETScManualPage,http://www.mcs.anl.gov/petsc/documentation/index.html

[12] A.H. Baker and M. Schulz and U. M. Yang, *On the Performance of an Algebraic Multigrid Solver on Multicore Clusters*, VECPAR 2010, J.M.L.M. Palma et al., eds., vol. 6449 of Lecture Notes in Computer Science, Springer-Verlag, 2011, pp. 102-115

[13] Vacondio, R., Rogers, B.D., Stansby, P.K., Mignosa, P., Feldman, J., *Variable resolution for SPH: A dynamic particle coalescing and splitting scheme*, Computer Methods in Applied Mechanics and Engineering, 256, pp. 132-148, 2013.

[14] Mark Howison, Andreas Adelmann, E. Wes Bethel, Achim Gsell, Benedikt Oswald, Prabhat, *H5hut: A High-Performance I/O Library for Particle-based Simulations*,In Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS '10), 20-24 September 2010, Heraklion, Crete, Greece. doi:10.1109/CLUSTERWKSP.2010.5613098