View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

Procedia Computer Science

Volume 80, 2016, Pages 1291–1300

ICCS 2016. The International Conference on Computational Science

An Accelerated Iterative Linear Solver with GPUs for CFD Calculations of Unstructured Grids

Justin Williams[1], Christian Sarofeen[2], Hau Shan[3], Matthew Conley[4]

[1]*North Carolina A&T State University, Greensboro,NC.*

[23]*Carderock Division, Naval Surface Warfare Center, Bethesda, MD*

[4]*Case Western Reserve University ,Cleveland, OH,*

Justinrome24@gmail.com,

Christian.sarofeen@navy.mil,hua.shan@navy.mil,

mattconley0@gmail.com

Abstract

Computational Fluid Dynamics (CFD) utilizes numerical solutions of Partial Differential Equations (PDE) on discretized volumes. These sets of discretized volumes, grids, can often contain tens of millions, or billions of volumes. The analysis time of these large unstructured grids can take weeks to months to complete even on large computer clusters. For CFD solvers utilizing the Finite Volume Method (FVM) with implicit time stepping or a segregated pressure solver, a large portion of the computation time is spent solving a large linear system with a sparse coefficient matrix. In an effort to improve the performance of these CFD codes, in effect decreasing the time to solution of engineering problems, a conjugate gradient solver for a Finite Volume Method Solver Graphics Processing Units (GPU) was implemented to solve a model Poisson's equation. Utilizing the improved memory throughput of NVIDIA's Tesla K20 GPU a 2.5 times improvement was observed compared to a parallel CPU implementation on all 10 cores of an Intel Xeon E5-2670 v2. The parallel CPU implementation was constructed using the open source CFD toolbox, Open-FOAM.

Keywords: Computational Fluid Dynamics, Graphics Processing Units (GPU), Finite Volume Method, Unstructured Grids, Computational Flow and Transport, Modeling, Simulations, Algorithms

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

© The Authors. Published by Elsevier B.V.

doi:10.1016/j.procs.2016.05.504

1 Introduction

The Finite Volume Method (FVM) is a method often utilized in Computational Fluid Dynamics (CFD) to approximate a solution to the governing Partial Differential Equations (PDE). Accurate solutions to fluid dynamics problems involving complex geometries often require the use of large unstructured 3-Dimensional grids. Due to the large number of small finite volumes (regular polyhedron) that make up the grid, CFD analyses can take weeks or months to complete on supercomputers. This long timeframe can often limit their use in engineering design efforts. Furthermore, unstructured grids lead to non-uniform memory access patterns creating memory bottlenecks in most of their implementations.

The improved computational performance of Graphics Processing Units (GPU) compared to their Central Processing Unit (CPU) counterpart make them an ideal candidate for accelerating scientific algorithms. For many years now, GPUs have been successfully utilized to accelerate algorithms that allow for fine grained parallelization. For example, GPUs have been used for algorithms involved in medical imaging [1], computational finance [2], and bioinformatics [3]. However, GPUs require strict memory access patterns to achieve high memory throughput [4] and FVM on unstructured grids are riddled with non-uniform memory access patterns making acceleration of unstructured CFD methods difficult [5]. Higher order Finite Element Methods (FEM), another method frequently used for CFD, have higher computation to memory access ratio and have therefore been successfully accelerated utilizing GPUs [6]. However, these algorithms hide their poor memory access patterns with the high computational requirements of solving a basis function on many points of each polyhedral cell. Although, higher order FEM do have improved accuracy, this accuracy often comes at a steep price in computational performance and is often not needed in engineering problems.

Two FVMs for CFD that are of particular interest are those with implicit time stepping for their improved temporal accuracy [7], and those utilizing a segregated pressure method for their improved performance in many situations [8] [9]. Both of these methods require solving a large sparse linear system where the structure of the coefficient matrix is derived from the connectivity of the computational grid. The solution to this linear system is the typical bottleneck for these methods and is therefore the primary concern of this paper [10] [11].

Presented herein is a GPU based conjugate gradient solver for FVMs. To test the performance of this iterative linear method, a model Poisson's equation is solved. This method provides a benchmark applicable to what would be seen in a full CFD implementation. Performance results of the GPU implementation is compared with a baseline parallel CPU implementation in the open source CFD toolbox, OpenFOAM. Both implementations are a standard conjugate gradient method. Performance of the K20X is observed at a maximum of 2.5 times that of all 10 cores of an Intel Xeon E5-2670 v2.

2 Methods

Section 2.01 Construction of the Linear System

A linear system is a mathematical model of a system based on the use of a linear operator. The linear system used in this research is the sparse matrix equation $A * x = b$ like that shown in Figure 1. For the work presented herein the system is assumed to have a unique solution, meaning that given a stable iterative solution method it will march towards the exact solution. Sparse matrix equation implies that many of the elements in the A matrix are zero, to the extent that storage and computational savings can be seen by not explicitly processing the zero entries. Operations using non- sparse matrix structures and algorithms can be slow and with inefficient memory storage for matrices with many zeroes.

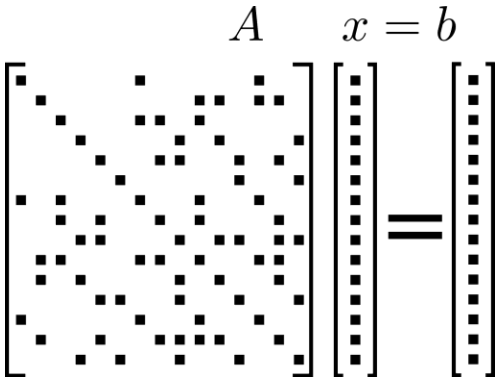

Figure 1: Matrix equation, where A is an n x n Sparse Matrix, x and b are column vectors.

For the work presented herein the system is assumed to have a unique solution, meaning that given a stable iterative solution method it will march towards the exact solution. Sparse matrix equation implies that many of the elements in the *A* matrix are zero, to the extent that storage and computational savings can be seen by not explicitly processing the zero entries. Operations using non-sparse matrix structures and algorithms can be slow and with inefficient memory storage for matrices with many zeroes.

A model Poisson's equation is constructed and solved with the FVM. Its solution leads to a sparse matrix equation that is then solved with the Conjugate Gradient Method. The model Poisson's equation used to construct the linear system is based on assuming a solution,

$$\varphi = x^2(1-x)y^2(1-y)z^2(1-z). \quad (1)$$

From this, a Poisson's equation was derived as

$$\nabla^2\varphi = 2[(1-3x)y^2(1-y)z^2(1-z) + x^2(1-x)(1-3y)z^2(1-z) + x^2(1-x)y^2(1-y)(1-3z)]. \quad (2)$$

So that once solved the solution could be compared to the analytic solution (1). Based on (2), a solver was created in OpenFOAM to recover a Finite Volume Method solution. This solution approaches the analytic solution as the grid is refined into smaller and smaller finite volumes. The solution process for this equation starts with the integral form of (2) on a volume Ω,

$$\int_\Omega \nabla^2\varphi \, d\Omega = \int_\Omega f \, d\Omega \quad (3)$$

Where, f is the right hand side of (2). Applying the divergence theorem (3) can be stated a

$$\int_S (\nabla\varphi \cdot \vec{n}) \, dS = \int_\Omega f \, d\Omega \quad (4)$$

Where, S is the surface of the volume Ω and $\vec{n}$ is the surface's normal vector. This can be discretized to all finite volumes of a grid as

$$\sum_f \left(\frac{\partial \varphi}{\partial n}\right)_f S_f = V_e f_e. \qquad (5)$$

Where f are the faces of the finite volume e, S_f is the area vector of face f, $\left(\frac{\partial \varphi}{\partial n}\right)_f$ is evaluated at the face center, V_e is the volume of finite volume e, and f_e is the right hand side of (2) evaluated at the element center. Although there are many methods for evaluating $\left(\frac{\partial \varphi}{\partial n}\right)_f$, this work uses a simple difference of φ from the neighboring finite volume of e, also connected to face f. Evaluating (5) for all finite volumes leads to a large sparse matrix equation similar to that shown in Figure 1. Solving this linear system gives an approximation of the solution (1)

Section 2.02 Computational Approach and the Conjugate Gradient Method

The software for this research effort is derived from the open source CFD software package, OpenFOAM, which has an extensive range of features for FVM of CFD problems. For the GPU implementation a combination of the NVIDIA CUDA library of basic linear algebra subprograms for sparse matrices (cuSPARSE), the CUDA basic linear algebra subprograms (cuBLAS), and some hand written GPU functions were integrated into OpenFOAM with a separately compiled library.

The iterative solver for the system of linear equations is a non-preconditioned Conjugate gradient method as described in [12]. The conjugate gradient method is a widely used iterative solver due to its stability on a wide range of scientific problems. Like many other iterative solvers, it is well suited for sparse linear systems as it does not require any zero entries of the A matrix to be filled in. Following the formulation of [12], the conjugate gradient method is outlined in Algorithm 1.

OpenFOAM is used to set up the system of linear equations from evaluating (5). The A matrix and b vector are sent from OpenFOAM to the GPU based conjugate gradient library, where the system is solved. The cuSPARSE is used for the sparse Matrix-Vector multiplication of the conjugate gradient method, as it is highly optimized for all generations of NVIDIA GPUs. NVIDIAS cuBLAS library is also utilized for some of the vector-vector calculations. However some vector-vector operations were hand-written as the standard implementation of cuBLAS would require memory duplication which can be costly.

In order to choose a stopping point for the convergence of the linear solution a residual vector is calculated based on the current solution of, x^k, as

$$\vec{r}^k = b - Ax^k. \qquad (6)$$

Once the L_1 norm of $\vec{r}^k$,

$$\left\|\vec{r}^k\right\|_1 = \sum_{i=1}^{N} \left|r_i^k\right|, \qquad (7)$$

is 1000 times smaller than the L_1 norm of the residual vector before the first iteration, the solution is said to be converged to a relative tolerance of 10^{-3} and iterations are halted.

$$k = 0$$
$$r_k = b - Ax_k$$
$$p_k = r_k$$
$$k = 0$$
$$while \ \|r_k\|_1 < tolerance$$
$$Repeat$$
$$\alpha_k = \frac{r_k{}^T r_k}{p_k{}^T A p_k}$$
$$x_{k+1} = x_k + \alpha_k p_k$$
$$r_{k+1} = r_k - \alpha_k A p_k$$
$$\beta_k = \frac{r_{k+1}{}^T r_{k+1}}{r_k{}^T r_k}$$
$$p_{k+1} = r_{k+1} + \beta_k p_k$$
$$k = k + 1$$

Vector Operation

Sparse Matrix-Vector Operation

Algorithm 1: Conjugate gradient algorithm. Algorithm requires one sparse Matrix-Vector multiplication, as well as many vector operations.

Section 2.03 Cell Sorting Method

Cell sorting algorithms have been shown to improve the solution of CFD linear systems derived from unstructured grids with CPUs [13] [14]. The approach used herein is the Cuthill-McKee ordering which follows a breadth first ordering based on the connectivity of finite volumes to their neighbors [14]. Sorting the cells can improve memory locality and therefore cache performance. An example of a grid before and after sorting, partitioned and non-partitioned is shown in Figure 1.

Section 2.04 Performance Analysis

A set of 3-dimensional unstructured tetrahedral element grids were constructed ranging from 20,000 cells to 20,000,000 cells defined in the x, y, and z Cartesian coordinate ranges of [0,1], [0,1], and [0,1]. Performance is calculated as the amount of time it takes to converge the solution, as described in Section 3.B, divided by the number of cells in the grid. Performance metrics were collected for the following four cases: (1) 1 to 10 CPU cores, (2) use of the implemented GPU library, (3) using 10 CPU cores on a sorted grid, (4) using the GPU library with a sorted grid. All cases were run on all grids, timing was taken as the time spent in the conjugate gradient solver. The CPU based cases were partitioned and run with the base OpenFOAM conjugate gradient solver.

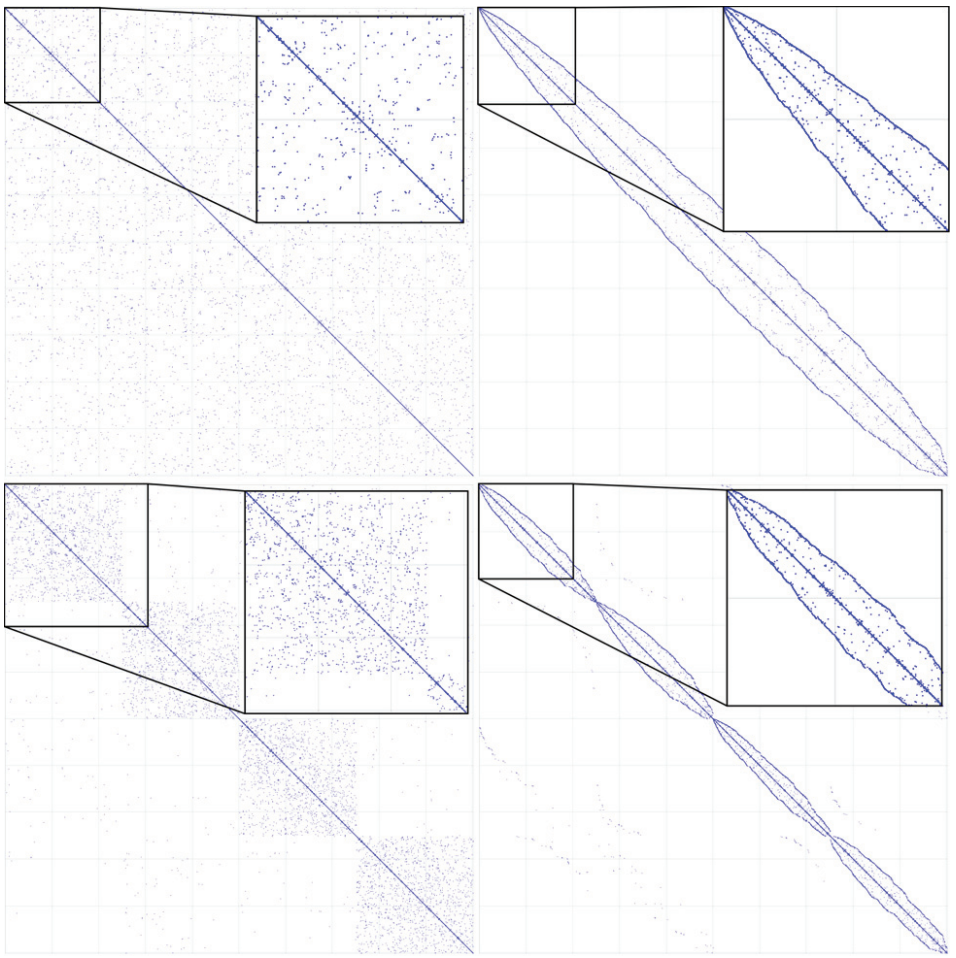

Figure 1: Top left, Unsorted sparse matrix. Top right, Cuthill-McKee sorted sparse matrix. Bottom left, sparse matrix resulting from a partitioned grid. Bottom right, sparse matrix resulting from a partitioned sorted grid.

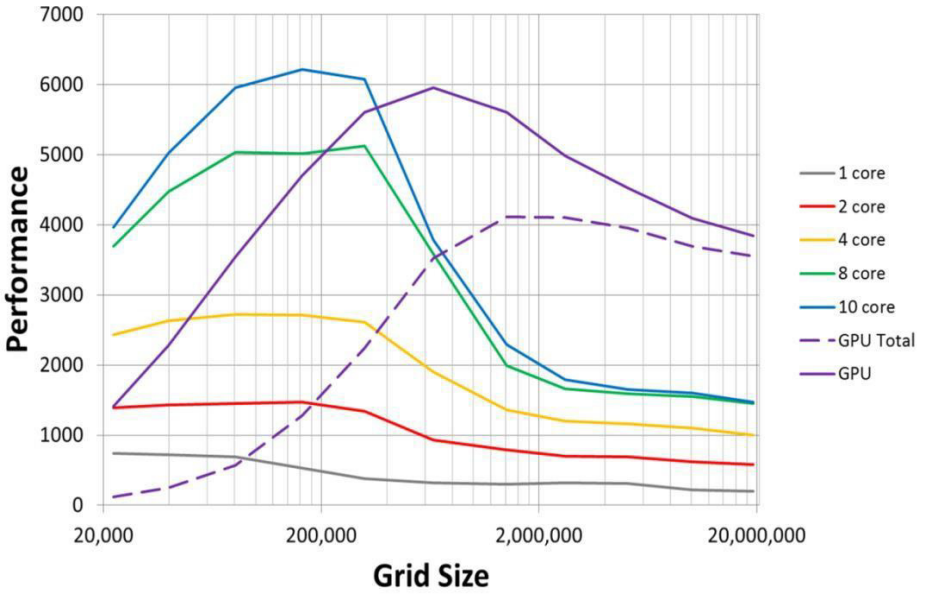

Figure 3: Performance of one through ten CPU cores as well as with the GPU, GPU Total includes the amount of time it takes to copy the A matrix and b vector to the GPU as well as copying the results back to the CPU.

3 Results

The timing study was done by observing how long the system took to solve the system of linear equations to a relative tolerance of 10-3. Figure 3 shows the performance of the algorithm versus size of grid used for cases without using the Cuthill-McKee sorting method. All grids used for this code are tetrahedral grids with small expansion factors. The performance is calculated by the number of cells processed multiplied by the number of iterations required divided by the total computation time. As expected performance of the OpenFOAM base solver is improved as more processers are used. However, for all CPU cases there is a sharp drop in performance for grids larger than 320,000 cells. This drop in performance is due to the computational grid becoming too large to completely store in CPU cache. Evidence of this is given by the decrease in performance occurring at a similar place for all CPU cases.

The GPU case on the unsorted grid shows similar behavior as the CPU where it has a peak performance and then decreases as grid size increases. This is still due to the grid becoming too large to fit into the GPU cache. The GPU hardware requires more operations, than the CPU, before the hardware is saturated, leading to a performance peak at higher grid sizes. The amount of time required to transfer all data to and from the GPU is included in Figure 3. However, this is not indicative of a fully implemented GPU accelerated CFD solver as the data is likely to reside on the GPU, and therefore not frequently transferred to and from the CPU.

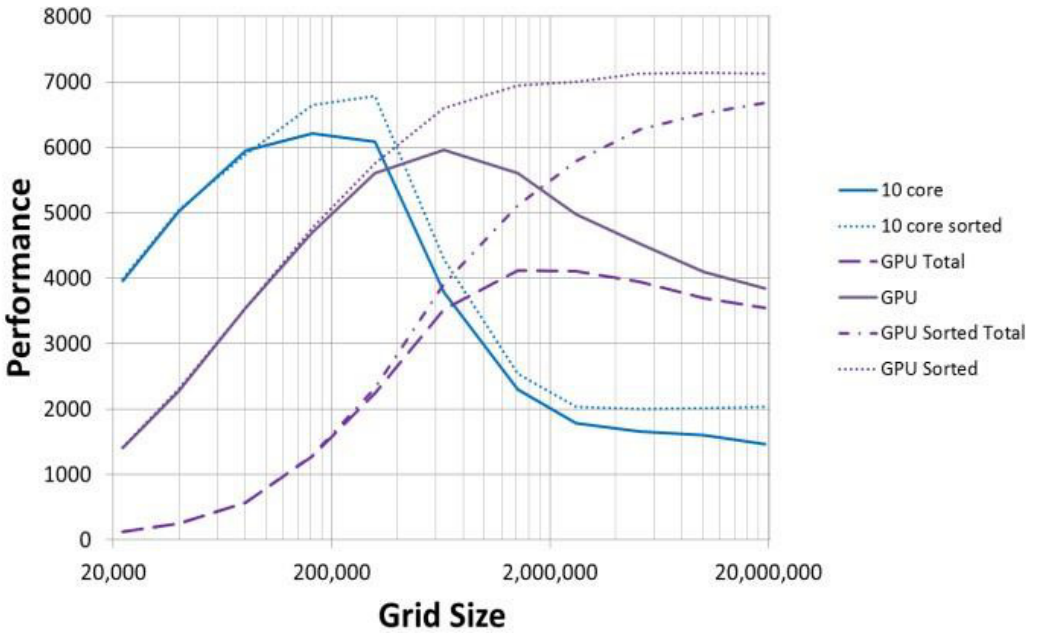

Figure 4: Performance of ten CPU and GPU on sorted and unsorted grids. GPU sorted total includes preparation of the A matrix and transfer time to and from the GPU.

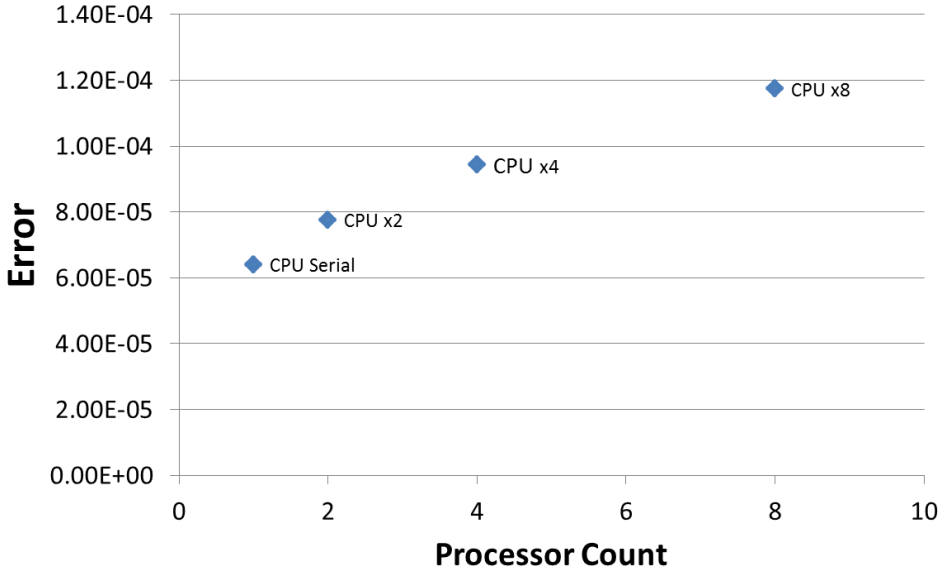

Figure 5: Error compared to the analytic solution using multiple CPU cores.

Figure 4 shows performance results on the same grids but sorted with the Cuthill-McKee ordering method. Performance of the 10 core CPU case on a sorted grid is only slightly better than on a non- sorted grid. However, the GPU is shown to greatly benefit from the Cuthill-McKee ordered grids especially when the grid sizes become larger. The GPU accelerated linear solver goes from being approximately 1.5 times faster than the CPU on larger grid sizes to 2.5 times faster. Even when factoring the additional GPU transfer time the performance of the GPU is still markedly improved compared to the CPU solver for large grid sizes. All of the GPUs cases still require a large grid size for good performance due to the required computational intensity to saturate the hardware and overcome overhead costs incurred for its use.

Further tipping the scales in favor of using GPUs is that as the grid is partitioned to use more CPU cores the error of the converged system increases. This is due to OpenFOAM partitioning the system of linear equations instead of solving the whole system. Communication is then performed by modifying the right hand side of the sub-systems to represent what is occurring on the borders of the partitions. This representation is similar to the original system of linear equations but not exact. Therefore it does not converge to the solution of the original system. Figure 5 shows the increasing error compared to the analytic solution when using more CPU cores.

4 Conclusion

For the solution of sparse systems of linear equations a K20 GPU was shown to provide a maximum of 2.5 times the performance of all 10 cores of an Intel Xeon E5-2670 v2. This shows great promise for the acceleration of Finite Volume Methods for Computational Fluid Dynamics (CFD). Their improved accuracy, by reduction in partitioning, could provide faster convergence methods for CFD by more accurately solving the system of linear equations for each iteration of the solution method. Future research will include a more computationally intensive iterative method like the preconditioned conjugate gradient method, as the higher arithmetic intensity can hide some of the high memory latency on GPUs, as well as full GPU implementation of a CFD solver and its scalability across a GPU based supercomputer.

5 Acknowledgement

This work was sponsored in part by the HPC Modernization Program’s HPC Internship Program (HIP).

References

[1] M. Smelyanskiy, D. Holmes, J. Chhugani, A. Larson and D. M. Carmean, "Mapping High- Fidelity Volume Rendering For Medical Imaging to CPU, GPU, and Many-Core Architectures," *Visualization and Computer Graphics, IEEE Transactions on,* pp. 1563-1570, 2009.

[2] L. A. Abbas-Turki, S. Vialle, B. Lapeyre and P. Mercier, "Pricing Derivatives on Graphics Processing Units Using Monte Carlo Simulation," *Concurrency and Computation: Practice and Experience,* vol. 26, no. 9, pp. 1679-1697, 2014.

[3] M. C. Schatz, C. Trapnell, A. L. Delcher and A. Varshney, "High-Throughput Sequence Alignment Using Graphics Processing Units," *BMC Bioinformatics,* vol. 8, no. 1, p. 474, 2007.

[4] N. CUDA, "NVIDIA CUDA C Programming Guide Version 7.0," *NVIDIA Corporation.*

[5] A. Corrigan, F. F. Camelli, R. Löhner and J. Wallin, "Running Unstructured Grid-Based CFD Solvers on Modern Graphics Hardware.," *International Journal for Numerical Methods in Fluids,* vol. 66, no. 2, pp. 221-229, 2011.

[6] F. Witherden, A. Farrington and P. Vincent, "PyFR: An Open Source Python Framework for High-Order CFD on Many-Core Platforms," *FEMTEC,* p. 141, 2013.

[7] D. Kim and H. Choi, "A Second-Order Time-Accurate Finite Volume Method for Unsteady Incompressible Flow on Hybrid Unstructured Grids," *Journal of Computational Physics,* vol. 162, no. 2, pp. 411-428, 2000.

[8] C. M. Rhie and W. L. Chow, "Numerical Study of the Turbulent Flow Past an Airfoil with Trailing Edge Separation," *AIAA Journal,* vol. 21, no. 11, pp. 1525-1532, 1983.

[9] L. S. Caretto, A. D. Gosman, S. V. Patankar and D. B. Spalding, "Two Calculation Procedures for Steady, Three-Dimensional Flows with Recirculation," in *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*, Springer Berlin Heidelberg,1973.

[10] M. Culpo, "Current Bottlenecks in the Scalability of OpenFOAM on Massively Parallel Clusters," PRACE White Paper to Appear on http://www.praceri.eu, 2011.

[11] J. Schmidt, M. Berzins, J. Thornock, T. Saad and J. Sutherland, "Large Scale Parallel Solution of Incompressible Flow Problems Using Uintah and Hypre," in *In Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, 2013.

[12] Y. Saad, Iterative methods for sparse linear systems, Siam, 2003.

[13] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th national conference*, 1969.

[14] E. Cuthill, in *Several Strategies for Reducing the Bandwidth of Matrices*, Springer US, 1972, pp.157-166.