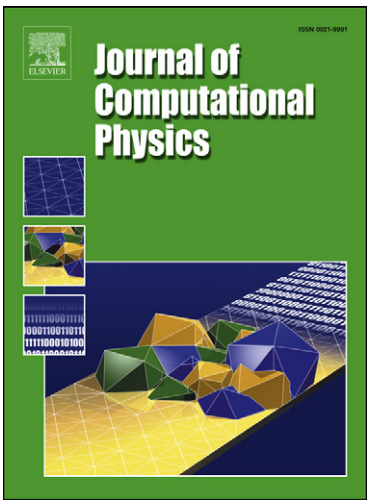


GPU computing of compressible flow problems by a meshless method with space-filling curves

Z.H. Ma, H. Wang, S.H. Pu

PII: S0021-9991(14)00050-3
DOI: 10.1016/j.jcp.2014.01.023
Reference: YJCPH 5047

To appear in: *Journal of Computational Physics*

Received date: 24 April 2013
Revised date: 6 December 2013
Accepted date: 5 January 2014

Please cite this article in press as: Z.H. Ma et al., GPU computing of compressible flow problems by a meshless method with space-filling curves, *Journal of Computational Physics* (2014), http://dx.doi.org/10.1016/j.jcp.2014.01.023

GPU computing of compressible flow problems by a meshless method with space-filling curves

Z. H. Ma[a,*], H. Wang[b,c], S. H. Pu[d]

[a] *Centre for Mathematical Modelling and Flow Analysis, School of Computing, Mathematics and Digital Technology, Manchester Metropolitan University, Manchester M1 5GD, United Kingdom*
[b] *Department of Marine Technology, Norwegian University of Science and Technology, Trondheim NO-7491, Norway*
[c] *Department of Mathematical Information Technology, University of Jyväskylä, Jyväskylä FI-40014, Finland*
[d] *Department of Aerodynamics, Nanjing University of Aeronautics & Astronautics, Nanjing 210016, P.R. China*

Abstract

A graphic processing unit (GPU) implementation of a meshless method for solving compressible flow problems is presented in this paper. Least-square fit is used to discretise the spatial derivatives of Euler equations and an upwind scheme is applied to estimate the flux terms. The compute unified device architecture (CUDA) C programming model is employed to efficiently and flexibly port the meshless solver from CPU to GPU. Considering the data locality of randomly distributed points, space-filling curves are adopted to re-number the points in order to improve the memory performance. Detailed evaluations are firstly carried out to assess the accuracy and conservation property of the underlying numerical method. Then the GPU accelerated flow solver is used to solve external steady flows over aerodynamic configurations. Representative results are validated through extensive comparisons with the experimental, finite volume or other available reference solutions. Performance analysis reveals that the running time cost of simulations is significantly reduced while impressive (more than an order of magnitude) speedups are achieved.

Keywords: computational fluid dynamics, least square, clouds of points, CUDA, data locality, mixed language programming

1. Introduction

With the fast development of computer technology and numerical analysis, computational fluid dynamics (CFD) has been more frequently and widely used in fundamental research and practical industrial applications including aerospace, automotive, coastal, offshore and ocean engineering etc. Development of new industrial products such as commercial airline carriers, offshore vessels and automobiles usually requires many cycles of data analysis, design, implementation and evaluation. Although scientists and engineers aim at the high efficiency and pay a lot of efforts, it takes vast amount of manual work and intensive numerical computing to tackle the tough assignments. Under these circumstances, a powerful CFD software package capable of providing fast and reliable flow solutions is of extreme importance.

Regarding the underlying numerical algorithm for solving partial differential equations in CFD, meshless methods provide an alternative choice in addition to mesh methods. One distinctive feature of meshless methods is that only clouds of points are required to discretise the domain, while connectivities between points are not necessary. This new kind of method has attracted more and more attention, and efforts have been made to explore its advantage to solve incompressible and/or compressible flows. For compressible flow problems, an early work was carried out by Batina, in which he proposed a least-square based meshless method for solving external inviscid and viscous flows [1]. Oñate et al. proposed a finite point method for potential flows [2]. Löhner et al. applied the finite point method to compressible flow simulations [3]. Sridar and Balakrishnan [4] developed a least-square based upwind scheme. Shu

*Corresponding author. Tel: +44 (0)161-247-1574
Email addresses: z.ma@mmu.ac.uk (Z. H. Ma), hong.wang@ntnu.no (H. Wang), nuaapu@yahoo.com.cn (S. H. Pu)

et al. devised a radius-basis-function based meshless method for solving compressible internal flows. Ma et al. [5] and Ortega [6] developed adaptive meshless methods to automatically move or add/delete points in the domain during computation. Chiu et al. [7] attempted to devise a conservative meshless method, in which mathematical optimisation tools were utilised to calculate the meshless coefficients in order to satisfy the conservation conditions. Löhner and Oñate [8, 9] investigated point generation for two- and three-dimensional problems. To improve the efficiency, Morinishi [10] and Chen & Shu [11] developed implicit meshless methods to compute compressible inviscid and/or viscous flows. Katz and Jameson [12] proposed a background accelerating method named as multicloud, which adopts meshless schemes.

Promising results were presented in the aforementioned works, however most of them were carried out on a CPU core. They have not adapted to parallel computing on modern graphic hardwares, which may deliver tera-scale single- and double-precision floating-point operations per second in very recent years. Different with the CPU, the GPU is purposely designed for intensive data-parallel processing so that it has many more arithmetic and logic units (ALUs) [13] as illustrated in Figure 1. Consequently, a few of data-intensive computer programs, which are based on traditional CFD algorithms for hydrodynamics or aerodyanmics, could greatly benefit from the novel GPU computing technology to improve the performance by several times or even orders of magnitude. Here we give a brief review of some related works. Engsig-Karup [14] et al. realised a fully non-linear free surface potential flow solver for coastal and offshore problems on GPU (with a 40× speedup). Corrigan et al. [15] proposed a 3D cell-centred unstructured grid inviscid flow solver for aerodynamics problems and obtained a 33× speedup. Kampolis et al. [16] ported a 2D/3D vertex-centred unstructured grid Navier-Stokes solver for optimisation problems to the GPU (20× $\sim$ 30× speedup). Asouti et al. [17] extended their unstructured grid steady flow solver to unsteady computation on the GPU (46× speedup). Ran et al [18] applied the structured grid based space-time conservation element and solution element method on the GPU (23× $\sim$ 71× speedup).

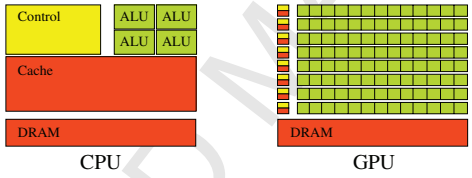


Figure 1: The architectures of CPU and GPU (reproducing Figure 1-2 of [13] for illustration purpose only).

From the point of view of computational efficiency [19], GPU accelerated meshless methods will be competitive. Hence, such kind of study is crucial to the development and application of meshless methods. However it is noticed that there is rare (if not no) investigation of meshless methods on the GPU for the solution of compressible flow problems. The current work intends to contribute to tackling this challenging task. To the best of our knowledge, this paper presents a very first many-core GPU implementation of a meshless method for solving compressible flow problems. Least-square curve fit is employed to estimate the spatial derivatives of mathematical functions as an example. The method is not exclusive to least-square fit scheme, other strategies like radius-basis-function may also be viable. Among various available GPU programming models including OpenGL, Stream (ATI), OpenCL and CUDA etc., we choose CUDA C as our main programming model aiming to achieve an astonishing speedup of the meshless solver on the GPU. For indirect access model based computer programs, memory latency is a bottleneck of the overall performance. Spatial local elements are generally advised to be stored closely in memory considering the cache hit rate. Poor data arrangement will result in high cache miss rate and slow down the computation inevitably no matter whether serial or parallel computing is adopted. In order to improve the data localisation for the meshless solver, space-filling curves (SFC), which are well known for their memory coherence characteristics and high spatial locality, are employed to re-order the irregularly distributed points in the present work.

The rest of the paper is organised as follows. The numerical model, including governing equations, least-square spatial discretisation, explicit Runge-Kutta time advancing scheme, is described in Section 2. Detailed procedure to implement GPU programming of the meshless method is presented in Section 3. Mixed language programming of Fortran and CUDA C is discussed and some valuable suggestions are given to tune the meshless program performance. Space-filling curves are described in Section 4 and a specific mapping algorithm transforming the floating-number

point coordinate to integer-number index is provided. Numerical examples are divided into two categories in Section 5. We firstly carry out detailed evaluations of the method to assess the accuracy and conservation property for steady and time-dependent problems. Then the GPU accelerated solver is applied to solve external steady flows over aerodynamic configurations. Representative results are validated through comparisons with the experimental, finite volume or other available reference results. Performance analysis is then conducted for the meshless GPU solver. Finally, some conclusions are drawn in Section 6.

2. Numerical model

2.1. Governing equations

The present work focuses on the two-dimensional single-phase inviscid compressible flows, of which the mathematical model is represented by the Euler equations indicating the conservation of mass, momentum and energy. The differential forms of these equations can be expressed as

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{E}}{\partial x} + \frac{\partial \mathbf{F}}{\partial y} = 0 \tag{1}$$

where $\mathbf{U}$ is a vector of conservative variables, $\mathbf{E}$ and $\mathbf{F}$ are the flux terms, and are defined as

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_\mathrm{t} \end{bmatrix}, \quad \mathbf{E} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (\rho e_\mathrm{t} + p)u \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (\rho e_\mathrm{t} + p)v \end{bmatrix} \tag{2}$$

in which, ρ is the density, p is the pressure, u and v are the components of velocity vector $\vec{V}$ along x and y axes respectively. The total energy per volume ρe_t is the sum of the internal energy ρe_i and kinematic energy ρe_k, they are evaluated by the following formulae

$$\rho e_\mathrm{t} = \rho e_\mathrm{i} + \rho e_\mathrm{k} \tag{3a}$$

$$\rho e_\mathrm{k} = \frac{1}{2}\rho(u^2 + v^2) \tag{3b}$$

$$\rho e_\mathrm{i} = e_\mathrm{i}(\rho, p) \tag{3c}$$

For ideal gas, the internal energy is given by

$$\rho e_\mathrm{i} = \frac{p}{\gamma - 1} \tag{4}$$

where γ is the ratio of specific heat coefficients and $\gamma = 1.4$ for air.

2.2. Spatial discretisation

For every scattered point in the flow domain, we select its natural neighbours to form a cloud of points. The Euler equations are required to be satisfied in every cloud of points. For any cloud C_i, Eq. (1) can be written as

$$\left.\frac{\partial \mathbf{U}}{\partial t}\right|_{C_i} + \left(\frac{\partial \mathbf{E}}{\partial x} + \frac{\partial \mathbf{F}}{\partial y}\right)_{C_i} = 0 \tag{5}$$

For simplicity, the subscript i is used to represent the cloud C_i in the following. With a proper meshless approximation, Eq. (5) can be discretised as

$$\frac{\partial \mathbf{U}_i}{\partial t} + \sum_{j=1}^{M_i} \left[\left(\alpha_{ij}\mathbf{E}_{ij} + \beta_{ij}\mathbf{F}_{ij}\right) - \left(\alpha_{ij}\mathbf{E}_i + \beta_{ij}\mathbf{F}_i\right)\right] = 0 \tag{6}$$

where M_i is the total number of satellite points in C_i. To determine the meshless coefficients α_{ij} and β_{ij}, least-square curve fit [1, 4, 5, 20], radius basis functions [11] and conservative meshless scheme [7] etc. can be used. Since

our main objective is to implement the meshless method on GPU, we choose least-square curve fit to compute the meshless coefficients in the present work only as an example. This is not exclusive to least-square, other strategies mentioned above may also be feasible.

In order to calculate the flux terms in a non-split manner, Eq. (6) needs to be expressed in a compact form. For this purpose, a parameter λ defined as

$$\lambda = \sqrt{\alpha^2 + \beta^2} \tag{7}$$

and a vector $\vec{\eta} = (\eta_x, \eta_y)$ estimated by

$$\eta_x = \frac{\alpha}{\lambda}, \quad \eta_y = \frac{\beta}{\lambda} \tag{8}$$

are introduced to Eq. (6). Now it can be written in the following form

$$\frac{\partial \mathbf{U}_i}{\partial t} + \sum_{j=1}^{M_i} (\mathbf{G}_{ij} - \mathbf{G}_i)\lambda_{ij} = 0 \tag{9}$$

where

$$\mathbf{G} = \eta_x \mathbf{E} + \eta_y \mathbf{F} = \begin{bmatrix} \rho q \\ \rho u q + p\eta_x \\ \rho v q + p\eta_y \\ (\rho e_t + p)q \end{bmatrix} = q\mathbf{U} + p\mathbf{N}_q \tag{10}$$

in which $\mathbf{N}_q = [0, n_x, n_y, q]^{\mathrm{T}}$, q is the dot product of $\vec{V}$ and $\vec{\eta}$

$$q = u \cdot \eta_x + v \cdot \eta_y \tag{11}$$

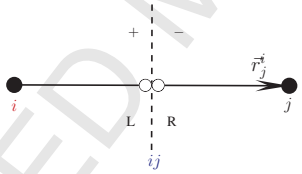


Figure 2: A satellite j, midpoint ij and the centre i in the cloud

As illustrated in Figure 2, the flux function $\mathbf{G}$ at the midpoint P_{ij} is evaluated by

$$\mathbf{G}_{ij} = \mathbf{G}(\mathbf{U}_{ij}^{\mathrm{L}}, \mathbf{U}_{ij}^{\mathrm{R}}) \tag{12}$$

A simple method to calculate the conservative variables for the left side $\mathbf{L}$ and the right side $\mathbf{R}$ is

$$\mathbf{U}_{ij}^{\mathrm{L}} = \mathbf{U}_i, \quad \mathbf{U}_{ij}^{\mathrm{R}} = \mathbf{U}_j \tag{13}$$

This is the classical first-order Godunov scheme, which assumes a piecewise constant distribution of the flow variables. To improve the accuracy, a piecewise linear reconstruction of the data is adopted in this research. When reconstructing the data, one option is to choose the conservative variables, another is to reconstruct the characteristic variables and the other way is to select the primitive variables. In the present work, the data is reconstructed with the primitive variables $\mathbf{W} = (\rho, u, v, p)$ due to its simplicity compared to the conservative and characteristic variables

$$\mathbf{W}_{ij}^{L} = \mathbf{W}_i + \frac{1}{2}\nabla \mathbf{W}_i \cdot \mathbf{r}_j^i \tag{14a}$$

$$\mathbf{W}_{ij}^{R} = \mathbf{W}_j - \frac{1}{2}\nabla \mathbf{W}_j \cdot \mathbf{r}_j^i \tag{14b}$$

As high order schemes tend to produce spurious oscillations in the vicinity of large gradients [21], a slope or flux limiter needs to be used to satisfy the TVD constrains [22]. In the present work, the following slope limiter is employed

$$\varphi^{\mathrm{L}} = \frac{\nabla \mathbf{W}_i \cdot \vec{r}^i_j \Delta \mathbf{W}^i_j + \left|\nabla \mathbf{W}_i \cdot \vec{r}^i_j \Delta \mathbf{W}^i_j\right| + \epsilon}{(\nabla \mathbf{W}_i \cdot \vec{r}^i_j)^2 + (\Delta \mathbf{W}^i_j)^2 + \epsilon} \tag{15a}$$

$$\varphi^{\mathrm{R}} = \frac{\nabla \mathbf{W}_k \cdot \vec{r}^i_j \Delta \mathbf{W}^i_j + \left|\nabla \mathbf{W}_k \cdot \vec{r}^i_j \Delta \mathbf{W}^i_j\right| + \epsilon}{(\nabla \mathbf{W}_k \cdot \vec{r}^i_j)^2 + (\Delta \mathbf{W}^i_j)^2 + \epsilon} \tag{15b}$$

where $\Delta \mathbf{W}^i_j = \mathbf{W}_j - \mathbf{W}_i$. A small value ϵ is introduced to prevent null division in the smooth regions where differences approach zero, it is set as $\epsilon = 10^{-12}$ in this research. In summary, the conservative variables are estimated by the reconstructed primitive variables

$$\mathbf{W}^L_{ij} = \mathbf{W}_i + \frac{1}{2}\varphi^{\mathrm{L}} \nabla \mathbf{W}_i \cdot \mathbf{r}^i_j \tag{16a}$$

$$\mathbf{W}^R_{ij} = \mathbf{W}_j - \frac{1}{2}\varphi^{\mathrm{R}} \nabla \mathbf{W}_j \cdot \mathbf{r}^i_j \tag{16b}$$

$$\mathbf{U}^{\mathrm{L,R}}_{ij} = \mathbf{U}(\mathbf{W}^{\mathrm{L,R}}_{ij}) \tag{16c}$$

The flux function $\mathbf{G}_{ij}$ is then computed by the HLLC approximate Riemann solver [21, 23–25].

2.3. Temporal discretisation

In the present work, the Euler equations are treated by the method-of-line, which separates the temporal and spatial spaces. The semi-discrete form of the governing equations is given by

$$\frac{\mathrm{d}\mathbf{U}}{\mathrm{d}t} + \mathbf{R} = 0 \tag{17}$$

where $\mathbf{R}$ represents the residual.

For steady flows, a forward difference discretisation of Eq. (17) for cloud i, is

$$\frac{\mathbf{U}^{n+1}_i - \mathbf{U}^n_i}{\Delta t} + \mathbf{R}_i = 0 \tag{18}$$

An explicit four-stage Runge-Kutta scheme is applied to update the solution from time level n to $n+1$,

$$\mathbf{U}^{(0)}_i = \mathbf{U}^n_i \tag{19a}$$

$$\mathbf{U}^{(1)}_i = \mathbf{U}^{(0)}_i - \alpha_1 \Delta t_i \mathbf{R}^{(0)}_i \tag{19b}$$

$$\mathbf{U}^{(2)}_i = \mathbf{U}^{(0)}_i - \alpha_2 \Delta t_i \mathbf{R}^{(1)}_i \tag{19c}$$

$$\mathbf{U}^{(3)}_i = \mathbf{U}^{(0)}_i - \alpha_3 \Delta t_i \mathbf{R}^{(2)}_i \tag{19d}$$

$$\mathbf{U}^{(4)}_i = \mathbf{U}^{(0)}_i - \alpha_4 \Delta t_i \mathbf{R}^{(3)}_i \tag{19e}$$

$$\mathbf{U}^{n+1}_i = \mathbf{U}^{(4)}_i \tag{19f}$$

where $\alpha_1 = \frac{1}{4}$, $\alpha_2 = \frac{1}{3}$, $\alpha_3 = \frac{1}{2}$ and $\alpha_4 = 1$ are the stage coefficients.

3. GPU programming

To carry out parallel computing on the GPU, heterogeneous programming models need to be utilised to deal with both the CPU and GPU. A GPU computing program usually starts from the CPU where the necessary data should be prepared then transferred to the GPU to start the computation. Once the computing intensive task is finished on the GPU, the results are sent back to the CPU. This general procedure is illustrated in Figure 3. OpenCL [26] and CUDA [13] are two important heterogeneous models used by many computer programmers to tackle parallel computing tasks on GPUs. In the present work, we adopt CUDA as our main GPU programming model.

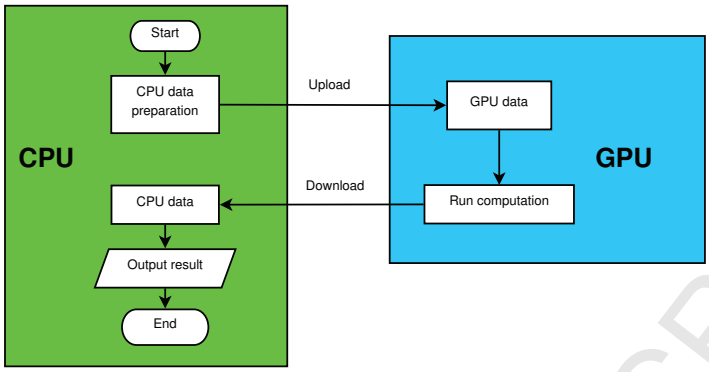


Figure 3: A general procedure of GPU computing

3.1. Mixed Fortran and CUDA C

The underlying single thread meshless CPU solver [27] was exclusively coded in Fortran 90 language. To re-code a Fortran program with CUDA for GPU computing, a relative easy way for a Fortran programmer is to use CUDA Fortran (PGI [28]) which natively supports Fortran language. Another option is re-write the key Fortran subroutine with CUDA C (NVIDIA [13]), which may exploit full features of CUDA for GPU computing. Hence, our choice is to program the meshless solver by mixing Fortran and CUDA C in the present work. This strategy is quite flexible as those lightweight CPU Fortran subroutines do not need to be revised and attention can be focused on the computing-intensive portions.

Since the Runge-Kutta iteration procedure, which consists of flow gradient estimation, data reconstruction, TVD limitation, flux term evaluation and temporal integration, is the most time-consuming part of the whole program, we decide to rewrite this portion of Fortran code with CUDA C. Other part of code, which consists of the IO operation and least-square coefficients computation, is kept the same. In this way, most of the computing task can be kept on the GPU and frequent memory transfer between the CPU and GPU can be avoided effectively.

The CUDA C source code files are compiled by the NVIDIA CUDA compiler driver NVCC [13] and the Fortran source code files can be compiled by the GNU, PGI or Intel Fortran compiler. The intermediate object files are created by the compilers, then the final executable file is generated by assembling all the object files linked with some appropriate libraries.

An example is given here to show our mixed language programming strategy by the following two pieces of computer codes Listing 1 and 2 . The main program is written in Fortran 90 and it is responsible for the IO operation and pre-processing of the meshless clouds. The Fortran main program calls the CUDA C function and the CPU data reference is sent to the CUDA C function through the arguments. The CUDA C function transfers the CPU data to the GPU and launch the GPU kernel functions to do the computing. Finally, the result is transferred back to the CPU.

Listing 1: The Fortran main program

```
1  program main
2    call CloudInput(CPU_data)
3    call CloudPreProcess(CPU_data)
4
5    !to run computation on the GPU
6    call Runge_Kutta_Iteration_GPU(CPU_data)
7
8    call ResultOutput(CPU_data)
9  end program main
```

Listing 2: A key CUDA C function

```
1  #ifdef __cplusplus
```

```
extern "C"
#endif
void Runge_Kutta_Iteration_GPU(CPU_data)
{
  //copy data from CPU to GPU;
  cudaMemcpy(CPU_data, GPU_data, data_size, cudaMemcpyHostToDevice);

  //time advancing
  do while(t<T){
      for(i=0;i<4;i++){
        //launch the kernel function to compute flux
        Flux<<<block,grid>>>(GPU_data);
        //update the flow variables
        FlowUpdate<<<block,grid>>>(GPU_data);
      }
    }

  //copy data from GPU to CPU;
  cudaMemcpy(GPU_data, CPU_data, data_size, cudaMemcpyDeviceToHost);
}
```

Special attention should be paid to the CPU data reference sent from the main Fortran program to the CUDA C function. When Fortran derived data types (e.g. Listing 3) are used to organise the CPU data for the main program, compatible C/C++ structures (e.g. Listing 4) ought to be used to receive the data reference correctly. The following two pieces of codes (Listing 3 and 4) are given to illustrate the strategy of data organisation with Fortran and C/C++. For a discrete point in the flow field, it may have information relating to the flow variables and flux terms besides the coordinate. All of these data can be capsulised in a Fortran derived type or C/C++ structure. In this paper, we would also like to mention that user defined structures like those listed below can be used on the GPU besides the CPU. As this important piece of information is not clearly revealed in the CUDA C programming guide [13], our findings might be useful to readers who prefer to utilise the C/C++ structures to manage their data.

Listing 3: A Fortran derived type for the point data

```
Type Point
  real::x,y !coordinate
  real::U(4) !conservative flow variables
  real::R(4) !right hand side residual
End Type Point
```

Listing 4: The compatible C/C++ structure for the point data

```
typedef Struct{
  float x,y; //coordinate
  float U[4]; //conservative flow variables
  float R[4]; //right hand side residual
}Point;
```

3.2. GPU memory hierarchy and addressing

Listing 5: A GPU kernel function to compute spatial derivatives

```
__global__ void FlowDerivative(GPU_data)
{
  //index
  int i=blockDim.x*blockIdx.x+threadIdx.x;

  //derivative of density
  for(int j=0;j<M;j++){
    dRhodx[i]+=alpha[j]*Rho[C[i][j]];
  }
}
```

In order to apply algebraic operations on the data stored in the GPU global memory, we need to use the thread index to address the memory as shown in the above CUDA C code for a GPU kernel function (Listing 5), which computes the derivatives of flow variables. In the kernel function *FlowDerivative*, the variables (*blockDim*, *blockIdx*

and *threadIdx*) relate to the thread hierarchy concept for GPU computing. Detailed description of these important variables can be found in [13] (Section 2.2).

Besides the global memory, the memory spaces on modern graphic cards also include local, shared, texture, constant memory and registers. Among these, the global, local and texture memory have the greatest access latency, followed by the constant memory, shared memory and registers [13]. A golden rule to improve the GPU memory performance is to use as many registers and shared memory as possible and to visit the global memory as less frequently as possible. However registers and shared memory are rare resources compared to the global memory.

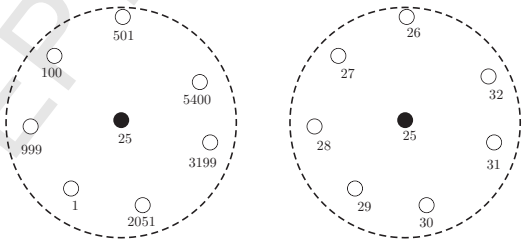
For direct addressing model based applications, proper use of the shared memory can greatly improve the performance especially when there are a lot of data reuse. However, for indirect addressing model based applications, it is difficult to utilise the shared memory due to the unpredictable memory access pattern. As the meshless solver indirectly addresses the data, we do not employ the shared memory but use as many registers as possible in the present work.

Although registers have the lowest latency, they are scarce as mentioned. Every thread block launched on the GPU has a limited number of registers provided by the hardware. A common CUDA programming guide suggests to increase occupancy by launching many threads in a block intending to hide the memory latency. However, this will result in register pressure while the threads in a block are competing for registers. For our GPU meshless program, high occupancy does not always mean high performance. Consequently, the number of threads in a block needs to be tuned to obtain the optimal performance.

When fetching data from or writing them to the global memory, coalesced memory access is the best pattern. However, this is very hard to realise for indirect addressing model based meshless solver. Improving the data locality will promote the memory performance. One option is to place the hot data which are accessed more frequently than others in the beginning of structures. The heavily accessed portions and rarely accessed portions can even be split into separated structures [29]. In addition, re-ordering the irregularly distributed points by using space-filling curves is also very useful.

4. Space-filling curve

For modern computers, the memory is far behind a CPU or GPU core in terms of clock rate and this gap is continuously widening in the near future. In consequence, the memory latency bottleneck has great effects on programs' performance. For indirect addressing model based applications, the locality of data is of extreme importance to the overall performance [15]. A general rule to achieve good data locality is that the elements which are spatially local should be stored closely in memory. Otherwise, high cache miss rate will occur and slow down the computation inevitably.

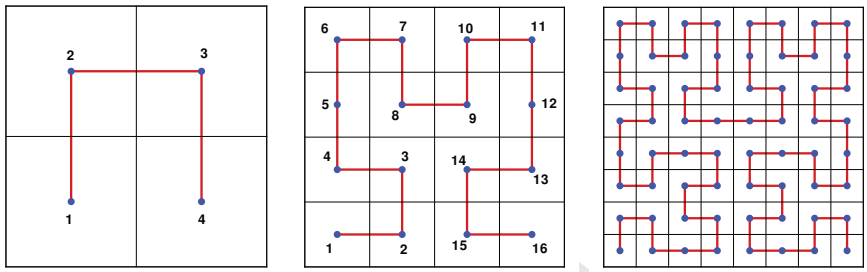


(a) non-sequential ordering (b) ideal sequential ordering

Figure 4: The arrangement of scatter points.

When programming a meshless solver, attention should be paid to the data arrangement in order to follow the locality rule. Figure 4 shows two kinds of ordering patterns for irregular points. One is a random non-sequential ordering, in which the index of each point might be greatly different with the others' even these points are spatially close. The other is an ideal sequential ordering, where the points are arranged and accessed linearly. The reason we use the word "ideal" to refer the second ordering pattern is that this layout is very difficult to achieve for many real applications with irregular distributions of scattered points.

Space-filling curves (SFC) are well known for their memory coherence characteristics and high spatial locality [30]. Because of these salient features, SFC are widely used in computer and computational science to enhance the performance of programs. Vo et al. [31] used SFC to compute cache-friendly layouts of unstructured geometric data. Aftosmis et al. [32] adopted SFC to produce single pass algorithms for mesh partitioning, multigrid coarsening, and inter-mesh interpolation to Cartesian grid CFD solver. Alauzet et al. [33] applied SFC to realise parallel anisotropic mesh adaptation. There are a great amount of literature investigating SFC and their applications, but we only dig up a small number of important publications for SFC [34–37] in the present work.



(a) $K = 1$ (b) $K = 2$ (c) $K = 3$

Figure 5: Hilbert space-filling curves

In order to improve the localisation of data for initially irregularly distributed meshless points, we apply the Hilbert SFC to re-number the points. The original Hilbert SFC [35] was defined in a two-dimensional Euclidean space and its image is a unit square of $[0, 1] \times [0, 1]$. Suppose the square consists of $2^K \times 2^K$ points, where K is an approximation number, we use an integer coordinate (i, j) to index one of these points and use a function d to indicate the distance along the curve when it reaches that point. Hilbert SFC provides a mapping between two- and one-dimensional spaces

$$H : P(i, j) \leftrightarrow d, \qquad P(i, j) \in R^2 \;\; \& \;\; d \in R^1 \tag{20}$$

The index i or j stands for integer value between 1 and 2^K. Figure 5 shows the Hilbert SFC for the approximation number $K = 1$, 2 and 3 within the unit square. Butz [38] proposed an effective algorithm for generating the Hilbert

SFC in a byte-oriented manner and more details can be found in his work.

Algorithm 1: Mapping (x, y) to (i, j)

Input : $P(x_m, y_m)$
Output: $P(i_m, j_m)$
begin Domain length L
  $L_{\max} = \max |P(x_m, y_m) - P(x_n, y_n)|, \quad m \neq n;$
  $L_{\min} = \min |P(x_m, y_m) - P(x_n, y_n)|, \quad m \neq n;$
end
begin Approximation number K
  $f_K = \log_2(L_{\max}/L_{\min} + 1)$;
  $K = \lceil f_K \rceil$;
end
begin Step size resolution h
  $h = 1/(2^K - 1)$;
end
begin Whole number index i & j
  $f_x = (x_m - \min(x))/(L_{\max} \times h)$;
  $f_y = (y_m - \min(y))/(L_{\max} \times h)$;
  $i_m = \lceil f_x \rceil$;
  $j_m = \lceil f_y \rceil$;
end

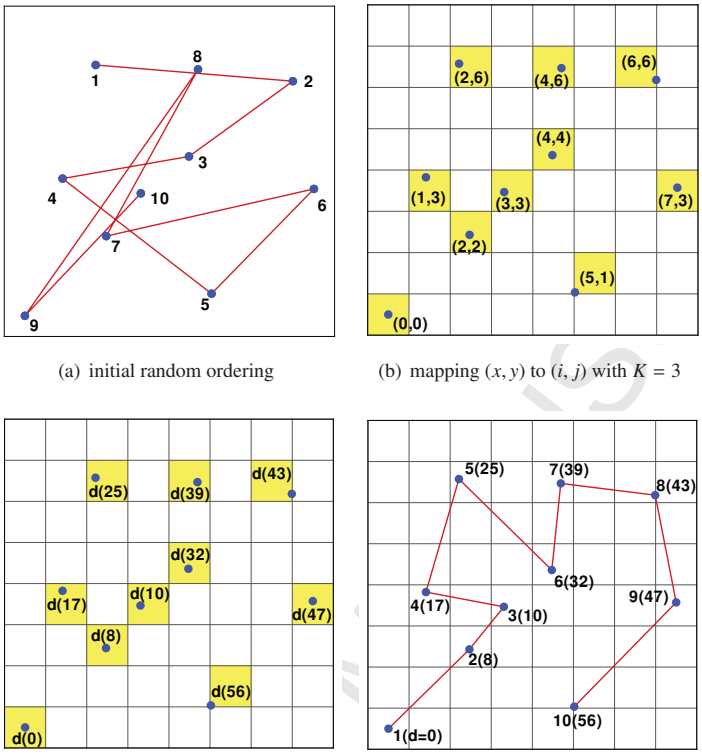
In our applications, the computational domains might not be unit squares. In order to construct the Hilbert SFC for these applications, we firstly need to map the real number coordinate (x, y) to the whole number index (i, j) for all the points and the corresponding procedure is listed in Algorithm 1. The subscript m in Algorithm 1 is an index of a node and the ceiling function $\lceil f_x \rceil$ maps the real number f_x to the smallest integer not less than f_x, e.g. $\lceil 0.5 \rceil = 1$. Once the whole number index is obtained for every point, Butz's algorithm [38] is applied to compute the distance function d. Then all of the points are re-arranged by a quick sort algorithm [32, 39] according to the value d.

To illustrate this approach, a simple example is given here as shown in Figure 6. Ten points are initially scattered in a 2D space with random indexing numbers. In Figure 6(a), although the $3th$ and $10th$ points are spatially close to each other, they are not stored closely in memory. To construct the Hilbert SFC, we firstly apply a mapping of the floating-number coordinate (x, y) to the integer-number index (i, j) with the approximation number $K = 3$ as shown in Figure 6(b). Then the function d is computed for every node with its integer-number index (i, j) as illustrated in Figure 6(c). Finally, all of the points are sorted according to the value of d. In Figure 6(d), the value of d for every point is shown in the brace behind its sorted index number, e.g. 2(8) represents the second point and its distance function is $d = 8$.

The current SFC re-numbering method is similar with the bin numbering scheme adopted by Corrigan et al. [15] (see Löhner's book [40] for detailed description of the bin numbering scheme) in improving the data locality. Other techniques such as advancing front and reverse Cuthill-McKe etc. might also be very useful as stated in [15].

5. Numerical results and discussions

Detailed evaluations of the fundamental numerical method are carried out to assess the accuracy and conservation property. Then, the GPU accelerated solver is employed to simulate compressible flows over single- and multi-element aerofoils. All the following numerical simulations presented in this paper are performed on a Linux workstation equipped with 4 Intel Xeon E5645 CPUs@2.4GHz and 24GB RAM. Two NVIDIA graphics cards Quadro 2000 and Tesla C2075 are installed on the workstation. The specifications of the two graphic cards are listed in Table 1. The operating system is Ubuntu 10.10 64-bit. We use PGI Fortran and NVCC to compile Fortran and CUDA C codes of single-precision respectively. The optimisation level for each compiler is set to `-O3` without debugging and profiling options. Two libraries *lstdc++* (C++ run time library) and *libcudart* (CUDA run time library) need to be linked to the object files in the final assembling stage in order to guarantee the executable program be generated successfully. The meshless flow solver on the CPU is run in a single thread pattern.



(a) initial random ordering

(b) mapping (x, y) to (i, j) with $K = 3$

(c) distance function d

(d) re-ordering by quick sort

Figure 6: An example of point re-ordering with Hilbert SFC

Table 1: Specifications of NVIDIA Quadro 2000 and Tesla C2075 graphic cards.

	Quadro 2000	Tesla C2075
Clock Rate	1.25 GHz	1.15 GHz
Global memory	1 GB	6 GB
Shared memory	48 KB	48 KB
Registers per block	32768	32768
Number of multiprocessor	4	14
Cores per multiprocessor	48	32
Total number of cores	192	448
Compute capability	2.1	2.0

5.1. Conservation study

Numerical experiments including vortex evolution, internal supersonic flow, shock tube and incident shock past a cylinder are carried out to empirically demonstrate the accuracy and conservation property of the present method.

5.1.1. Vortex evolution

This test was proposed by Shu [41] to verify the accuracy of a numerical method for Euler equations. The problem is set up in a square domain $[0, 10] \times [0, 10]$ with the mean flow $(\rho, u, v, p) = (1, 1, 1, 1)$. An isentropic vortex is added to the mean flow

$$(\delta u, \delta v) = \frac{\epsilon}{2\pi} e^{0.5(1-r^2)}(5 - y, x - 5) \tag{21a}$$

$$\delta T = -\frac{(\gamma - 1)\epsilon^2}{8\gamma\pi^2} e^{1-r^2} \tag{21b}$$

$$\delta S = 0 \tag{21c}$$

where $S = p/\rho^\gamma$ is the entropy, $T = p/\rho$ is the temperature, $\epsilon = 5$ is the vortex strength and $r^2 = (x - 5)^2 + (y - 5)^2$. Periodical condition is applied on all the boundaries. We firstly distributed 489 points in the domain as shown in Figure 7 and then we systemically refine the point cloud. On each level of point cloud, the flow is simulated from start until $t = 2$, the corresponding numerical error for density is listed in Table 2, which indicates that the present method is of second order accuracy generally.

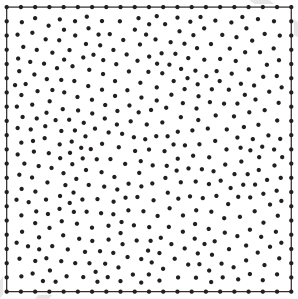


Figure 7: Distribution of the first level point cloud for vortex evolution problem.

Table 2: Density error estimate on different levels of point cloud at $t = 2$.

Level	Point number	h	L_1 error	L_1 order	L_∞ error	L_∞ order
1	489	1/5	4.075×10^{-3}	-	8.106×10^{-2}	-
2	1873	1/10	9.022×10^{-4}	2.18	1.851×10^{-2}	2.14
3	7329	1/20	2.228×10^{-4}	2.03	3.561×10^{-3}	2.39
4	28993	1/40	4.875×10^{-5}	2.20	7.638×10^{-4}	2.23

5.1.2. Supersonic flow in a channel

The conservation property of a numerical scheme is very important for internal flows and time-dependant problems etc. In order to empirically demonstrate the conservation of the present method, here we choose supersonic flows in a rectangular channel with a 4% circular bump. The height of the channel is 1, its width is 3 and the circular bump is located at the centre of the bottom. The top and bottom boundaries are solid walls, the flows enter through the left boundary (inlet), where the Mach number is $M_{\text{inlet}} = 1.4$. The right side is the outlet boundary.

This kind of problem provides an ideal candidate to assess the conservation property of a numerical algorithm. According to the conservation law, mass must remain constant over time, as mass cannot change quantity if it is not added or removed. For this case, the inflow mass should be equal to the outflow mass. Otherwise, the difference in

the mass flow can be indicative of the numerical source/sink associated with the underlying scheme [4]. At the inlet and outlet boundaries, the mass flow can be calculated as

$$F_{\text{mass}} = \sum_{i=1}^{N} h_i \rho_i u_i \tag{22}$$

where i is the index of a boundary edges, h is the length of the edge, ρ is the density, u is the velocity, and N is the total number of boundary edges. The difference between the inflow and outflow mass can be estimated as

$$\Delta M = \left| F_{\text{mass, outlet}} - F_{\text{mass, inlet}} \right| \tag{23}$$

and this is an important parameter indicating the conservation of algorithm.

Initially, 341 points are regularly distributed in the domain (group A) and 366 points are irregularly scattered in the domain (group B) as shown in 8. Then these two groups of point clouds are gradually refined up to the fifth level. The computed pressure contours on each level of point cloud are shown in Figure 9, which clearly illustrates that the resolution of the shock waves is greatly improved with cloud refinement. The mass flow difference on each level of point cloud is shown in Figure 10. The conservation error continuously decreases toward zero with refined point clouds. The obtained Mach number distributions on the top and bottom walls are compared to Liou's FVM result [42] in Figure 11, where a good agreement is noted.

5.1.3. Shock tube

A shock tube (see Section 6.4 of [21]) is chosen here to test the capability of the present method to handle time-dependant problems, in which complex physical features such as shock wave, contact discontinuity and rarefaction may evolve with time. The tube is filled with ideal gases of different states separated by an imaginary barrier imposed at $x = 0.3$ initially as shown in Figure 12. The density, velocity and pressure of the gases at the initial state are given by

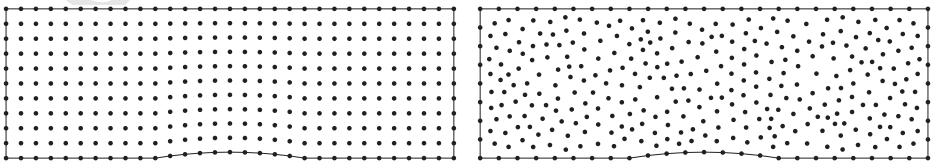
$$\mathbf{W} = \begin{cases} (1, 0.75, 1) & 0 \le x \le 0.3 \\ (0.125, 0, 0.1) & 0.3 < x \le 1 \end{cases} \tag{24}$$

Transmissive boundary conditions are applied at $x = 0$ and 1, while the top and bottom boundaries are treated as solid walls. Two groups of point clouds namely regular and irregular distribution are utilised. On the coarsest level, the domain is uniformly covered by 41×11 points or irregularly discretised with 402 points as shown in Figure 13. Refinement is then carried out to increase the number of points.

The exact solution is produced by the computer program listed in Chapter 4 of Toro's book [21]. The program utilises an iterative guess-correction (Newton-Raphson) strategy to find the exact solution. The flow is computed with fixed time step $\Delta t = 10^{-4}$ until $t = 0.2$. The density contours on the finest level of cloud are shown in Figure 14. Distribution of the density and internal energy along the mid-section of the tube are presented in Figure 15 and 16, respectively. The L_2 norms of error for density and internal energy between the numerical solution and exact result are listed in Table 3. From the corresponding logarithmic scale plots depicted in Figure 17, it is easily observed

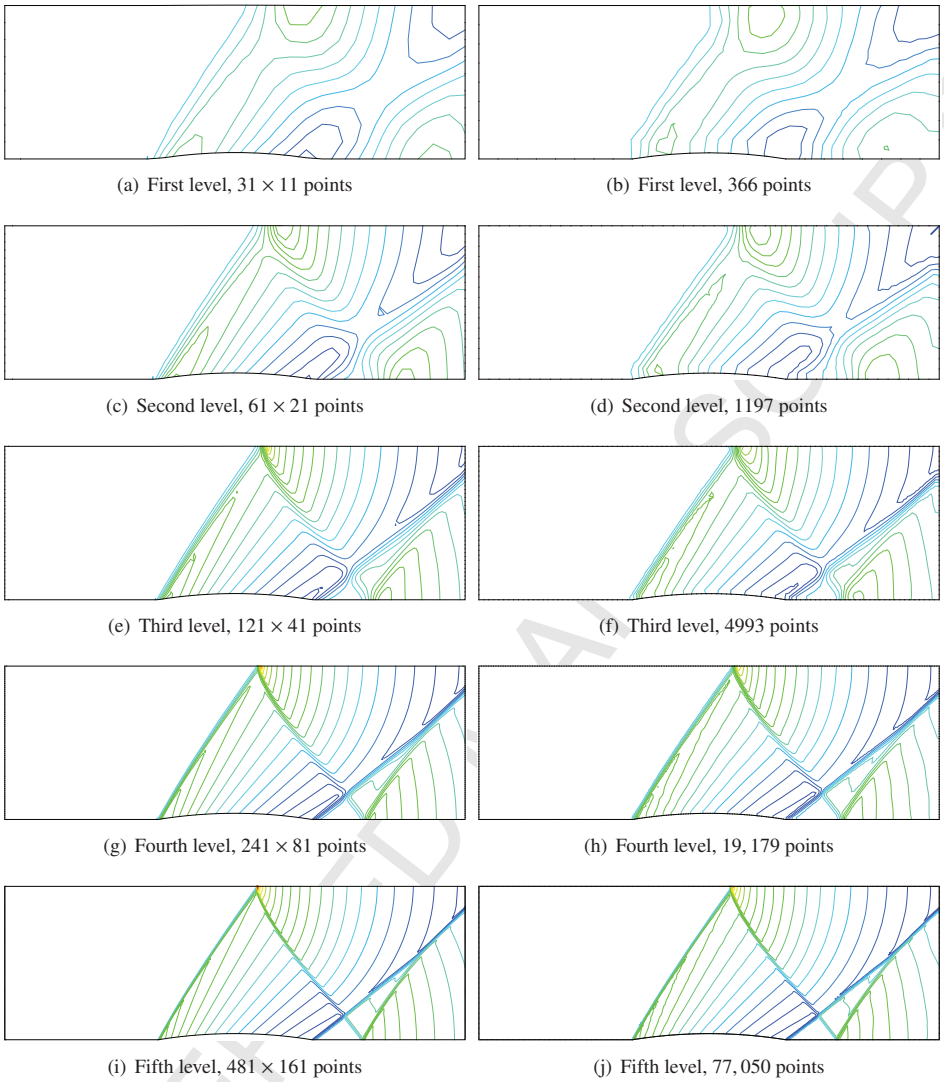
$$\|e\|_2 \approx \mathrm{O}(h), \quad h \to 0 \tag{25}$$

where h represents the averaged point cloud step size. This indicates the numerical solution approaches to the exact result in the limit.



(a) Regular distribution, 31×11 points (b) Irregular distribution, 366 points

Figure 8: Point distribution on the coarsest level of cloud for a channel with a 4% circular bump.



(a) First level, 31×11 points (b) First level, 366 points

(c) Second level, 61×21 points (d) Second level, 1197 points

(e) Third level, 121×41 points (f) Third level, 4993 points

(g) Fourth level, 241×81 points (h) Fourth level, 19, 179 points

(i) Fifth level, 481×161 points (j) Fifth level, 77, 050 points

Figure 9: Pressure contours for supersonic flow ($M_{\text{inlet}} = 1.4$) in a channel with a 4% circular bump ($p_{\min} = 0.7$, $p_{\max} = 2.2$, $\Delta p = 0.071$). Left: regular distribution, Right: irregular distribution.

Table 3: L_2 norm errors for density and internal energy (N_p: number of points, e_ρ: error for density, e_e: error for internal energy).

		Regular distribution			Irregular distribution		
Level	h	N_p	$\|\|e_\rho\|\|_2$	$\|\|e_e\|\|_2$	N_p	$\|\|e_\rho\|\|_2$	$\|\|e_e\|\|_2$
1	1/40	41×11	5.32×10^{-3}	2.88×10^{-2}	402	6.33×10^{-3}	3.53×10^{-2}
2	1/80	81×21	2.70×10^{-3}	1.67×10^{-2}	1,588	3.22×10^{-3}	2.04×10^{-2}
3	1/160	161×41	1.49×10^{-3}	1.04×10^{-2}	6,169	2.05×10^{-3}	1.34×10^{-2}
4	1/320	321×81	8.37×10^{-4}	5.41×10^{-3}	24,795	1.01×10^{-3}	6.94×10^{-3}
5	1/640	641×161	4.25×10^{-4}	2.65×10^{-3}	102,768	5.06×10^{-4}	3.36×10^{-3}

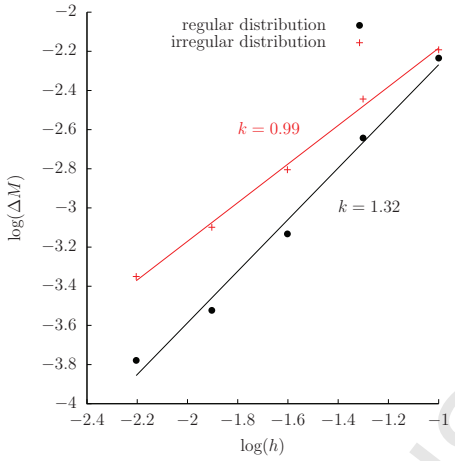
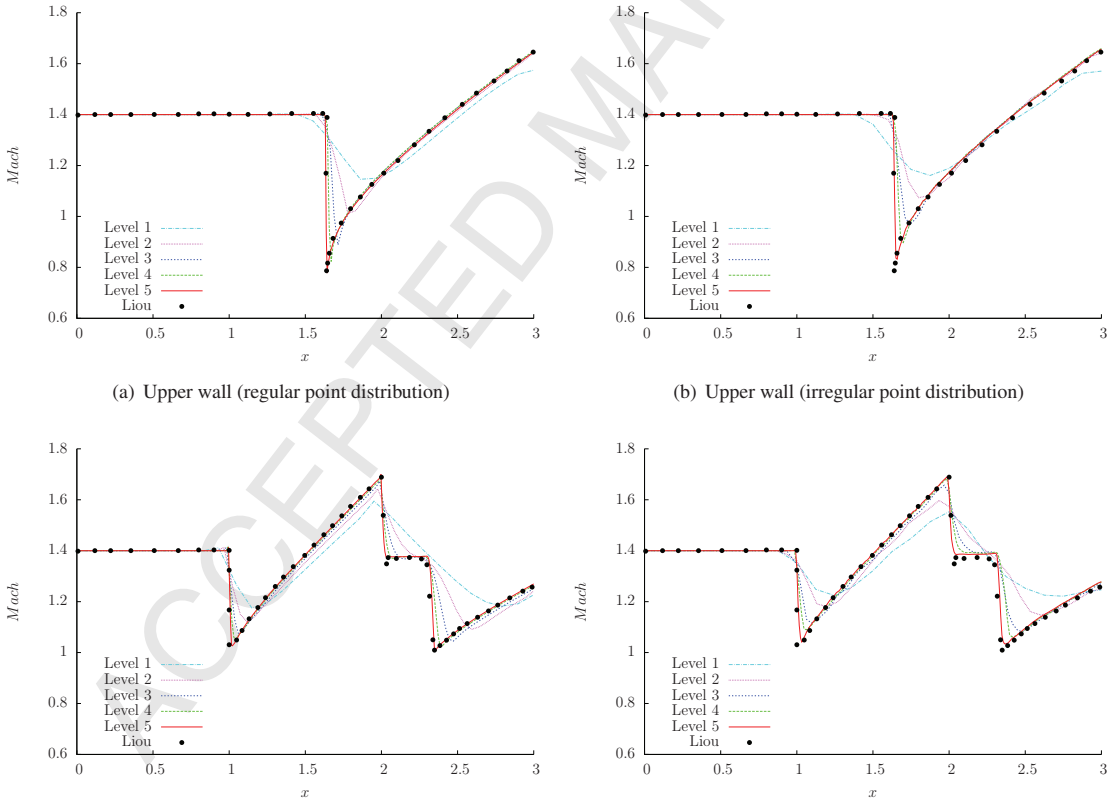


Figure 10: Mass flow conservation on different levels of point clouds (k is the slope of the best fit line).



(a) Upper wall (regular point distribution)

(b) Upper wall (irregular point distribution)

(c) Lower wall (regular point distribution)

(d) Lower wall (irregular point distribution)

Figure 11: Wall Mach number distribution (Left: regular point distribution, Right: irregular point distribution). The black dot is Liou's result [42].

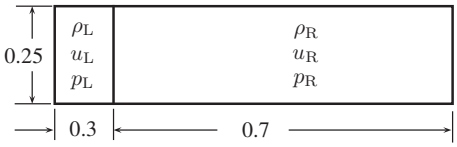
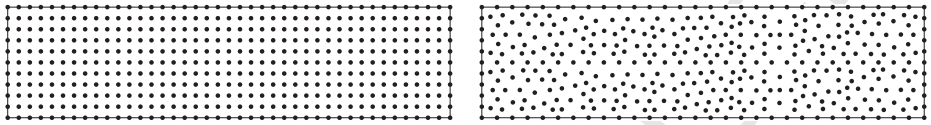
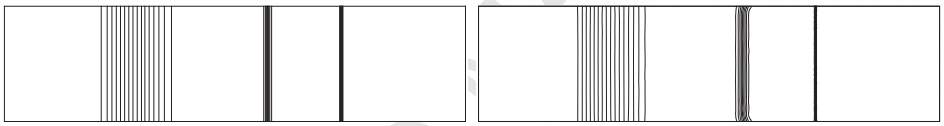


Figure 12: Initial condition setup for the gas shock tube.



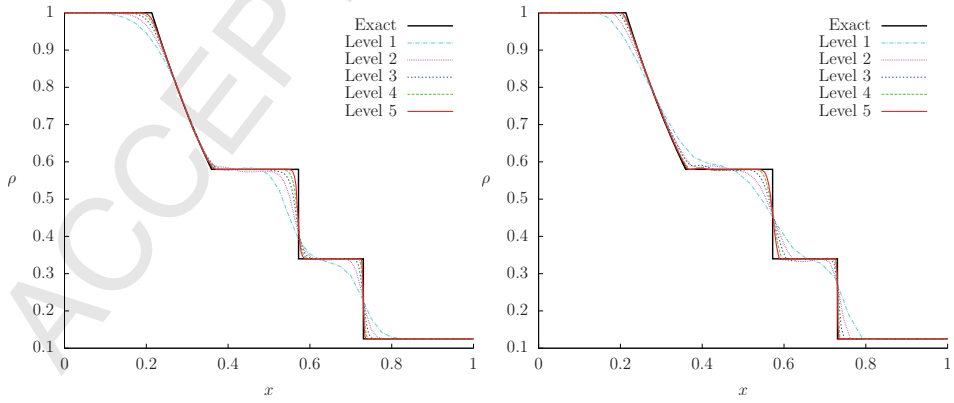
(a) Regular distribution, 41×11 points (b) Irregular distribution, 402 points

Figure 13: Point distribution on the coarsest level of cloud.



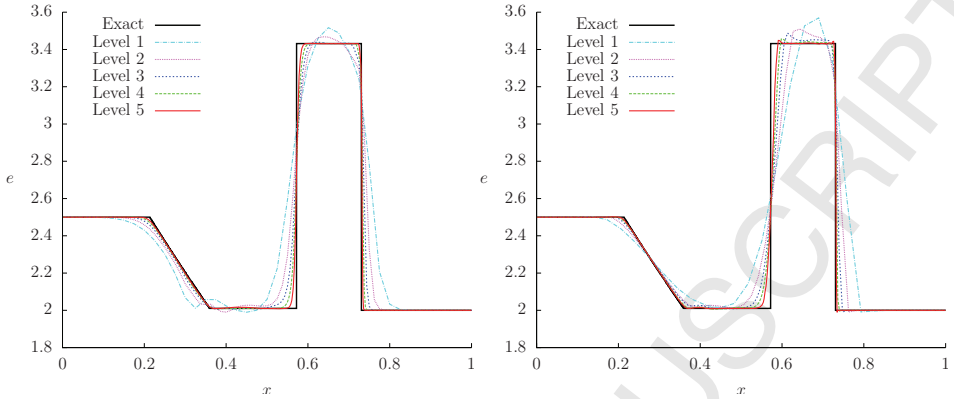
(a) Regular distribution, 640×160 points (b) Irregular distribution, $102,768$ points

Figure 14: Density contours on the finest level of cloud ($\rho_{\min} = 0.14$, $\rho_{\max} = 0.98$, $\Delta\rho = 0.02625$).



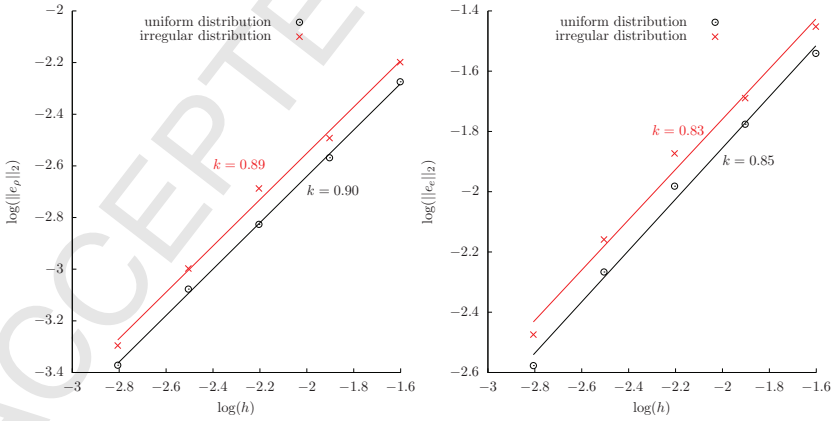
(a) Regular distribution (b) Irregular distribution

Figure 15: Density on mid-section of the tube.



(a) Regular distribution (b) Irregular distribution

Figure 16: Internal energy on mid-section of the tube.



(a) Density (b) Internal energy

Figure 17: The L_2 norm of errors on different levels of point clouds (k is the slope of the best fit line).

5.1.4. Incident shock past a cylinder

This case was proposed by Luo et al. [43], in which a shock wave of shock Mach number $M_s = 2$ is moving from the left side to the right side and passing a cylinder in a rectangular tube. The length of the tube is 6 and its height is 3. The cylinder of radius $r = 0.5$ is located at the centre $(0, 0)$ of the tube. The number of the points distributed in the domain is $45,298$. The flow is advanced to $t = 1.445$ with the fixed time step $\Delta t = 0.001$. Three snapshots of the flow field at $t = 0.8$, 1.1 and 1.4 are illustrated in Figure 18. Time history of pressure on the cylinder is plotted in Figure 19, a satisfactory agreement between the present result and Luo's [43] solution is observed.

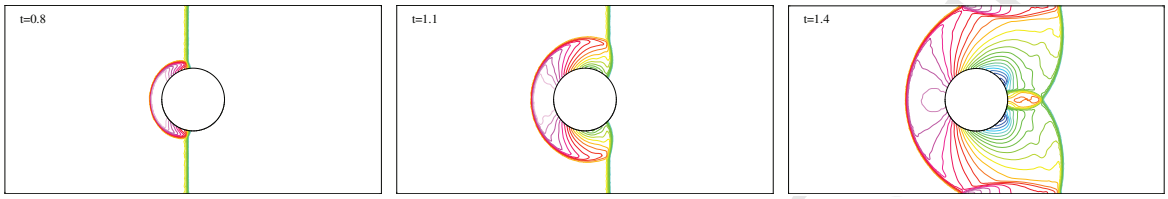


Figure 18: Snapshots of density contours in the domain.

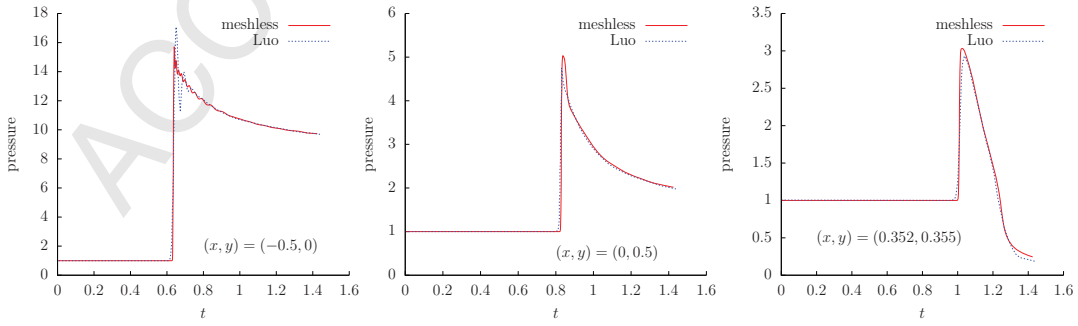


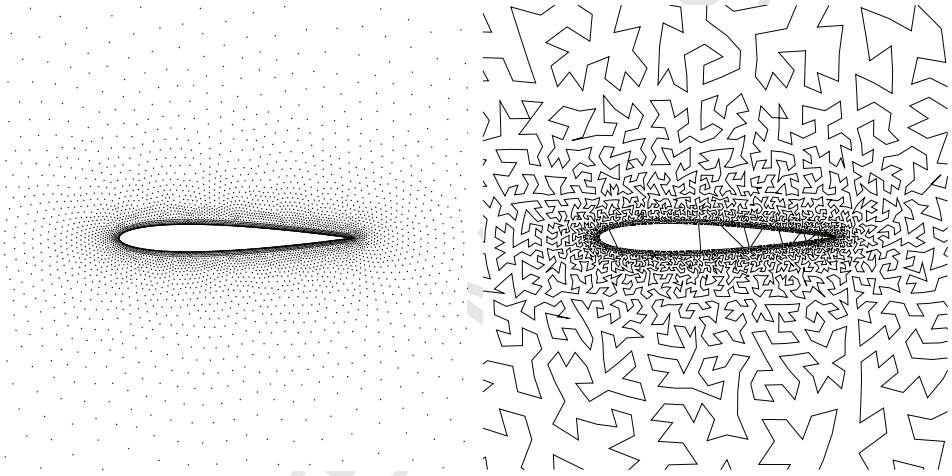
Figure 19: Time history of pressure on the cylinder (the dashed line is Luo's result [43]).

5.2. Application to external steady flows

For steady compressible flows, the variables (density, velocity and pressure) are initially set up as uniform and constant with the corresponding Mach number and angle of attack. Disturbances in the flow field start from the aerofoil surfaces, where no-penetration boundary condition is applied. On the far field boundary, no-reflection condition is adopted. An unstructured grid based finite volume method (FVM, brief description is provided in 7) using the JST (Jameson-Schmidt-Turkel) scheme [44] is also run on the CPU in a single thread to produce reference results for the following cases.

5.2.1. A single NACA0012 aerofoil

The first test case is the transonic flow over a single NACA0012 aerofoil with the conditions $M_\infty = 0.85$ and $\alpha = 1^o$. A number of 5,557 random points are distributed around the aerofoil as shown in Figure 20(a). The generated Hilbert SFC for this case is displayed in Figure 20(b). Two strong shocks appear in the flow and they are clearly captured by the meshless solver as shown Figure 21(b). The computed surface pressure coefficient is compared to the FVM solution and Munikrishna's work [45] and a good agreement is illustrated in Figure 21(a).



(a) irregular points (5557) (b) Hilbert curve ($K = 13$)

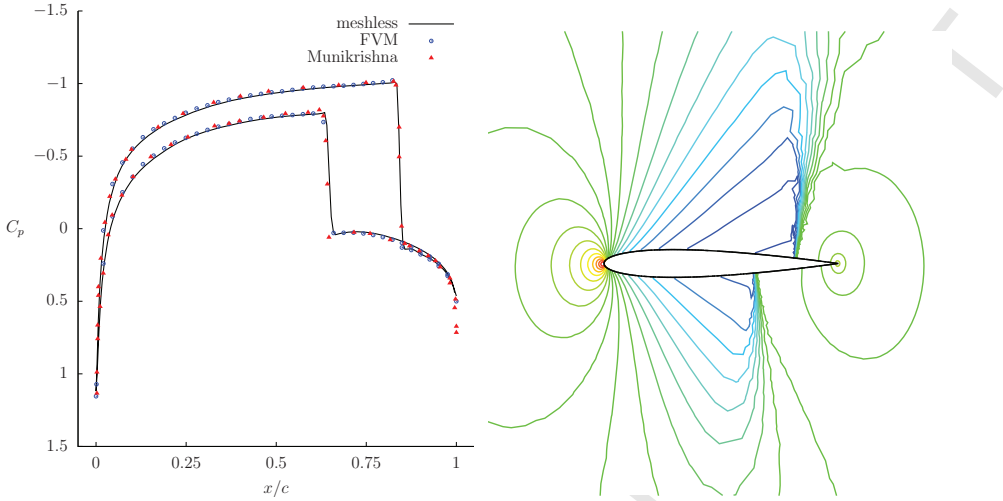
Figure 20: Memory access pattern for a single NACA0012 aerofoil

5.2.2. A single RAE2822 aerofoil

The second case is the flow over a single RAE2822 aerofoil with the conditions $M_\infty = 0.725$ and $\alpha = 2.55^o$. The number of irregular points distributed in the domain is 5482 as shown in Figure 22(a). The Hilbert SFC for this case is displayed in Figure 22(b). Figure 23(b) shows the pressure contours in the flow filed and a shock wave on the upper surface of the aerofoil near the half chord is spotted. The pressure coefficient on the aerofoil surface is illustrated in Figure 23(a). Wang et al. [46] give the highest pressure near the leading edge followed by the FVM and meshless results; meanwhile, a little stronger shock wave close to the half chord is observed. In general, there is a satisfactory agreement between the present result and reference solutions.

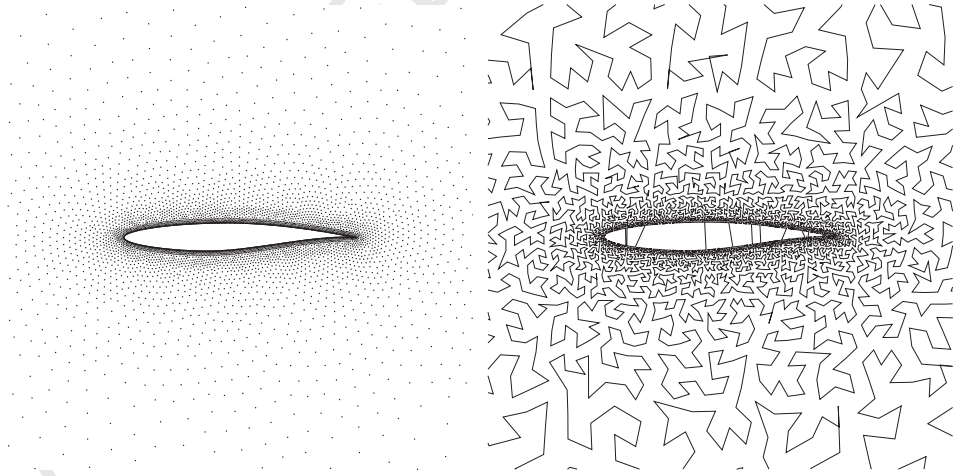
5.2.3. Dual NACA0012 aerofoils

A more complex problem is considered in this case, where two identical NACA0012 aerofoils are misaligned in the domain. The vertical distance between these two aerofoils equals to a half chord and the horizontal discrepancy is also a half chord. Figure 24 gives overviews of the point distributions (9064) and the corresponding generated Hilbert SFC. The region between the upper and lower aerofoils is very similar to a nozzle, where the air shall experience compression and then expansion. The flow conditions for this case are $M_\infty = 0.7$ and $\alpha = 0^o$. Figure 25(b) shows the



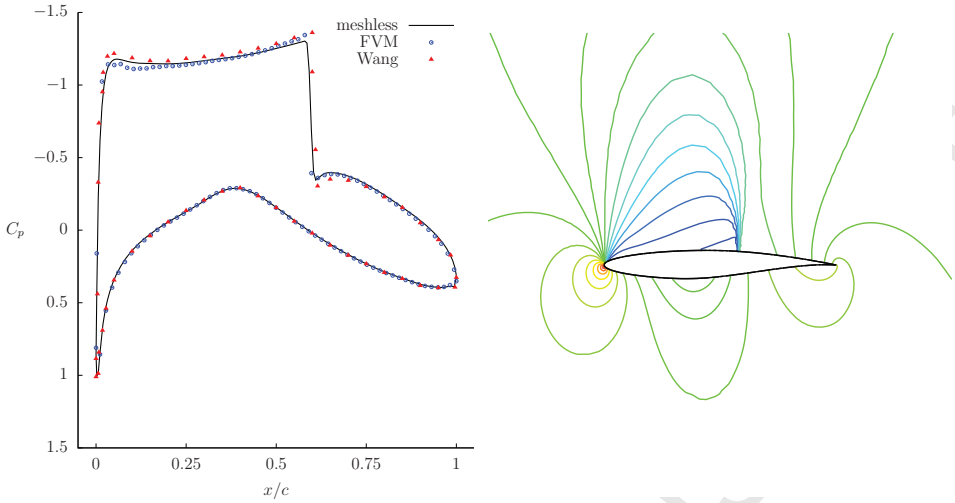
(a) pressure coefficient (b) pressure contours

Figure 21: Transonic flow over a single NACA0012 aerofoil ($M_\infty = 0.85$, $\alpha = 1^o$), the red triangles represent the result of Munikrishna [45].



(a) irregular points (5482) (b) Hilbert curve ($K = 13$)

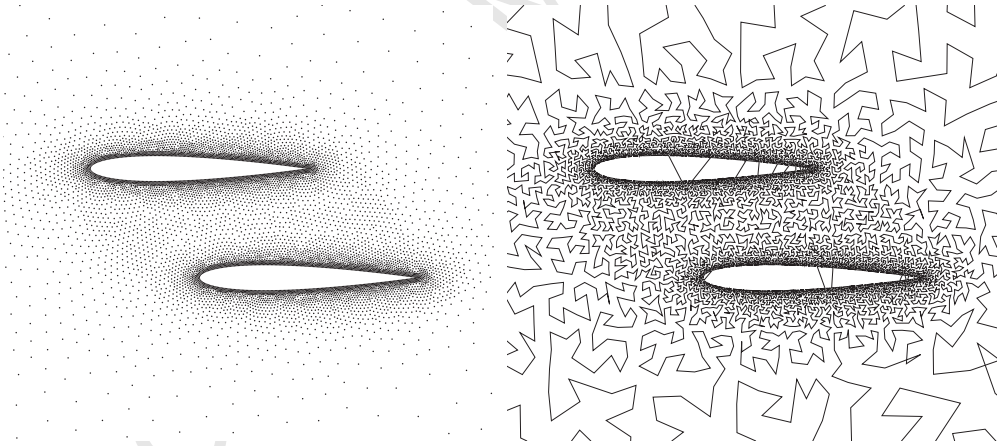
Figure 22: Memory access pattern for a RAE2822 aerofoil



(a) pressure coefficient (b) pressure contours

Figure 23: Transonic flow over a single RAE2822 aerofoil($M_\infty = 0.725$, $\alpha = 2.55^o$), the red triangles represent the result of Wang [46].

pressure contours in the flow field, where a strong shock wave in the nozzle region is clearly captured. Comparison with the FVM solution and Kirshman & Liu's result [20] in Figure 25(a) indicates that the meshless method produces a satisfactory result considering the shock wave location and strength.

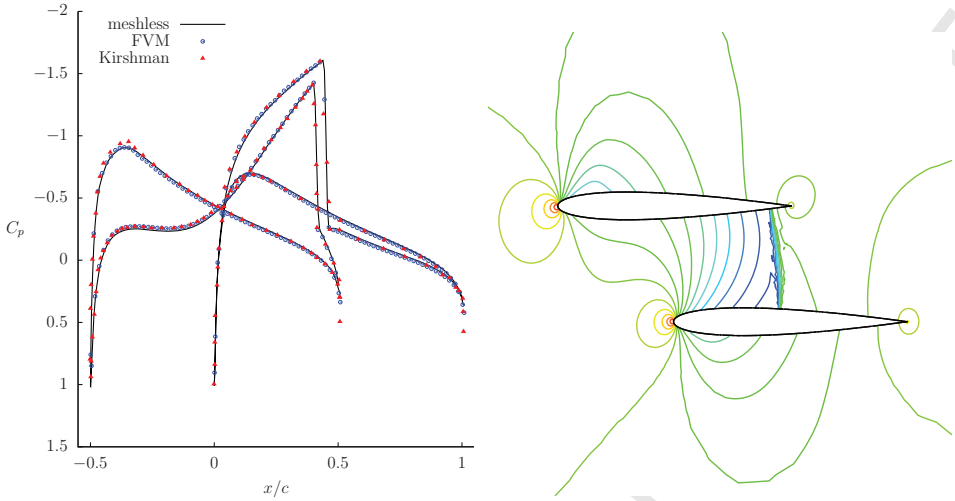


(a) irregular points (9064) (b) Hilbert curve ($K = 13$)

Figure 24: Memory access pattern for two NACA0012 aerofoils

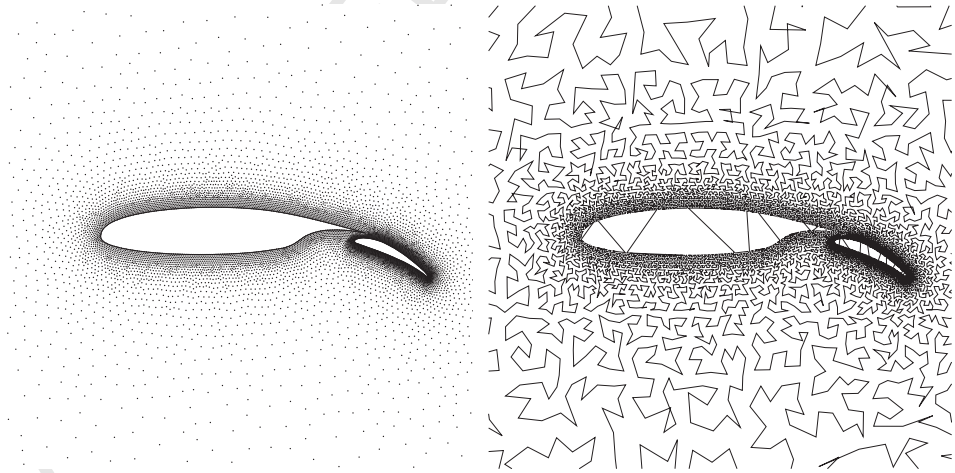
5.2.4. A GA(W)-1 two-element aerofoil

The subsonic flow over a GA(W)-1 two-element aerofoil is considered for this case. The flow conditions are $M_\infty = 0.2$ and $\alpha = 10^o$. A number of $10,849$ points are scattered in the domain as shown in Figure 26(a) and the corresponding Hilbert curve is shown in Figure 26(b). The pressure contours in the flow field are illustrated in Figure 27(b). The computed pressure coefficient on the aerofoil surface agrees well with the experimental data (Wentz and Seetharam [47]) as displayed in Figure 27(a).



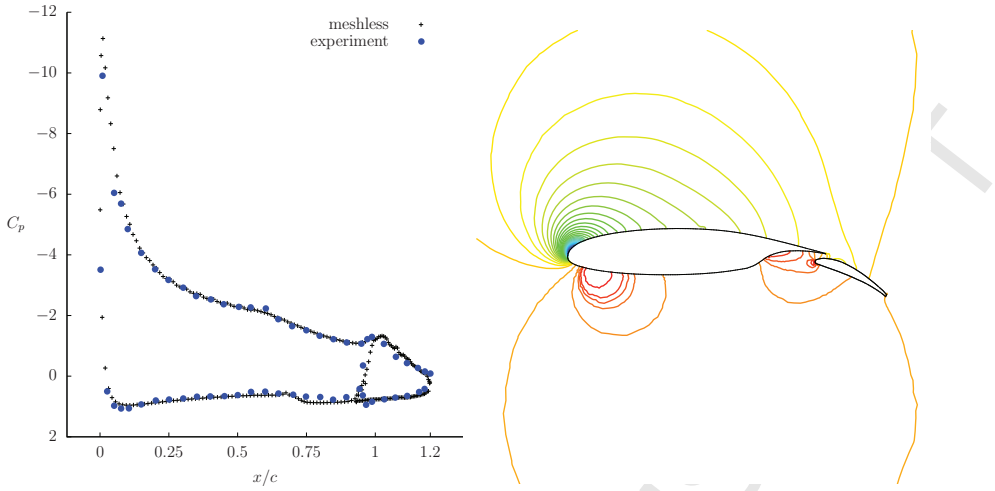
(a) pressure coefficient (b) pressure contours

Figure 25: Transonic flow over two NACA0012 aerofoils ($M_\infty = 0.7$, $\alpha = 0^o$), the red triangles represent the result of Kirshman & Liu [20].



(a) irregular points (10849) (b) Hilbert curve ($K = 15$)

Figure 26: Memory access pattern for a GA(W)-1 two-element aerofoil

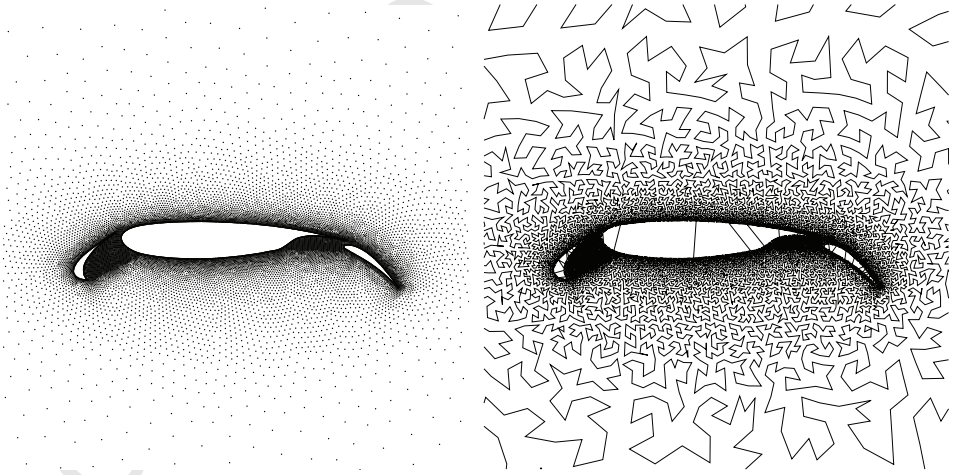


(a) pressure coefficient (b) Pressure contours

Figure 27: Subsonic flow over a GA(W) two-element aerofoil ($M_\infty = 0.2$, $\alpha = 10^o$), the blue dots represent the work of Wentz & Seetharam [47].

5.2.5. *A three-element aerofoil*

The flow conditions for this case are $M_\infty = 0.21$ and $\alpha = 10^o$. A number of 15, 198 points are scattered in the domain as shown in Figure 28(a) and the corresponding Hilbert curve is shown in Figure 28(b). The Mach number contours in the flow field are illustrated in Figure 29(b). The computed pressure coefficient around the aerofoil surface is compared to the FVM solution in Figure 29(a). A satisfactory agreement between the results is obtained.

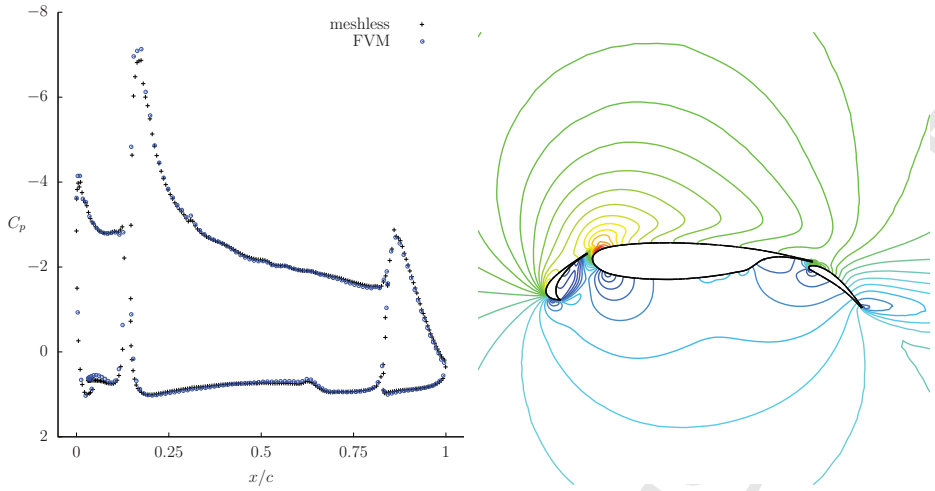


(a) irregular points (15198) (b) Hilbert curve ($K = 14$)

Figure 28: Memory access pattern for a three-element aerofoil

5.2.6. *Performance analysis*

In order to evaluate the efficiency of the meshless flow solver on the CPU and GPU quantitatively, we fix the Runge-Kutta iteration number to 10, 000 for all the external flow cases. In practice, steady computation can be



(a) pressure coefficient (b) Mach number contours

Figure 29: Subsonic flow over a three-element aerofoil ($M_\infty = 0.21$, $\alpha = 10^o$)

terminated when the residual satisfies the accuracy requirement before reaching the maximum iteration number. Nevertheless, we fix this parameter only for comparison purpose.

FlowDerivative and *FlowFlux* are two key functions in the whole computer program, and more than 90% of the total running time is spent on them. They are used to compute the spatial derivatives of primitive flow variables and flux terms, respectively. In Table 4, we specifically list average performance of these two functions on different compute devices.

Table 4: Performance (GFLOPS) of two key functions *FlowDerivative* and *FlowFlux*.

	FlowDerivative			*FlowFlux*		
Case	CPU	Quadro 2000	Tesla C2075	CPU	Quadro 2000	Tesla C2075
1	0.70	7.08	23.62	1.69	17.94	55.77
2	0.70	7.09	23.21	1.69	18.08	56.17
3	0.71	7.23	25.28	1.67	18.47	59.35
4	0.72	7.31	26.02	1.69	18.37	60.88
5	0.72	7.37	26.41	1.70	18.55	62.75

For every case, we count the wall clock time for the Intel Xeon E5645 CPU, Quadro 2000 GPU and Tesla C2075 GPU. The wall clock time to run the Runge-Kutta iterations for all the cases is plotted in Figure 30(a). The running time on the CPU grows quickly (from around 380s to 1050s) with the increased number of points in the flow field. It might be a challenge to the patience of a researcher who requests a reliable solution in a short period if the number of points increases continuously, while the GPU is able to accomplish the task efficiently. It is also interesting to find that the SFC re-ordering procedure is indeed beneficial to the program performance on the CPU and GPU.

In order to know exactly how much performance improvement we can gain from the GPU computing, we introduce the speedup to compare the running time and it is defined as follows

$$\text{Speedup} = \frac{\text{T}_{\text{CPU,SFC}}}{\text{T}_{\text{GPU}}} \tag{26}$$

where T_{GPU} represents the GPU wall clock time (with or without the SFC re-ordering) and $\text{T}_{\text{CPU,SFC}}$ indicates the CPU wall clock time aided with the SFC re-ordering technique. This is a very strict criterion to gauge the speedup, since

using CPU wall time without SFC re-ordering will give a higher value. The corresponding calculated result for the speedup is depicted in Figure 30(b). It is easy to notice that impressive speedups are achieved on the two GPUs for all the test cases. On the Quadro 2000 graphic card, the speedup is around 7× ∼ 9× without point re-ordering and it is increased to around 10× ∼ 12× when the points are re-numbered with the SFC. On the Tesla C2075 graphic card, a speedup around 30× is obtained for random point distribution and the SFC re-ordering boosts the speedup further (nearly 40× for the three-element aerofoil case).

6. Conclusions

A GPU implementation of a meshless method for solving compressible flow problems is presented in this paper. Crucial serial-computing Fortran subroutines are successfully ported to the GPU with the CUDA C programming model. The Hilbert SFC is employed to improve the data locality for randomly distributed points. The fundamental numerical method is firstly detailed evaluated to assess the accuracy and conservation property. It is then applied to solve steady compressible flows over aerodynamic configurations. Numerical analysis reveals that the program performance is greatly improved with more than 10× speedup on the Quadro 2000 GPU and over 30× speedup on the Tesla C2075 GPU. This exhibits the potential of the GPU based meshless method for more complicated flow problems.

In future work, unsteady fluid-structure interaction problems will be considered and demonstrated using the current GPU meshless flow solver. In addition, the developed method will be implemented for multi-objective optimisation problems using evolutionary algorithms coupled to game strategies for various real-world design problems.

Acknowledgements

The first author acknowledges with gratitude the CUDA training courses provided by HECToR (High-End Computing Terascale Resource) and Daresbury Laboratory, UK. The second author would like to acknowledge the support by Department of Mathematical Information Technology, University of Jyväskylä, Finland.

7. Finite volume method

Here we give a brief description of the finite volume method used in the present work to produce reference results for steady compressible flows. The integral form of the Euler equations (1) can be written as

$$\int_{\Omega}\left(\frac{\partial \mathbf{U}}{\partial t}+\frac{\partial \mathbf{E}}{\partial x}+\frac{\partial \mathbf{F}}{\partial y}\right)\mathrm{d}\Omega=0 \tag{27}$$

The volume integration of the flux terms can be converted into closed surface integration by the Green-Gauss theorem and then we obtain the following form

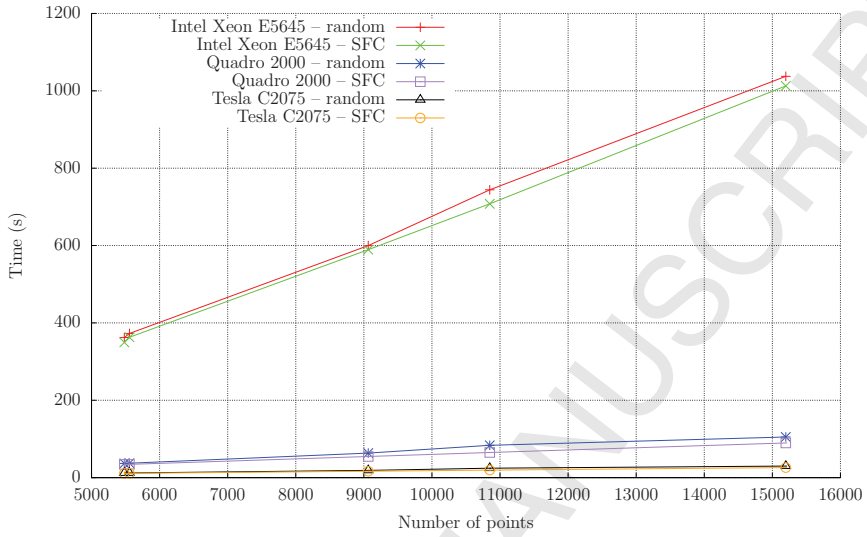
$$\frac{\partial}{\partial t}\int_{\Omega}\mathbf{U}\mathrm{d}\Omega+\oint_{\partial\Omega}\left(\mathbf{E}n_x+\mathbf{F}n_y\right)\mathrm{d}S=0 \tag{28}$$

where $\vec{n}=(n_x,n_y)$ is a unit outward normal vector of the boundary S ($S=\partial\Omega$). This equation is usually referred as the global conservation law [21]. The domain Ω is divided into a number of discrete control volumes

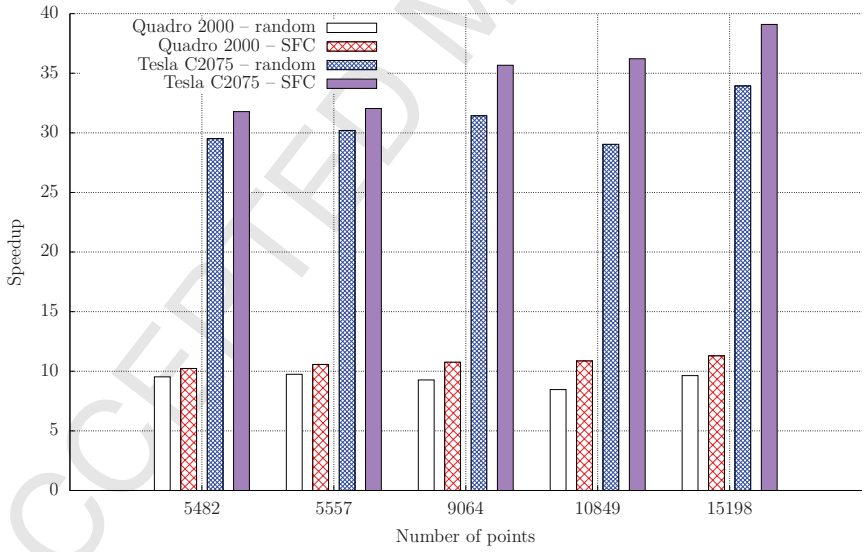
$$\Omega=\cup\Omega_i \qquad (i=1,2,\ldots,N) \tag{29}$$

and generally the summation of each volume must be equal to the volume of the whole domain (the overlapping volumes are not considered here)

$$\Omega=\sum_{i=1}^{N}\Omega_i \tag{30}$$



(a) wall clock time



(b) speedup of run time (the single thread CPU program is aided with the SFC reordering)

Figure 30: Benchmark of the meshless flow solver on the CPU and GPU

where N is the total number of control volumes in the domain. Eq. (28) is required to be satisfied in every control volume, the local conservation law for any control volume Ω_i is

$$\frac{\partial}{\partial t}\int_{\Omega_i}\mathbf{U}\mathrm{d}\Omega+\oint_{\partial\Omega_i}\left(\mathbf{E}n_x+\mathbf{F}n_y\right)\mathrm{d}S=0 \tag{31}$$

For steady flows, Ω_i does not vary with time, as we apply the JST scheme [44] with FVM, the above equation can be rewritten into the following form

$$\Omega_i\frac{\partial\mathbf{U}_i}{\partial t}+(\mathbf{Q}_i-\mathbf{D}_i)=0 \tag{32}$$

where $\mathbf{Q}_i$ is the inviscid flux computed as

$$\mathbf{Q}_i=\sum_{j=1}^{M_i}\left(\mathbf{E}n_x+\mathbf{F}n_y\right)_{ij}S_{ij} \tag{33}$$

in which M_i is the total number of boundary edges of Ω_i, and $\mathbf{D}_i$ is the artificial dissipation used to stabilise the computation (more detail of this scheme can be found in reference [44]).

[1] J. T. Batina, A gridless Euler/Navier-Stokes solution algorithm for complex-aircraft applications, in: 31st Aerospace Sciences Meeting & Exhibit, 1993, AIAA Paper 93-0333.

[2] E. Oñate, S. Idelsohn, O. Zienkiewicz, R. Taylor, A finite point method in computational mechanics. Applications to convective transport and fluid flow, International Journal for Numerical Methods in Engineering 39 (22) (1996) 3839 – 3866.

[3] R. Löhner, C. Sacco, E. Oñate, S. Idelsohn, A finite point method for compressible flow, International Journal for Numerical Methods in Engineering 53 (8) (2002) 1765 – 1779. doi:10.1002/nme.334.

[4] D. Sridar, N. Balakrishnan, An upwind finite difference scheme for meshless solvers, Journal of Computational Physics 189 (1) (2003) 1–29. doi:10.1016/S0021-9991(03)00197-9.

[5] Z. Ma, H. Chen, C. Zhou, A study of point moving adaptivity in gridless method, Computer Methods in Applied Mechanics and Engineering 197 (21-24) (2008) 1926–1937. doi:10.1016/j.cma.2007.12.012.

[6] E. Ortega, E. Oñate, S. Idelsohn, A finite point method for adaptive three-dimensional compressible flow calculations, International Journal for Numerical Methods in Fluids 60 (9) (2009) 937 – 971. doi:10.1002/fld.1892.

[7] K.-y. E. Chiu, Q. Wang, R. Hu, A. Jameson, A conservative mesh-free scheme and generalized framework for conservation laws, SIAM Journal on Scientific Computing 34 (6) (2012) A2896–A2916. doi:10.1137/110842740.

[8] R. Löhner, E. Oñate, An advancing front point generation technique, Communications in Numerical Methods in Engineering 14 (12) (1998) 1097–1108.

[9] R. Löhner, E. Oñate, A general advancing front technique for filling space with arbitrary objects, International Journal for Numerical Methods in Engineering 61 (12) (2004) 1977–1991.

[10] K. Morinishi, An implicit gridless type solver for the Navier-Stokes equations, Computational Fluid Dynamics Journal Special Issue (2001) 551–560.

[11] H. Q. Chen, C. Shu, An efficient implicit mesh-free method to solve two-dimensional compressible Euler equations, International Journal of Modern Physics C 16 (2005) 439–454. doi:10.1142/S0129183105007327.

[12] A. Katz, A. Jameson, Multicloud: Multigrid convergence with a meshless operator, Journal of Computational Physics 228 (14) (2009) 5237 – 5250. doi:DOI:10.1016/j.jcp.2009.04.023.

[13] NVIDIA, CUDA C programming guide (2012).
URL http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[14] A. P. Engsig-Karup, M. G. Madsen, S. L. Glimberg, A massively parallel GPU-accelerated model for analysis of fully nonlinear free surface waves, International Journal for Numerical Methods in Fluids 70 (1) (2012) 20–36. doi:10.1002/fld.2675.

[15] A. Corrigan, F. F. Camelli, R. Löhner, J. Wallin, Running unstructured grid-based CFD solvers on modern graphics hardware, International Journal for Numerical Methods in Fluids 66 (2) (2011) 221–229. doi:10.1002/fld.2254.

[16] I. Kampolis, X. Trompoukis, V. Asouti, K. Giannakoglou, CFD-based analysis and two-level aerodynamic optimization on graphics processing units, Computer Methods in Applied Mechanics and Engineering 199 (2010) 712 – 722. doi:10.1016/j.cma.2009.11.001.

[17] V. G. Asouti, X. S. Trompoukis, I. C. Kampolis, K. C. Giannakoglou, Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on Graphics Processing Units, International Journal for Numerical Methods in Fluids 67 (2) (2011) 232–246. doi:10.1002/fld.2352.

[18] W. Ran, W. Cheng, F. Qin, X. Luo, GPU accelerated CESE method for 1D shock tube problems, Journal of Computational Physics 230 (24) (2011) 8797 – 8812. doi:10.1016/j.jcp.2011.08.026.

[19] E. Ortega, E. Oñate, S. Idelsohn, R. Flores, Comparative accuracy and performance assessment of the finite point method in compressible flow problems, Computers & Fluids 89 (0) (2014) 53 – 65. doi:http://dx.doi.org/10.1016/j.compfluid.2013.10.024.

[20] D. J. Kirshman, F. Liu, A gridless boundary condition method for the solution of the Euler equations on embedded Cartesian meshes with multigrid, Journal of Computational Physics 201 (1) (2004) 119 – 147. doi:10.1016/j.jcp.2004.05.006.

[21] E. Toro, Riemann solvers and numerical methods for fluid dynamics: a practical introduction, Springer, 1997.

[22] A. Harten, High resolution schemes for hyperbolic conservation laws, Journal of Computational Physics 135 (2) (1997) 260 – 278. doi:10.1006/jcph.1997.5713.

[23] P. Batten, N. Clarke, C. Lambert, D. Causon, On the choice of wavespeeds for the HLLC Riemann solver, SIAM Journal on Scientific Computing 18 (6) (1997) 1553–1570.

[24] P. Batten, M. A. Leschziner, U. C. Goldberg, Average-state Jacobians and implicit methods for compressible viscous and turbulent flows, Journal of Computational Physics 137 (1997) 38–78.

[25] E. F. Toro, M. Spruce, W. Speares, Restoration of the contact surface in the HLL- Riemann solver, Tech. rep., Cranfield Institute of Technology (1992).

[26] Khronos, The open standard for parallel programming of heterogeneous systems (2013).
URL http://www.khronos.org/opencl/

[27] Z. Ma, Research of adaptive meshfree and hybridized mesh/meshfee methods, Ph.D. thesis, Nanjing University of Aeronautics and Astronautics (March 2008).

[28] The Portland Group, CUDA Fortran Programming Guide and Reference (2013).
URL http://www.pgroup.com/doc/pgicudaforug.pdf

[29] T. Chilimbi, M. Hill, J. Larus, Making pointer-based data structures cache conscious, Computer 33 (12) (2000) 67–74. doi:10.1109/2.889095.

[30] H. Sagan, Space-filling curves, Springer-Verlag, 1994.

[31] H. T. Vo, C. T. Silva, L. F. Scheidegger, V. Pascucci, Simple and efficient mesh layout with space-filling curves, Journal of Graphics Tools 16 (1) (2012) 25–39. doi:10.1080/2151237X.2012.641828.

[32] M. J. Aftosmis, M. J. Berger, S. M. Murman, Applications of space-filling curves to Cartesian methods for CFD, in: 42nd Aerospace Sciences Meeting and Exhibit, 2004.

[33] F. Alauzet, A. Loseille, On the use of space filling curves for parallel anisotropic mesh adaptation, in: B. Clark (Ed.), Proceedings of the 18th International Meshing Roundtable, Springer Berlin Heidelberg, 2009, pp. 337–357. doi:10.1007/978-3-642-04319-2_20.

[34] J. W. Cannon, W. P. Thurston, Group invariant peano curves, Geometry & Topology 11 (2007) 1315–1355.

[35] D. Hilbert, Ueber die stetige Abbildung einer Line auf ein Flächenstück, Mathematische Annalen 38 (3) (1891) 459–460. doi:10.1007/bf01199431.

[36] B. B. Mandelbrot, The Fractal Geometry of Nature, 1982, Ch. Harnessing the Peano Monster Curves.

[37] G. Peano, Sur une courbe, qui remplit toute une aire plane, Mathematische Annalen 36 (1890) 157–160. doi:10.1007/BF01199438.

[38] A. Butz, Alternative algorithm for Hilbert's space-filling curve, Computers, IEEE Transactions on C-20 (4) (1971) 424–426. doi:10.1109/T-C.1971.223258.

[39] C. A. R. Hoare, Algorithm 64: Quicksort, Commun. ACM 4 (7) (1961) 321–322. doi:10.1145/366622.366644.

[40] R. Löhner, Applied computational fluid dynamics techniques: An Introduction Based on Finite Element Methods, Wiley, 2008.

[41] C.-W. Shu, Lecture Notes in Mathematics, Springer, 1998, Ch. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws, pp. 325–432.

[42] M. S. Liou, An extended Lagrangian method, Journal of Computational Physics 118 (1995) 294–309.

[43] H. Luo, J. D. Baum, R. Löhner, A hybrid Cartesian grid and gridless method for compressible flows, Journal of Computational Physics 214 (2) (2006) 618 – 632. doi:10.1016/j.jcp.2005.10.002.

[44] A. Jameson, W. Schmidt, E. Turkel, Numerical solutions of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes, AIAA paper 81 (1981) 1259.

[45] N. Munikrishna, On viscous flux discretization procedures for finite volume and meshless solvers, Ph.D. thesis, Indian Institute of Science (2007).

[46] H. Wang, H.-Q. Chen, J. Periaux, A study of gridless method with dynamic clouds of points for solving unsteady CFD problems in aerodynamics, International Journal for Numerical Methods in Fluids 64 (1) (2010) 98–118. doi:10.1002/fld.2145.

[47] W. H. Wentz, H. C. Seetharam, Development of a fowler flap system for a high performance general aviation airfoil, Tech. rep., Wichita State University, NASA CR-2443 (1974).