Stochastic Simulation Efficiency of Parallel CFD Solver on Elastic Cloud Environment

Omran Malik Omer Awad, Awad Hag Ali Ahmed, Abdelmonem M. Ali Artoli

Abstract*—Computational fluid dynamics applications become crucial for scientist to understand various Natural phenomenon. These applications require high performance computing resources that most small academic institutions cannot afford. Elastic cloud clusters are best suited environment for those small academic institutions to gain high performance computing power and enable researchers to explore new trends in scientific computing with reasonable cost. This work aims to study the parallelism efficiency; in term of communication time and execution time for a highly optimized parallel lattice Boltzmann solver on elastic cloud clusters. On these elastic clusters we have found that the lattice Boltzmann implementation is fully adaptive, highly flexible and cost effective to use for solving complex large fluid mechanical systems.

Index Terms*—Computational Fluid Dynamics, Elastic Cloud Computing, Multi-core Programming, Lattice Boltzmann.

I. INTRODUCTION

Scientific computing requires rapidly increasing number of resources to deliver results for growing problem sizes in a reasonable time frame. In the last decade, while the largest academic institutions were able to afford expensive supercomputers to conduct high quality researches in the parallel processing field, other small institutions adopted cheaper resources such as commodity clusters and grids. Cloud computing offers cost-effective alternative solutions to such institutions in which resources are leased from cloud service providers only when needed rather than hosted in their premises [1]. In the other side, cloud computing promises a much more reliable platform than grids, as well as much more scalable platform than the largest of commodity clusters. Although there is several cloud computing providers, such as Amazon [2] and Azure [3], the potential of clouds remains unexplored. Our work aims to explore the performance of cloud environment running scientific applications.

A. Computational Fluid Dynamics

With the development in complexity of our real life and interactivity with the living environment it is now crucial to forecast future parameters affecting our lives to avoid sudden changes and catastrophes. It is known that flooding, tornados, and earth quakes, are the most killing machines of human.

Manuscript received on February, 2014.

Omran Malik Omer Awad, Department of Commuter Science, Alneelain University – Faculty of Computer Science and Information Technology, Khartoum, Sudan,

Prof. Awad Hag Ali Ahmed, Department of Computer Science, Alneelain University – Faculty of Computer Science and Information Technology

Prof. Abdelmonem Mohamed Ali Artoli, Computer Science Department, College of Computer & Information Sciences King Saud University, Riyadh, Kingdom of Saudi Arabia

These phenomena could be modeled as transport differential equations which may be solved numerically to obtain critical parameters affecting future changes and estimate future changes. In recent years flooding, tsunami, ice melting, economic catastrophes, and other sudden catastrophes are happening more frequent and less predictable. The lack of scientific approach to strategically solve these issues is witnessed. This might be due to lack of expertise and computational resources.

Fluid (gas, liquid, or mixtures) flows are governed by partial differential equations (PDEs) which represent conservation laws for the mass, momentum, and energy [4]. Solutions for these equations give us information on the velocity fields, stresses, and fluid structure interiors. A few analytic solutions on these equations exist for idealized cases. In most of realistic models, the structure is divided into blocks and domains and a numerical discretization for the PDE applied at each cell/grid point.

Computational fluid dynamics is derived from both fluid mechanics and computational techniques. It uses a set of numerical methods, computational resources, and visualization algorithms to enable us to solve the PDEs equations that governs the flow for fluids in a given domain. CFD is one of these branches that perfectly reflect the utilization for scientific computing. It is well known that CFD simulations require highly massive calculations that could not be performed with ordinary personal [5]. The computing power of today's supercomputers is mainly exploited in utilizing CFD solutions.

The Lattice Boltzmann Method (LBM) has been developed in the last two decades as promising numerical techniques for flows and other transport phenomena in fluids, to provide an alternative to traditional computational fluid dynamics (CFD) methods [6]. Unlike conventional numerical methods based on macroscopic continuum equations, the lattice Boltzmann method was developed from microscopic models and mesoscopic kinetic equations. In this paper LBM will be used for its known capabilities and suitability for parallel computing. Due to its simple implementation, straightforward parallelization and easy grid generation, the capability of the lattice Boltzmann method has been demonstrated in various complex applications including Newtonian blood flow simulations, non-Newtonian and suspension flows and complex geometry. As time-dependent flow simulations are known to be computationally expensive, a need for an efficient flow solver is crucial. Traditional Navier-Stokes solvers frequently use artificial compressibility and pressure projection methods to accelerate convergence [7].

B. LBM Governing Equations

The dynamics of the fluid is modeled by the transport of simple fictitious particles on the nodes of a Cartesian grid and is based on two steps;

streaming to the neighboring nodes and colliding with local node populations represented by the probability f_i of a particle moving with a velocity e_i per unit time-step Δt as shown in Eq. (3), Populations are relaxed towards their equilibrium states during a collision process [8] [9]. The most common models are the D2Q9 model and D3Q19 model. We focus in D2Q9 model here in the simulations. The macroscopic variables are defined as functions of the particle distribution functions in the following equations:

$$\rho = \sum_{i=0}^{\beta-1} f_i \tag{1}$$

and

$$u = \frac{1}{\rho}\sum_{i=0}^{\beta-1} f_i e_i \tag{2}$$

Where Eq. (1) and (2), represent macroscopic fluid density and macroscopic velocity respectively. The particle distribution functions at each lattice point are updated using the following equation:

$$\underbrace{f_i(x + e_i\Delta t, t + \Delta t) = f_i(x,t)}_{Streami\ ng} \underbrace{- \frac{[f_i(x,t) - f_i^{eq}(x,t)]}{\tau}}_{Collision} \tag{3}$$

Where $i \in [0, \beta - 1]$ is an index for discrete velocities and τ is a relaxation parameter, which is related to the fluid viscosity as detailed in the following sections.

The Eq. (3) apply for lattice points located inside the fluid domain, but not for those at boundaries, where boundary conditions should be applied to predict the absent particle distribution functions, so the two steps streaming and collisions should be treated separately in actual implementations. The streaming step, where the particle distribution functions are translated to the neighboring sites according to the respective discrete velocity direction, see [10].

The equilibrium distribution functions can be obtained from the local Maxwell-Boltzmann (see the following equation):

$$f_i^{eq}(x) = \omega_i \rho(x)\left[1 + 3\frac{e_i}{c^2} + \frac{9}{2}\frac{(e_i \cdot u)^2}{c^4} - \frac{3}{2}\frac{u^2}{c^2}\right] \tag{4}$$

Where ω_i are the weights for the particles and c is the propagation speed on the lattice, in most cases considered $c = 1$.

For the D2Q9 model the velocity weights are:

$$\begin{cases} \omega_{i=0} & = \frac{4}{9} \\ \omega_{i=\{1..4\}} & = \frac{1}{9} \\ \omega_{i=\{5..8\}} & = \frac{1}{36} \end{cases} \tag{5}$$

It is well known that lattice Boltzmann method by using the single relaxation time approximation and particular Maxwell-type distribution will recover the Navier-Stokes equations [11].

The fluid viscosity is calculated according to the following equation:

$$v = \left[\frac{(2\tau - 1)}{6}\right] e^2 \delta t \tag{6}$$

Where e the lattice velocity.

II. ELASTIC COMPUTING PLATFORM

Elastic computing platforms (EC2) are based on Xen virtualization technology [12]. This allows one physical computer to be shared by several virtual instances, each of which hosts different operating systems. Each virtual OS has its own root, and lives in its own separate universe.

EC2 provides users with virtual instances based on Linux operating systems. A range of 32-bit and 64-bit kernels supporting the common Linux varieties such as Ubuntu and Fedora Core are available. Amazon has made available a number of Amazon Machine Images (AMIs) which can be hosted on their computers. [2].

Our experiments have been conducted on five baseline platform architectures that are offered by Amazon scientific cloud computing services. These platforms are: four virtual cores machine (m1.xlarge), eight virtual cores machine (m2.4xlarge) and (c1.xlarge), eight (2 x Intel Xeon X5570, quad-core with hyperthread) machine (cc1.4xlarge), sixteen (2 x Intel Xeon E5-2670, eight-core with hyperthread) machine (cc2.8xlarge). Table 1 summaries the architectures used in our study.

Table 1 EC2 machine instances used for experiments on Amazon Cloud

EC2 Instances	***m1***	***m2***	***c1***	***cc1***	***cc2***
Cluster size (**n** nodes)	16	8	8	8	4
Virtual CPUs per machine	4	8	8	8	16
Computing Units per core[1]	2	3.25	2.5	2 x Intel Xeon X5570, with hyperthread	2 x Intel Xeon E5-2670, with hyperthread
Memory size in GB	15	68.4	7	22.5	60.5
Network Interface bandwidth	High	High	High	10 Gbps	10 Gbps
Support for H.W virtualization	No	No	No	No	Yes
Grouped in one place	No	No	Yes	Yes	Yes
System Architecture	64-bit	64-bit	64-bit	64-bit	64-bit
Operating System	Ubuntu 12			Centos 5.6	

A. Elastic Cloud setup

We now describe the experimental setup in which we use the performance evaluation method presented in section III.

For our experiments we build homogeneous environments with 1 to 128 cores based on the five EC2 instance types., we used the images based on Ubuntu 12 OS with Linux 2.6 kernel for the first three instances and Centos 5.6 with the same kernel for cc1 and cc2 platforms. We used the ***starcluster*** deployment tool programmed by MIT to facilitate the cluster configuration. Using ***starcluster*** we configured all machines with MPI-2 libraries based on the mpich2 [31] implementation. Also the MIT deployment tools equipped

[1] One compute unit (CU) provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor according to Amazon EC2 documentation

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

with scripts to install Sun Grid Engine (SGE) as cluster scheduler.

III. SIMULATION METHODOLOGY

In the following sections we will discuss the cluster performance measurements. In our measurements we used different approach that implements real CFD problem rather than the industry standard benchmarks. Our benchmark consists of two-dimensional tube flow tested on the five different cloud platforms. the measurement metrics being used is explained in section B.

Our code adopted the algorithm shown in Table 2. It is executed under Linux OS.

Table 2 Lattice Boltzmann simulation algorithm

	Lattice Boltzmann Algorithm
1	Allocate memory
2	Initialize parameters
3	Partition the domain
4	***Do***
5	Compute mass and momentum
6	MPI_Reduce computed mass & momentum
7	Set upper and lower velocity boundary conditions
8	Set inflow and outflow pressure boundary conditions
9	Collide lattice sites
10	Stream sties
11	Set Bounce back on links
12	Send boundary info to west neighbor
13	Send boundary info to east neighbor
14	Update periodic boundary
15	***End do***
16	Compute output

We choose to divide the problem space geometry into *n* subdomains among x-axis. To ensure load balancing in simple way, we set *n=p*, where *p* is the number of processors involved in the solution. The subdomains then distributed randomly to the processors in a way that each processor will have one subdomain to process it. This technique has enhanced communications and lead to better load balancing, the disadvantage of this technique appear on small number of processors which cause increase time in local calculations. This drawback however is negligible when the code is executed in modern multi-core processors. The domain partitioning is illustrated in Fig. 1

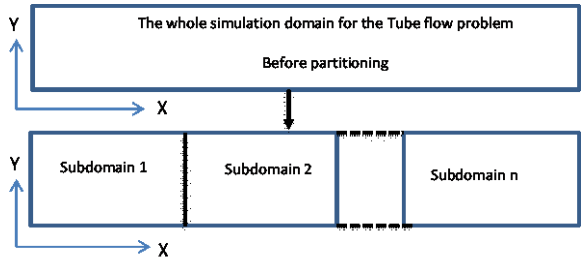

Fig. 1 Tube flow partitioning among x-axis

A. Simulation Parameters

In this section we will describe the test benchmark problem. The tube flow problem is the candidate; we consider the tube geometry to be of 100 lattice units' width and 1000 lattice units' length as shown in the following figure.

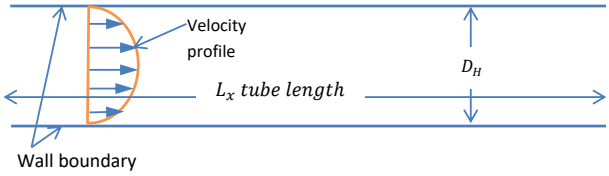

Fig. 2 Tube flow simulation geometry

The flow in the tube is considered to be at *Re=100*. Reynolds number (Re) is used to perform dimensional analysis of fluid dynamics problems, and as such can be used to determine dynamic similarity between different experimental cases [8] [11] [12].

To formulate the tube flow parameters we consider the Reynolds number equations as follows [7]:

$$R_e = \frac{vL}{\gamma} = \frac{vD_H}{\gamma} \quad (7)$$

Where v is the velocity, L is the characteristic length of the tube, γ is the kinematic viscosity, D_H is the characteristic tube diameter length

$$\gamma = \frac{\mu}{\rho} \quad (8)$$

Where μ is the dynamic viscosity of the fluid, ρ is the density of the fluid.

The velocity can be obtained from the following equation:

$$v = \left[\frac{(2\tau-1)}{6}\right] e^2 \delta t \quad (9)$$

The lattice distance scales selected to be of $\Delta x = 0.1$ and $\Delta y = 0.1$, the initial velocity in x-direction is selected to be 0.1 lattice unit, by using the Eq. (9) we get:

$$\tau = \frac{\left[\frac{6v}{e^2}+1\right]}{2} \quad (10)$$

By substituting *v=0.001* and *e=0.1* in Eq. (10) we get *τ=0.8* for relaxation time. Table 3 shows the complete parameters for the simulation initial setup.

Table 3 Tube flow simulation parameters

Parameter (in Lattice Units)	***Value***
Initial fluid velocity in x-direction (v_x)	0.1
Initial fluid velocity in y-direction (v_y)	0.0
Relaxation Time (τ)	0.8
Fluid density[2] (ρ)	1.0
Number of lattice cells in x-direction (L_x)	1000
Number of lattice cells in y-direction (L_y)	100
Pertubation[3] (*Statistical fluctuations*)	0.001

A. Execution Time

For all test beds, we first evaluate the solver execution time which is measured in seconds. The way we measured this metric is hardcoded in the solver itself by utilizing an intrinsic function that is bundled in mpich-2 library. This function is MPI_Wtime() which records the time at the

[2] Density of water in room temperature $\rho = 1000\ Kg/m^3$

[3] Used to drive the flow in the tube in case of body force absence by making pressure gradient in the tube

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

beginning of the simulation code and at the end of the code. Then we subtract the start time from the end time to get the exact execution time as illustrated below.

```
Sim_StartTime = MPI_Wtime();
/* Solver code */
/* Solver Code */
.
.
.
/* Solver Code */
Sim_EndTime = MPI_Wtime();
Execution_Time = Sim_EndTime-Sim_StartTime
```

This execution time computes the total execution including communication time and idle.

B. Message Communication Time

Communication time metric is measures similarly to the execution time except that we record the communication time inside the simulation loop and accumulate the time in each loop step. We record the starting time of communication just before the message sending code line and record the ending time write after the message sending code line. After the loop is completed, the final value is recoded in the log file. The following code illustrates the process of communication time calculation.

```
Comm_StartTime = MPI_Wtime();
send_west(param.NX, param.NY, top
send_east(param.NX, param.NY, top
Comm_EndTime = MPI_Wtime();
total_comm_time += (Comm_EndTime-Comm_StartTime);
```

Typical measurement for communication time increases as the number of processing cores increase. And the communication overhead becomes more influential.

IV. RESULTS

A. *Results for Parallel Execution Time*

To compare the least performing platform m1.xlarge cluster execution time with it is ancestor (m2.4xlarge) Amazon cloud cluster, we read the graph in Fig. 3. It shows the execution time in seconds as a function of the number of processing cores (in a logarithmic scale). It is to be noted that for ideal behavior, execution time is inversely proportional to the number of processors. From the aforementioned graph we notice that the execution time is initially reduced one order of magnitude (~O(n)) as the number of processing cores is doubled. As the number of processor cores is increased to more than 16 cores, the execution time fluctuates due to the fact that inter-process communication between processing elements becomes more dominant. The two clusters perform similarly as the number of processors is more than 32.

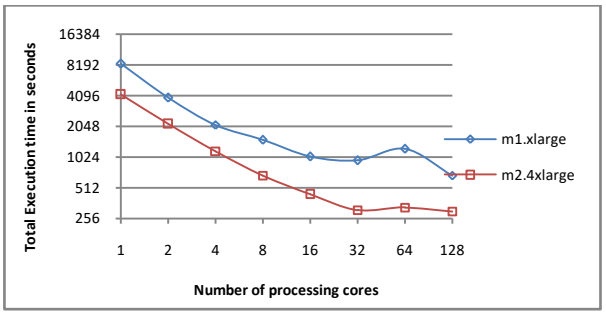

Fig. 3 Execution time for m1.xlarge and m2.4xlarge Amazon clusters

In Fig. 4 an execution time comparison between m1.xlarge and c1.xlarge clusters are presented. An interesting phenomenon is observed in this figure. The cluster c1.xlarge has less memory than m1.xlarge cluster but it is an 8-core. Therefore the execution time is better than the m1.xlarge but has greater communication overhead. Therefore, as the number of processors is increased, the communication time dominates. This is a property of the used solver.

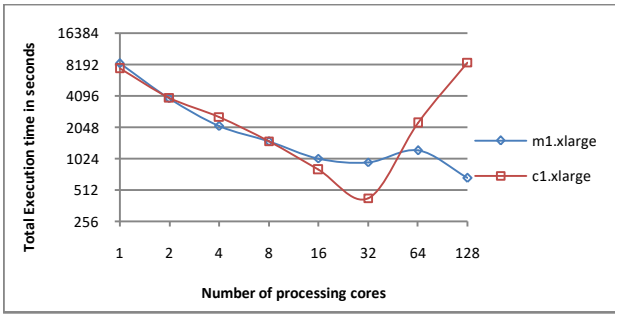

Fig. 4 Execution time for m1.xlarge and c1.xlarge Amazon clusters

In Fig. 5 we present an execution time comparison between m1.xlarge and cc1.4xlarge Amazon clusters. We noticed that cc1.4xlarge cluster execution time decreases consistently as number of processing cores increase. cc1.4xlarge performed better because the communication overhead is eliminated using Amazon placement group. Also an interesting phenomenon has been noticed. The execution time for both clusters goes similarly up to 16 processes. This is because the communication overhead is barely influencing the performance. When the number of processing cores increased (more than 16 processing cores) we noticed the influence of the communication overhead.

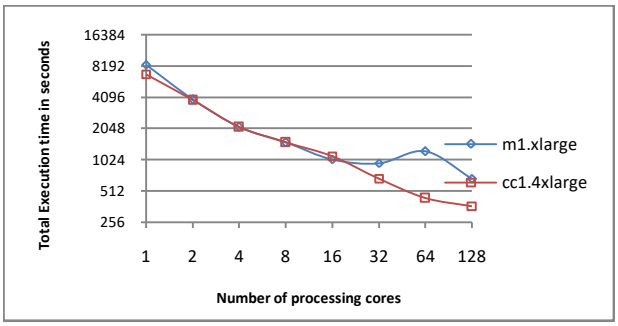

Fig. 5 Execution time for m1.xlarge and cc1.4xlarge Amazon clusters

In the following graph, see Fig. 6 an execution time comparison between m1.xlarge and cc2.8xlarge the largest Amazon cluster

is presented. In this graph we noticed that the highest performing clusters perform better when solvers are running with greater number of processing cores.

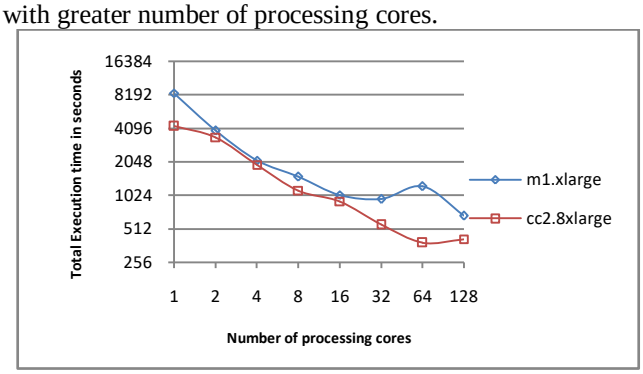

Fig. 6 Execution time for m1.xlarge and cc2.8xlarge Amazon clusters

B. *Results for Message Communication Time*

In Fig. 7 we present the communication time elapsed during the simulation of our code on m1.xlarge and m2.4xlarge Amazon clusters. From this figure we read that the communication time for single processing core is Zero (no message passing occurs). Starting from 2 processing cores the execution time is increased linearly as the number of processing cores is doubled. We noticed a fluctuation in the performance of m1.xlarge due to fact that m1.xlarge machines are placed randomly by Amazon in different regions which affects the performance of network bandwidth. However, the two clusters perform similarly as the number of processors is more than 32.

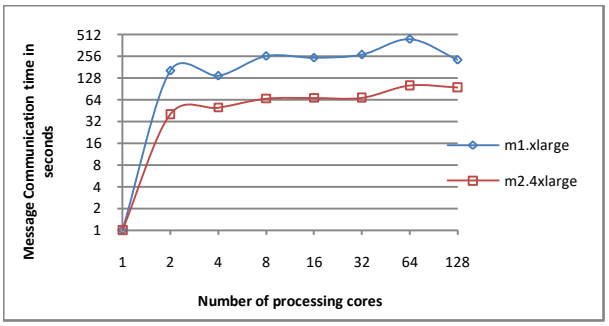

Fig. 7 communication time for m1.xlarge and m2.4xlarge Amazon clusters

In Fig. 8 we present a communication time comparison between m1.xlarge and c1.xlarge clusters. We notice that m1.xlarge outperform c1.xlarge cluster when the number of processing cores is greater than 32. This is because the communication overhead is dominating.

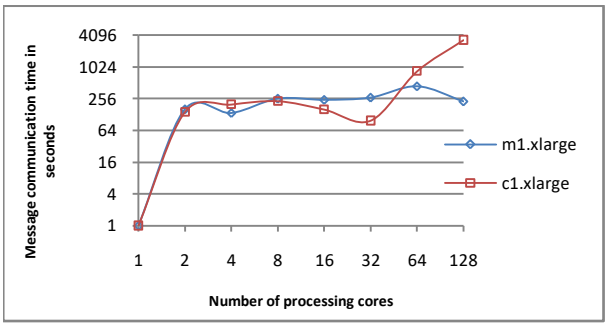

Fig. 8 communication time for m1.xlareg and c1.xlarge Amazon clusters

In Fig. 9 a message communication time comparison between m1.xlarge and cc1.4xlarge is presented. Both clusters are performing similarly with less communication time in cc1.4xlarge.

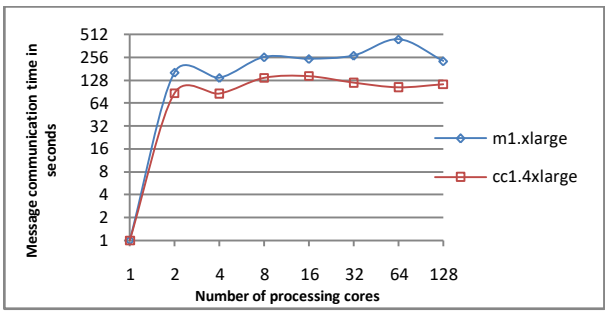

Fig. 9 Messages communication time comparison between m1.xlarge and cc1.4xlarge Amazon clusters

Finally, in Fig. 10 we present a message communication time between m1.xlarge and cc2.8xlarge clusters. We notice that cc2.8xlarge is the best performing cluster among all the test beds.

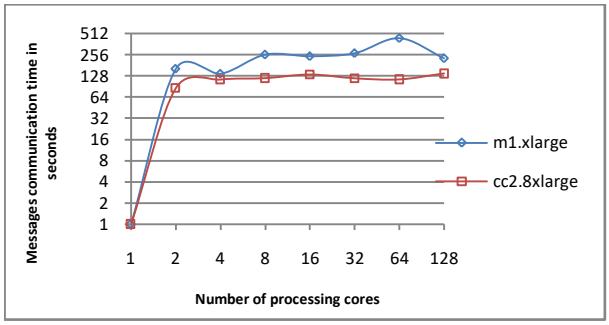

Fig. 10 Messages communication time comparison between m1.xlarge and cc2.8xlarge Amazon clusters

V. CONCLUSION

The comparison results were given in figures 1 – 10 which compared execution time, communication time, the five test beds. From these figures we can conclude that:

- Because infrastructure is rented, not purchased, the cost is controlled, and the capital investment can be zero.
- General purpose platforms in Amazon could be used as high performance cluster for simulating CFD phenomena with reduced cost in comparison with high end dedicated platforms for scientific cloud computing in Amazon such as cc1.4xlarge and cc2.8xlarge platforms.

FUTURE WORK

We are hoping to use the findings of this work to implement a general purpose highly optimized, self-tuning, less costing and robust CFD solver to be used for forecasting and engineering problems.

ACKNOWLEDGE

Work conducted by the researchers is supported (in part) by King Abdulaziz University – Faculty of Computing and Information Technology – Khulais Branch.

REFERENCES

[1] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer and D. Epema, "A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing," in *Cloud Compting* , Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 2010.

[2] "AWS Amazon Elastic Compute Cloud (EC2) – Scalable Cloud Servers:," 2013. [Online]. Available: http://aws.amazon.com/ec2/.

[3] "Cloud Services - Windows Azure," Microsoft Co. Ltd., 2013. [Online]. Available: http://www.windowsazure.com/en-us/services/cloud-services/.

[4] M. J. H. Ferziger, "Computational Methods for Fluid Dynamics," 1996.

[5] P. Wesseling, "Principles of Computational Fluid Dynamics," 2001.

[6] Y. Yan, Flow and particle transport by the Lattice Boltzmann Method, New York: ProQuest, 2008.

[7] X. He and L.-S. Luo, "Lattice Boltzmann model for the incompressible Navier-Stokes equation," vol. Journal of Statistical Physics, no. 88, 1997.

[8] A. M. Artoli, "Mesoscopic Computational Haemodynamics," *PHD Thesis*, 2003.

[9] A. M. Artoli, D. Kandhai, H. J. Hoefsloots, A. G. Hoekstra and P. M. Sloot, "Lattice BGK simulations of flow in a symmetric bifurcation," *Future Generation Computer Systems*, vol. 20, no. 6, pp. 909-916, 2004.

[10] M. Geveler, D. Ribbrock, D. Goddeke and S. Turek, "Lattice-Boltzmann Simulation of the Shallow-Water Equations with Fluid-Structure Interaction on Multi- and Manycore Processors," *Facing the Multicore-Challenge*, vol. 6310, 2010.

[11] H. Chen, S. Chen and W. H. Matthaeus, "Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method," *The American Physical Society*, vol. 45, no. 8, 1992.

[12] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the Nineteenth ACM Symposium on Operating systems principles*, New York, USA, 2003.

[13] S. Succi, The Lattice Boltzmann Equation for Fluid Dynamics and Beyond, USA : Oxford University Press, 2002.

[14] A. M. Artoli, D. Kandhai, H. C. Hoefsloot and A. G. Hoekstra, "Lattice Boltzmann, a Robust and Accurate Solver for Interactive Computational Hemodynamics," *Computational Science*, vol. 2657, pp. 1034-1043, 2003.

AUTHOR PROFILE

Omran Malik Omer Awad Ph.D. student at Alneelain University, Lecturer at King Abdulaziz University – Khulais branch, Faculty of Computers and Information Technology, Department of Information Technology. E-mail: omranawad@hotmail.com

Prof. Awad Hag Ali Ahmed Ph.D., Department of Computing Science, University of New Castle Upone-Tyne, England 1981. Vice-chancellor Al-Neelain University 1997- 2005. External examiner and accreditor of computing science and IT programs in many local and foreign Universities. Publications (70+)

Prof. Abdel Monim Artoli Professor of computational science, Computer Science Department, Vice-deanship for Development and Quality, Head of Alumni Unit, Department of Computer Science, College of Computer & Information Sciences, King Saud University