SEMARAK ILMU PUBLISHING

Journal of Advanced Research in Applied Sciences and Engineering Technology

Journal homepage:
https://semarakilmu.com.my/journals/index.php/applied_sciences_eng_tech/index
ISSN: 2462-1943

Evaluation of CFD Computing Performance on Multi-Core Processors for Flow Simulations

Iman Fitri Ismail[1], Akmal Nizam Mohammed[1,*], Bambang Basuno[2], Siti Aisyah Alimuddin[1], Mustafa Alas[3]

[1] Flow Analysis, Simulation, and Turbulence Research Group, Universiti Tun Hussein Onn Malaysia, 86400, Parit Raja, Batu Pahat, Johor, Malaysia
[2] Aerodynamics and Propulsion Research, Universiti Tun Hussein Onn Malaysia, 86400, Parit Raja, Batu Pahat, Johor, Malaysia
[3] Civil Engineering Department, Near East University, Near East Boulevard, Nicosia, Turkish Republic of Northern Cyprus, Mersin-10, Turkey

ABSTRACT

Previous parallel computing implementations for Computational Fluid Dynamics (CFD) focused extensively on Complex Instruction Set Computer (CISC). Parallel programming was incorporated into the previous generation of the Raspberry Pi Reduced Instruction Set Computer (RISC). However, it yielded poor computing performance due to the processing power limits of the time. This research focuses on utilising two Raspberry Pi 3 B+ with increased processing capability compared to its previous generation to tackle fluid flow problems using numerical analysis and CFD. Parallel computing elements such as Secure Shell (SSH) and the Message Passing Interface (MPI) protocol were implemented for Advanced RISC Machine (ARM) processors. The parallel network was then validated by a processor call attempt and core execution test. Parallelisation of the processors enables the study of fluid flow and computational fluid dynamics (CFD) problems, such as validation of the NACA 0012 airfoil and an additional case of the Laplace equation for computing the temperature distribution via the parallel system. The experimental NACA 0012 data was validated using the parallel system, which can simulate the airfoil's physics. Each core was enabled and tested to determine the system's performance in parallelising the execution of various programming algorithms such as pi calculation. A comparison of the execution time for the NACA 0012 validation case yielded a parallelisation efficiency above 50%. The case studies confirmed the Raspberry Pi 3 B+'s successful parallelisation independent of external software and machines, making it a self-sustaining compact demonstration cluster of parallel computers for CFD.

Keywords:
Computational Fluid Dynamics;
Raspberry Pi; parallel computing; RISC;
MPI; SSH;NACA

Received: 17 July 2022 **Revised: 28 August 2022** **Accepted: 1 September 2022** **Published: 19 Sept. 2022**

1. Introduction

Computational Fluid Dynamics (CFD) is a contemporary tool used in conjunction with experimental work to compute and predict the mechanics of flow media such as liquids and gases. CFD results are analogous to those obtained via physical experiments as the numerical solutions usually agree with the empirical data [1]. However, analysing these complex simulations requires

* *Corresponding author.*
E-mail address: akmaln@uthm.edu.my

https://doi.org/10.37934/araset.28.1.6780

substantial computing power and time to acquire good accuracy [2]. The development of computational technology allows for accelerated fluid flow analysis since it can be accessed via terminals and is easily transportable compared to wind tunnels, with many constraints, including obstruction, ground modelling, and other boundary interference effects [3].

Nowadays, CFD analysis can be performed on a network of parallel computing systems. Parallel computing is the latest technology to model complicated fluid dynamic problems efficiently. Moreover, it employs several processors to fully exploit these parallel networks' computational capabilities and vast memory capacity to address the needs of an increasing large-scale CFD application in scientific research [4].

In recent years, energy-efficient and data-intensive computing is becoming an area of interest in the industry and educational institutions for research and development purposes. However, the new approach and techniques of CFD require state-of-the-art computers with higher processing capabilities that usually impose higher installation and operation costs. As a result, there is a growing interest in utilising low-power CPUs fitted with energy-efficient chips as the architectures of these CPUs have the potential to be an innovative and cost-effective alternative for future high-performance computing systems [5].

The development of parallel computing for CFD is always constricted to conventional computers fitted with complex instruction set computing (CISC) processors. However, to accommodate the current demand increase in software development for ARM processors, it is essential to understand its architecture and implementation to yield results from its optimum [6].

The study aims to construct a network of parallel computers via Message Passing Interface (MPI) and Secure Shell (SSH) protocol to study the performance of computing time in distributed parallel system of Raspberry Pi 3 B+. Then, the parallelisation of onboard ARM processors is verified by test cases and fluid flow case studies. To evaluate these ARM processors' ability to handle CFD test scenarios, the airfoil coefficient from NACA 0012 experimental data will be validated on the same parallel system. While numerous studies have been conducted to evaluate the performance of conventional processors in CFD applications, few to none have made use of the ARM processors installed in Raspberry Pi single-board computers (SBCs). Thus, this work is vital to implementing fluid dynamics computation in a parallel computing system based on the RISC processor architecture. Furthermore, this technology will aid future academics and engineers who wish to analyse fluid flow over an airfoil on a small-scale parallel computer system. The possible benefits include the ability to rapidly execute a program to analyse fluid flow over an airfoil using a low-cost setup and open-source code compared to currently available commercial software.

1.1 Parallel Computers with RISC Processors

Motivated by the fact that earlier single-board computers (SBCs), such as the Raspberry Pi 1, were limited to a single-core processor, parallel computing algorithms became more complicated to implement. Thus, researchers focused on connecting SBCs as microclusters, aided by the Message Passing Interface (MPI), a standard library for message passing across parallel computers [7]. The first study of such implementation was IridisPi, using Raspberry Pi Model B [8]. Epiphany is a high-performance manycore architecture that is used in an embedded system, and it was used by Adapteva in its first product that features a 16-core 32 GFLOPS chip with a size of 65nm in 2011 [9]. Another example of small scales parallel computing is for computational fluid dynamics (CFD) application, which consists of clusters of peak performance over 3 GFLOPS which is made with three Raspberry Pi 2 as nodes and costs £120 without the networking equipment [10].

Theoretically, applying the concept of parallel computing is intended to minimise the time taken for the execution of a task. Contrary to a conventional computer, which is costly and bulky, parallel computing in the form of clusters can be constructed with a low-cost and small system of components such as Iridis-pi, which was assembled as a demonstration cluster using 64 Raspberry Pi nodes interconnected via ethernet links. A single node exhibited around 65000 kFLOPS for a problem size of n=100, and 1.14 GFLOPS for the whole cluster of 64 nodes with a problem size of n=10240 [10]. The performance of Iridis-pi is much poorer than Guthrie's CFD simulation cluster of three Raspberry Pi 2 since the cluster consists of 64 nodes of the first-generation Raspberry Pi with a 500MHz single-core processor.

The processors of Raspberry Pi 3 Model B+ are classified as RISC, this architecture of processors allows them to have fewer cycles per instruction (CPI) compared to a CISC. Comparatively, Raspberry Pi 3 Model B+ have higher CPI compared to its previous generation. This is because RISC processors work on simple instructions executed in one clock cycle, whereas CISC aims to complete a task in fewer lines of assembly as possible [11].

1.2 Parallel RISC Machines for CFD

The RISC processors suffer low floating-point operations per second (FLOPS) of only 15% to 20%. The fundamental causes of these occurrences are the complexity of instruction-level coding for devices with asynchronous internal design and, more importantly, the unacceptably poor throughput of primary memory subsystems [12].

Boeing's high-performance computing benchmark suite (BHPCBS), consisting of NASA's OVERFLOW codes and computational electromagnetics (CEM) codes, has been benchmarked on RISC parallel systems and CISC clusters. The connection between the processors was facilitated using MPI. However, CFD tests conducted on the RISC processors posed several limitations, such as the low amount of memory. In addition, some CFD studies revealed several constraints, including insufficient memory and processor cache, a lack of virtual memory, and poor memory per-node performance [13].

Hence, the performance and development of finite element method-based CFD software were studied using IBM RISC workstations. The study concluded that solving non-linear equations on RISC machines required substantial communication between processors, reducing the expected parallelisation speed-up [14].

Although the introduction of parallel computing via RISC processors has garnered drawbacks over the recent decade, the continuously improved architecture of the processors introduced much more powerful capabilities. However, integrating RISC processors into smartphones and Internet of Things (IoT) systems necessitates trade-offs in die size, power consumption, cost, and performance, all of which are the benefits of RISC processors. As a result, demands for x86-based systems declined by 10% per year peaking in 2011, while RISC chips now made up to 99% of 32-bit and 64-bit processors today [15].

2. Raspberry Pi 3B+ Parallelisation Setup

This section provides an overview of the Raspberry Pi cluster, including its hardware and software components. The parallelisation and CFD test cases are fully described, including their governing equations and how they assess processor capability.

2.1 Hardware

The devices connected as nodes in the Raspberry Pi parallel system are of the Raspberry Pi 3 Model B+, featuring 1.4Ghz Quad-core Cortex-153 processors that allow multiple task processing. The device is composed of a system-on-chip (SoC) board, which is described as a tiny chip that contains all of the components necessary for the operation of a computer [16]. The pi is powered by a 4.4A 22W power supply and connected to the ethernet switch via RJ45 cable for parallelisation. Local storage is provided via a secure digital (SD) card slot, and the board includes low-level interfaces such as Inter-Integrated Circuit (I2C), Serial Peripheral Interface (SPI), Universally Asynchronous Receiver / Transmitter (UART), and General-Purpose Input Output (GPIO) that enable the connection and powering of active cooling systems [17].

The central hub was constructed using the ethernet switch. As a result, the parallel system was connected to a D-Link (DES-1008A). The D-Link (DES-1008A) features eight fast ethernet LAN ports that enable high-speed data transfer between nodes. The parallelisation processes follow the procedures used in prior literature [18]–[20]. Its plug-and-play feature also eases the setting of the networking configuration for the parallel system. The simple parallel system described here is visualised in Fig. 1.

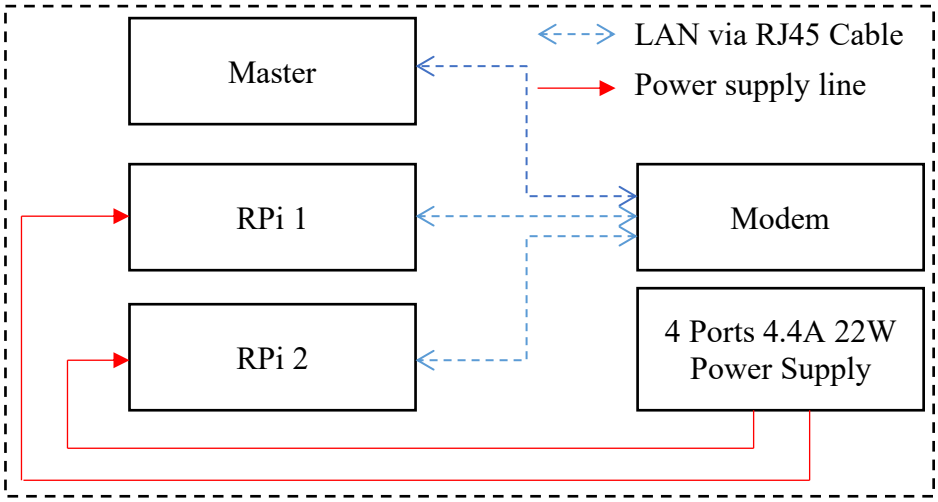

Fig. 1. Raspberry Pi 3 Model B+ Parallel System

The equation that exhibits efficiency for two nodes of Raspberry Pi 3 Model B+ against a single node is given by the following equation.

$$Efficiency, \% = \left| \frac{t_{8\ cores} - t_{4\ cores}}{t_{4\ cores}} \right| \times 100 \quad (1)$$

Where, t_{8cores} is execution time for two nodes of Raspberry Pi and t_{4cores} is execution time for a single node of the board.

2.2 Software

Raspbian OS Stretch Lite was chosen for this experiment because it is a lightweight operating system based on the Debian distribution. The distribution includes numerous tips and a handbook for troubleshooting if any issue arises throughout the experiment. Raspbian is installed on the devices

before the inclusion of parallel components, as the devices require an operating system to function. The Raspbian Stretch Lite OS can be downloaded from Raspberry Pi's official website.

The hostname is modified to make the pi easily identifiable in the parallel system. The master node is referred to as 'RPAR-1,' which stands for Raspberry Pi in Parallel 1. The naming of the subsequent new device will begin with number two and continue upward. Secure Shell (SSH) was set on the 'Interfaces' page to enable the master node to remotely access and log into other devices connected to the parallel system.

After enabling SSH, as illustrated in Table 1, the packages list from the repositories must be updated to reflect the most recent versions of the packages and dependencies. This assures that protocols installed will use the most recent version, maximising the system's capabilities. After installing the parallel computing parts in both devices, they are rebooted to apply the system's changes. Confirming the parallel system's successful configuration of SSH can be done on a distant computer running a Linux distribution and linked to the parallel system over the same ethernet switch. The MPI necessities were installed on the devices to allow the parallelisation algorithm to be automatically assigned once the module is called inside the programming codes. Configuration of static IP address allows for one-time changes to the host files, making it easier for connection and SSH between devices to be conducted with ease.

Table 1
Installation of parallel system components via terminal

No.	Terminal Inputs	Explanation
1	pi@RPAR-1:~$sudo apt-get update	Packages list and dependencies update.
2	pi@RPAR-1:~$sudo apt-get upgrade	The upgrade process of dependencies and packages.
2	pi@RPAR-1:~$sudo apt-get install manpages-dev	GNU/Linux devices, file formats, and syntaxes.
3	pi@RPAR-1:~$sudo apt-get install gfortran	Installation of GNU Fortran compiler.
4	pi@RPAR-1:~$sudo apt-get install nfs-common	Network File System (NFS) for client/server application.
5	pi@RPAR-1:~$sudo apt-get install nfs-kernel-server	NFS server installation for directories sharing.
6	pi@RPAR-1:~$sudo apt-get install vim	'Vim' editor installation to edit programming codes.
7	pi@RPAR-1:~$sudo apt-get install openmpi-bin	OpenMPI dependencies and MPI components.
8	pi@RPAR-1:~$sudo apt-get install libopenmpi-dev	OpenMPI libraries for developer's installation.
9	pi@RPAR-1:~$sudo apt-get install openmpi-doc	Description of Message Passing Interface standards.
10	pi@RPAR-1:~$sudo apt-get install keychain	SSH password manager via Debian-based cloud server.
11	pi@RPAR-1:~$sudo apt-get install nmap	Installation of auditing security utility and discovery of network.

2.3 Pi Calculation via Monte Carlo Simulation

The use of Secure Shell (SSH) is a cryptographic protocol that allows the use of a secure connection over an unsecured network to connect one machine to another. To establish the successful connection of the devices connected as nodes over a parallel system, verification of SSH must be implemented before any process is carried out over the network [21].

To call processes on many nodes inside the cluster, a minor change to the processor call command was made at this point to ensure that all of the cores on the master node have been successfully called before any test cases can be run in parallel. Figure 2 exhibits the number of 8 processes distributed over two nodes with four cores each that will be used for calculation.

```
pi@RPAR-1:~/Desktop/MPItestcases/call-procs $ mpiexec -H RPAR-1,RPAR-1,RPAR-1,RP
AR-1,RPAR-2,RPAR-2,RPAR-2,RPAR-2 call-procs
pi@rpar-2's password:
Calling process 0 out of 8 on RPAR-1
Calling process 1 out of 8 on RPAR-1
Calling process 2 out of 8 on RPAR-1
Calling process 3 out of 8 on RPAR-1
Calling process 4 out of 8 on RPAR-2
Calling process 5 out of 8 on RPAR-2
Calling process 6 out of 8 on RPAR-2
Calling process 7 out of 8 on RPAR-2
```

Fig. 2. Call processors attempt for two nodes of Raspberry Pi 3B+

Because this type of calculation can be spread across numerous cores and nodes and conducted multiple times on each machine, Monte Carlo methods are used to do a basic pi calculation [22]. The program calculates the value of pi over 3×10^5 iterations, and the execution time is measured using the 'time mpiexec' command to generate an output consisting of 'real time', 'user time', and 'sys time'. The execution time decreases with the number of cores, as observed in Fig. 3, but there is a drop in the increase of efficiency from 48.1% from one core to two cores, 17.17% from two cores to three cores, and 8.43% from three cores to four cores.

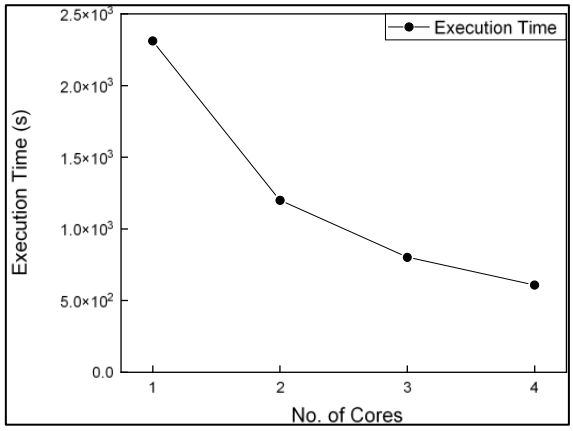

Fig. 3. Graph of execution time (s) against no. of cores

The effectiveness of the four cores is not as expected, as shown in Fig. 3, as it fell below the expected percentage of greater than 80%. This is explained by parallel system scalability rules or Amdahl's law, which states that increasing the number of cores reduces the efficiency of a parallel system [23].

2.4 Grid Generation and Solution of Laplace Equation on Steady Heat Transfer via Jacobi Iteration

The Jacobi Iteration Program was used to investigate each Raspberry Pi 3 B+ node's ability to produce and output a data file based on a processing load, calculation time, and overall execution time. Using the Laplace Equation, the programme solves the discretisation via finite-difference of a square domain [24]. Figure 4 shows the initiation of the Jacobi Iteration with the solution parameters given by the user in the application. By splitting the domain for the investigation of the stable heat equation, the iteration was divided among the processors.

The designations given to the allocated processors connected in parallel for ease of identification are 'RPAR-1' and 'RPAR-2.' The first node contains RPAR-1, whereas the second node has RPAR-2. Because each node has four cores, the declaration of each core is required after the 'time mpiexec'

instruction. The numbers 'nx global' and 'ny global' represent the number of nodes in the x- and y-coordinate directions, respectively.

```
pi@RPAR-1:~/Desktop/MPItestcases/jacobi_mpi $ time mpiexec -H RPAR-1,RPAR-1,RPAR
-1,RPAR-1,RPAR-2,RPAR-2,RPAR-2,RPAR-2 jacobi_mpi
pi@rpar-2's password:
  npes =            8  mype =            0

    nx_global =          512
    ny_global =          512
  Total nodes =       262144

          nx_local =          512
          ny_local =           64
  Total local nodes =        32768
```

Fig. 4. Initiation of Jacobi Iteration

The calculation involves a square domain, which yields a total of 262144 nodes by multiplying 'nx global' and 'ny global'. The term 'npes' refers to the number of cores or processors connected in this simulation between two nodes of the devices. Figure 5 depicts the steady heat equation's temperature distribution, which exhibits an increase in temperature along the x- and y-axes, with the maximum temperature at x=1.0 and y=0.5. Figure 5 shows the approximate temperature calculation using finite difference discretisation in the square domain.

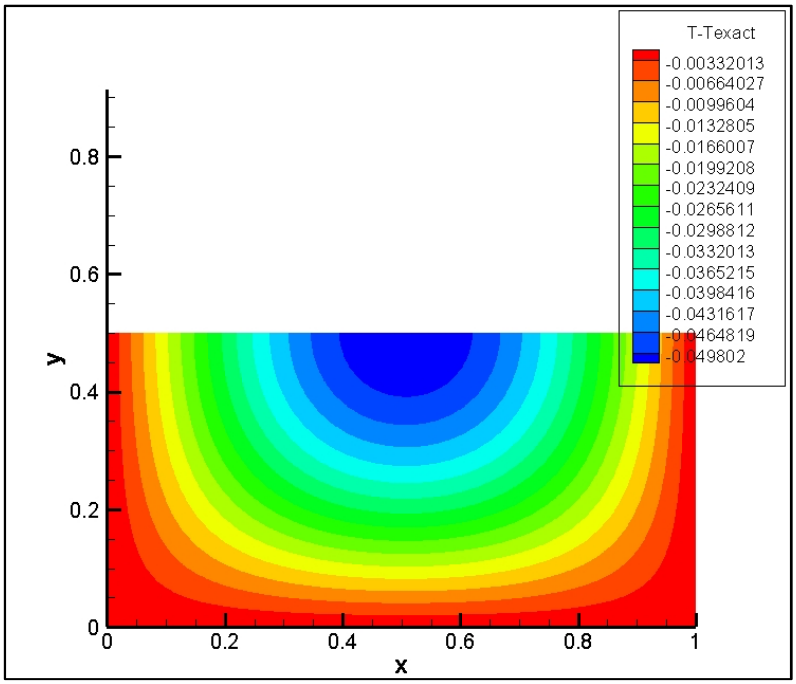

Fig. 5. Graph of execution time (s) against no. of cores

It took 1794.206 seconds to generate a grid and calculate the temperature and exact temperature distribution of a square domain via Jacobi Iteration based on the reading of 'real-time. In addition, the influence of the number of loads processed by a single node in the calculation based on the 'nnodes' can be studied using Jacobi Iteration. For example, in a square domain, a single core of a Raspberry Pi computed a total of 65536 nodes, while four cores computed a total of 262144 nodes. The parameters of the solution in two Raspberry Pi nodes and the reduction in computing time, are shown in Table 2.

Table 2
Execution time and parallelisation efficiency of Jacobi Iteration

No. of Devices	No. of Cores	Nodes Processed per Core	Execution Time (s)	Efficiency of Parallelisation
1	4	65536	1794.206	-
2	8	32768	1009.086	43.76

3. Computational Fluid Dynamics Procedure

3.1 Parameters of the Case Study

The parameters of an airfoil and corresponding properties of air at 1atm pressure and temperature of 25°C are shown in Table 3.

Table 3
Properties of NACA 0012 airfoil and air at 1atm pressure of 25°C

Airfoil nomenclature	NACA 0012
Chord length, c	1m
The angle of attack, α	0°, 10°, 15°
Reynolds number, Re	3×10^6, 6×10^6, 9×10^6
Dynamic viscosity, μ	1.184×10^{-5} m^2s^{-1}
Kinematic viscosity, v	1.562×10^{-5} m^2s^{-1}

3.2 Grid Generation of NACA 0012 Airfoil

The first step in running the Finite Volume Method code for airfoil validation is to generate the two-dimensional grid. The grid is an unstructured mesh constructed following the theory of grid computation via the Joukowsky transformation formula [25].

$$Z = g_1(\zeta) = \zeta + \frac{\iota^2}{\zeta} \tag{2}$$

The grid is generated using the Raspberry Pi parallel system's execution of program code on an exact solution. The exact solution generates an output file in a grid format that is visible via '0012.GRID'. After that, the file is visualised using the Tecplot software. The grid must follow the tolerance set by NASA's OVERFLOW Turbulence Modelling Resource so that the calculation over the generated grid will not produce undesirable and unexpected results [26]. The codes will be improvised and executed again until a proper unstructured grid is generated.

3.3 Solver Code

The Courant-Friedrichs-Lewy (CFL) number, the maximum time steps for iteration, the type of inviscid flux, the limiter, the number of variables in the target equation gradient, the Least-Square (LSQ) gradient, the gradient weight, and the time is taken to stop the calculation are specified as input parameters [27].

A boundary condition map must be prepared in a file to specify the boundary between the fluid flow and the solid body in the generated grid of the airfoil. The original solver comes with an example of a study case which is obtained from Masatsuka [28]. The boundary condition parameters in the file are identified and adjusted according to the study case of this research, as specified in Table 2.

The initial solutions are computed by considering the infinity velocity on the x-axis, the infinity velocity on the y-axis, the density, and the pressure. The analysis in this case study makes use of the Roe approximation Riemann solver, which Puri and Ramachandran deduced for airfoils using the following equation [29].

$$F^* = \frac{1}{2}\left[F_l + F_r - \sum_f r_{lr}^k \left|\lambda_{lr}^k\right|(a_r - a_l)\right] \quad (3)$$

$$p^* = \frac{1}{2}\left[p_l + p_r - \frac{1}{c_{lr}}(u_r - u_l)\right] \quad (4)$$

The computational residual is calculated using the Roe solver, and the iteration will stop once it converges based on the input parameters. To produce the result in a readable format, a subroutine will be created that will generate a '.DAT' output file that can be visualised using TECPLOT. 'DAT' output file format resembling '0012.DAT' will be produced in the folder. It can be opened and viewed via the TECPLOT software and compared with the visual result of ANSYS CFX [30]. If the visual result does not match the programme, debugging processes will be executed to update and compile the source code until the proper result is reached. The parallel computing test can be run once both findings are in agreement.

3.4 NACA 0012 Airfoil Validation

This case is specifically chosen for its ability to portray the numerical analysis of turbulence modelling in terms of its convergence properties and order of accuracy. The case is executed at an incompressible condition in which the Mach number $Ma = 0.15$ and the compressibility effects are ignored since theoretically, the compressibility effects of the flow will be significant for $Ma > 0.3$ [31]. The Reynolds number used is $Re = 6 \times 10^6$, and turbulent boundary layers are implemented over most of the airfoil. The results for the pressure over the NACA 0012 airfoil at the angle of attack, α of 0°, 10°, and 15° have been obtained experimentally, as shown in Fig. 6.

Ladson and Johnson's surface pressure coefficients measured at Langley Research Centre (LaRC) utilising the 0.3 Meter Transonic Cryogenic Tunnel (0.3M TCT) do not appear to resolve the leading edge upper surface pressure efficiently [32]. On the other hand, the surface pressure coefficients of Gregory and O'Reilly appear to be more resolved, with considerable differences on the airfoil's leading edge at an angle of attack, α of 10° and 15°. Furthermore, as shown in Fig. 6, Gregory's data are considered more two-dimensional and thus more acceptable for CFD validation of surface pressure coefficients [33].

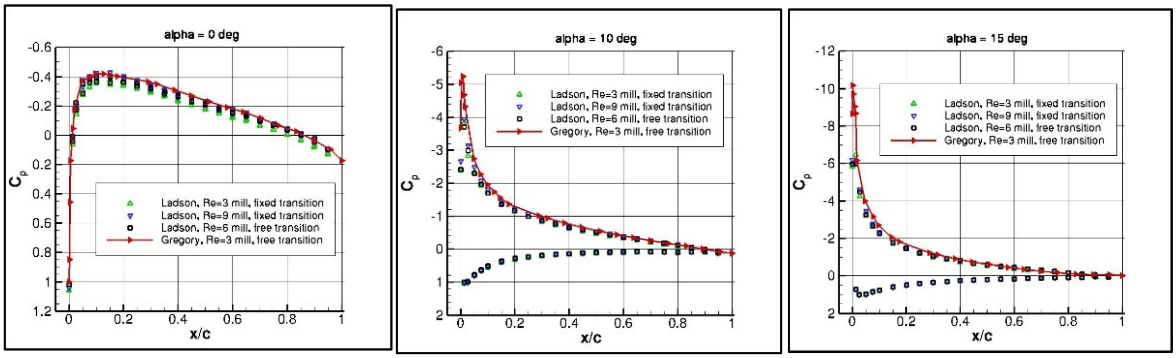

Fig. 6. Pressure coefficient on the angle of attack of; (a) 0°, (b) 10°, and (c) 15°

Due to the shape of NACA 0012, which has symmetrical geometry of the upper and lower surface, this affects the outcome of the graph as exhibited in Fig. 7 to Fig. 9, where the pressure coefficient distribution over X/c is not 'looping'. The trend in the graph signifies that high negative pressure distribution was developed at the airfoil's leading edge. Along with the non-dimensional distance, X/c of the airfoil, the distribution of pressure increases from negative to positive towards the trailing edge. The Finite Volume Method code obtained an overshoot graph line at the leading edge due to the limitation of the solver code. The trend of both graphs confirms each other even though there is a slight overshoot.

Validation of this experimental data is the overall result of the pressure coefficient formula, which involves pressure at each node, infinite pressure, infinite velocity, and infinite density. The validation of the experimental data confirms and establishes the finding of the ability of the CFD model to emulate the actual pressure distribution over the NACA 0012 airfoil. The discrepancies in the leading-edge pressure distribution are explained by the error obtained from other validation cases.

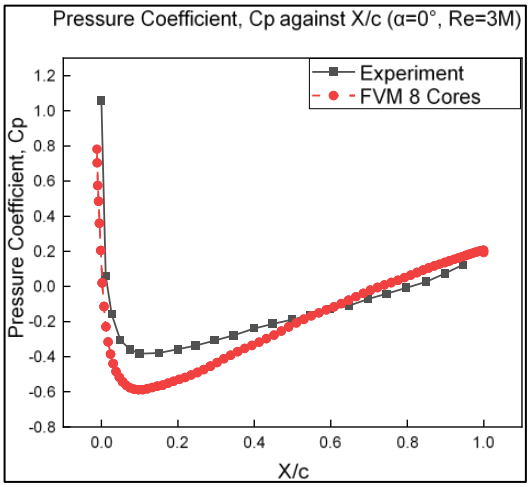

Fig. 7. Pressure Coefficient at Re = 3×10^6

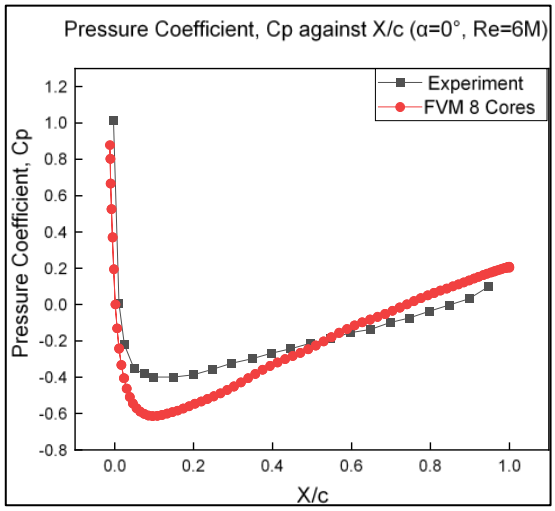

Fig. 8. Pressure Coefficient at Re = 6×10^6

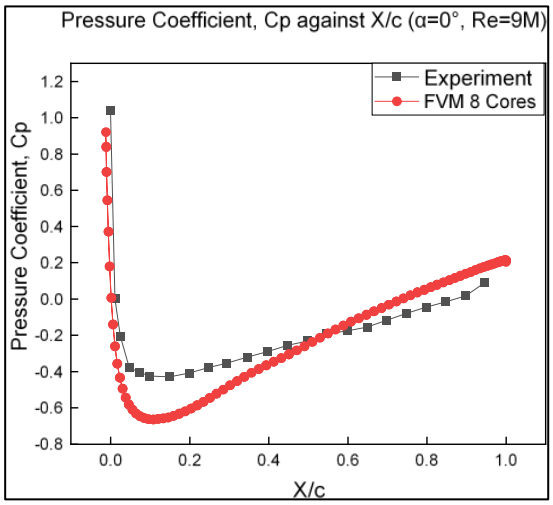

Fig. 9. Pressure Coefficient at Re = 9×10^6

The limitation of the FVM codes persists due to the definition of NACA 0012, which was modified from the original definition to end the chord of the airfoil at X/c = 1 in non-dimensional form. The airfoil simulation model is also scaled down by 1.008930411365 to produce the perfect scaled-down copy of NACA 0012. The maximum thickness of the NACA 0012 airfoil is 12% relative to the blunted chord. At the same time, the scaled-down airfoil has an approximate 11.894% relative to the chord. The FVM codes also implemented one inviscid Roe Solver.

Previous researchers applied the k-epsilon model which includes boundary layer solver, inviscid solver, and turbulence solver on CFD simulation software such as ANSYS CFX. The result of ANSYS CFX has higher accuracy to the experimental data compared with FVM codes [30]. But due to the limitation of the 1.4 GHz Quad-core processors of the Raspberry Pi 3 B+, it does not allow for graphical intensive software such as ANSYS to run. Therefore, errors and discrepancies were caused by the limitation of FVM codes, not hardware.

4. Execution Time for NACA 0012 Validation

The implementation of parallel computing over the devices allows for a total of 8 cores with 1.4Ghz each executing a modified FVM code which has been altered with MPI protocol. The alteration makes it possible for the execution of the codes written in Fortran to be paralleled with the result produced on the master node. To test the paralleling ability of Raspberry Pi, it is redundant to vary the Reynolds number as the graph shown in the previous subtopic shows the Reynolds number heavily impacts the value of the pressure coefficient. In Figure 10, the implementation of two devices observed an overall reduction in computation time.

At an angle of attack α of 0°, the time taken to solve the pressure distribution is the highest. This is explained due to the symmetrical nature of NACA 0012, where α=0° requires intensive calculation for the upper surface of the airfoil compared to the lower. As the angle of attack α increases, the velocity is distributed more on the upper surface of the airfoil than on the lower, resulting in a decrease in computational needs over the grid of the lower airfoil.

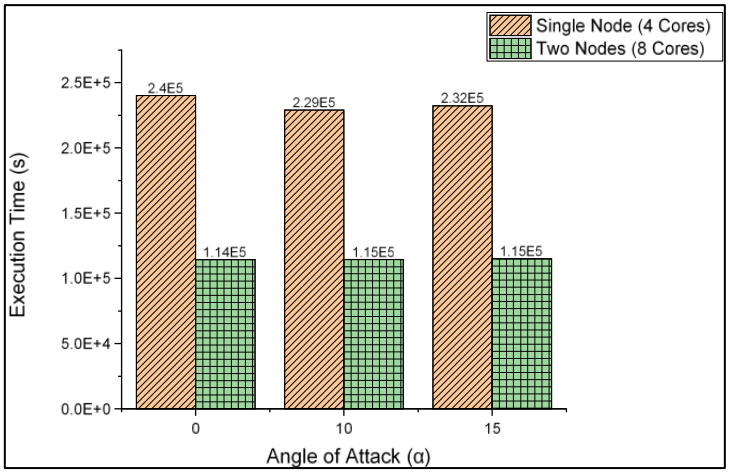

Fig. 10. Execution time (s) according to the angle of attack, α and no. of cores

Referring to Figure 10, the efficiency of parallelisation is at its lowest on the angle of attack α=10°. This is because solving the high-pressure distribution at the airfoil's lower leading edge necessitates a lot of computing resources. Therefore, the computational simulation is slightly targeted towards the decreasing pressure distribution of the lower airfoil body as the angle of attack approaches 15°.

Because few extrinsic effects have affected the result, the experiment is performed three times to acquire the best computation time. For the overall angle of attack, the efficiency of parallel devices is increased by an average of 50%, with 15° having the longest execution time.

When the computational time of FVM codes on two Raspberry Pi nodes is compared, the time required for a single Raspberry Pi cluster is cut in half when another node is added. It can be inferred that the Raspberry Pi's ARM Architecture's Reduced Instruction Set Computing can handle sophisticated Fortran calculations and that processor parallelism improves the calculation's efficiency in terms of processing time.

The cluster overheats while calculating FVM codes and throttles down to half speed. Heatsinks and active cooling were added to both devices to address the issues. The consequences of overheating can be studied thanks to the case studies' frequent testing. Disabling most of the device's unnecessary background processes is one of the procedures followed. In addition, the computational power is impacted by the use of cooling fans, which drain a large amount of current from the equipment. Consequently, the devices must be supplied with a constant amperage at all times.

5. Conclusion

The results obtained from the validation proved that the parallel system can be utilised for flow problems and represents the ability of the device to function similar to Complex Instruction Set Computing (CISC) processors and generate solved outputs and data to be analysed by the users. To authenticate and confirm that the cores and nodes are working in parallel, processors call attempt was used that individually called out each core within the cluster. The first parallelisation test case of Monte Carlo simulation was used, in which a reduction of execution time from 2311s for one core, reduced to 607s for four cores. Jacobi Iteration allows the generation of grid to be segregated and tested over the parallel clusters, in which 512 points over x-axis and 64 points over y-axis were distributed to the eight cores. A calculation of Jacobi Iteration took 1794s for four cores (one node) and 1009s for eight cores (two nodes) of Raspberry Pi 3B+. The NACA 0012 validation was aimed to test parallelability of the Roe solver codes. Discrepancies were observed on the airfoil's leading edge due to the limitation of the solver codes that are unable to emulate the two-dimensionality of the experimental data at the location of the leading edge, the parameters of the surface skin friction was also not considered. The validation case yielded 23275s for a node of Raspberry Pi and 114987s for two nodes, recording an overall parallelisation efficiency of 50%.

Acknowledgement

The authors would like to thank the Ministry of Higher Education Malaysia for supporting this research under Fundamental Research Grant Scheme Vot No. FRGS/1/2020/STG06/UTHM/02/3 and partially sponsored by Universiti Tun Hussein Onn Malaysia.

References

[1] Tey, Wah Yen, Yutaka Asako, Nor Azwadi Che Sidik, and Rui Zher Goh. "Governing equations in computational fluid dynamics: Derivations and a recent review." *Progress in Energy and Environment* 1 (2017): 1-19.

[2] Raase, Sebastian, and Tomas Nordström. "On the use of a many-core processor for computational fluid dynamics simulations." *Procedia Computer Science* 51 (2015): 1403-1412. https://doi.org/10.1016/j.procs.2015.05.348

[3] Ljungskog, Emil, Simone Sebben, and Alexander Broniewicz. "Inclusion of the physical wind tunnel in vehicle CFD simulations for improved prediction quality." *Journal of Wind Engineering and Industrial Aerodynamics* 197 (2020): 104055. https://doi.org/10.1016/j.jweia.2019.104055

[4] Wang, Yong-Xian, Li-Lun Zhang, Wei Liu, Xing-Hua Cheng, Yu Zhuang, and Anthony T. Chronopoulos. "Performance optimizations for scalable CFD applications on hybrid CPU+ MIC heterogeneous computing system

with millions of cores." *Computers & Fluids* 173 (2018): 226-236. https://doi.org/10.1016/j.compfluid.2018.03.005

[5] Oyarzun, Guillermo, Ricard Borrell, Andrey Gorobets, Filippo Mantovani, and Assensi Oliva. "Efficient CFD code implementation for the ARM-based Mont-Blanc architecture." *Future generation computer systems* 79 (2018): 786-796. https://doi.org/10.1016/j.future.2017.09.029

[6] Yokoyama, Daniel, Bruno Schulze, Fábio Borges, and Giacomo Mc Evoy. "The survey on ARM processors for HPC." *The Journal of Supercomputing* 75, no. 10 (2019): 7003-7036. https://doi.org/10.1007/s11227-019-02911-9

[7] Pajankar, Ashwin. "Raspberry pi supercomputing and scientific programming." Ashwin Pajankar (2017). https://doi.org/10.1007/978-1-4842-2878-4

[8] Cox, Simon J., James T. Cox, Richard P. Boardman, Steven J. Johnston, Mark Scott, and Neil S. O'brien. "Iridis-pi: a low-cost, compact demonstration cluster." *Cluster Computing* 17, no. 2 (2014): 349-358. https://doi.org/10.1007/s10586-013-0282-7

[9] Ross, James A., David A. Richie, Song J. Park, and Dale R. Shires. "Parallel programming model for the epiphany many-core coprocessor using threaded mpi." In Proceedings of the 3rd International Workshop on Many-core Embedded Systems, pp. 41-47. 2015. https://doi.org/10.1145/2768177.2768183

[10] Guthrie, James John. "CFD simulations on a Raspberry Pi cluster." (2015).

[11] Luntovskyy, Andriy. "Performance and energy efficiency in distributed computing." In International Multi-Conference on Advanced Computer Systems, pp. 281-292. Springer, Cham, 2016. https://doi.org/10.1007/978-3-319-48429-7_26

[12] Bessonov, Oleg, and Bernard Roux. "Optimization techniques and performance analysis for different serial and parallel RISC-based computers." In International Conference on Parallel Computing Technologies, pp. 168-174. Springer, Berlin, Heidelberg, 1997. https://doi.org/10.1007/3-540-63371-5_18

[13] Manke, Joseph W., G. David Kerlick, David Levine, Subhankar Banerjee, and Eric Dillon. "Parallel performance of two applications in the Boeing high performance computing benchmark suite." *Parallel Computing* 27, no. 4 (2001): 457-475. https://doi.org/10.1016/S0167-8191(00)00070-3

[14] Kačeniauskas, Arnas, and Peter Rutschmann. "Parallel FEM software for CFD problems." *Informatica* 15, no. 3 (2004): 363-378. https://doi.org/10.15388/Informatica.2004.066

[15] Hennessy, John L., and David A. Patterson. "A new golden age for computer architecture." *Communications of the ACM* 62, no. 2 (2019): 48-60. https://doi.org/10.1145/3282307

[16] Nath, Omkar. "Review on raspberry pi 3b+ and its scope." *Int. J. Eng. Appl. Sci. Technol*. 4, no. 9 (2020): 157-159. https://doi.org/10.33564/IJEAST.2020.v04i09.020

[17] Benoit-Cattin, Théo, Delia Velasco-Montero, and Jorge Fernández-Berni. "Impact of thermal throttling on long-term visual inference in a CPU-based edge device." *Electronics* 9, no. 12 (2020): 2106. https://doi.org/10.3390/electronics9122106

[18] Nugroho, S., and A. Widiyanto. "Designing parallel computing using raspberry pi clusters for IoT servers on apache Hadoop." In Journal of Physics: Conference Series, vol. 1517, no. 1, p. 012070. IOP Publishing, 2020. https://doi.org/10.1088/1742-6596/1517/1/012070

[19] Kent, Brian R. Science and Computing with Raspberry Pi. Morgan & Claypool Publishers, 2018. https://doi.org/10.1088/978-1-6817-4996-9

[20] Dennis, Andrew K. Raspberry Pi super cluster. Packt Publishing Ltd, 2013.

[21] K. Doucet and J. Zhang, 'The creation of a low-cost raspberry Pi cluster for teaching', in *Proceedings of the 24th Western Canadian Conference on Computing Education, WCCCE 2019*, May 2019, pp. 1–5. doi: 10.1145/3314994.3325088. https://doi.org/10.1145/3314994.3325088

[22] Dorr, Greg, Drew Hagen, Bob Laskowski, Erik Steinmetz, and Don Vo. "Introduction to parallel processing with eight node Raspberry Pi cluster." In Midwest Instruction and Computing Symposium (MICS), The University of Wisconsin–La Crosse in La Crosse, pp. 7-8. 2017.

[23] Bourhnane, Safae, Mohamed Riduan Abid, Khalid Zine-Dine, Najib Elkamoun, and Driss Benhaddou. "High-Performance Computing: A Cost Effective and Energy Efficient Approach." *Adv. Sci. Technol. Eng. Syst. J* 5 (2020): 1598-1608. https://doi.org/10.25046/aj0506191

[24] Yang, Wenxiang, Jiming Zou, and Liang Deng. "Optimization of Jacobi Iteration on the Intel Xeon Phi." In Proceedings of the 2019 3rd High Performance Computing and Cluster Technologies Conference, pp. 1-5. 2019. https://doi.org/10.1145/3341069.3341071

[25] Plotkin, Allen. "Low-Speed Aerodynamics- the theoretical aspects." *International Journal of Aerodynamics* 2, no. 1 (2012): 1-21. https://doi.org/10.1504/IJAD.2012.046564

[26] Jespersen, Dennis C., Thomas H. Pulliam, and Marissa Lynn Childs. Overflow turbulence modeling resource validation results. No. ARC-E-DAA-TN35216. 2016.

[27] Yusoff, Hamid, Koay Mei Hyie, Halim Ghaffar, Aliff Farhan Mohd Yamin, Muhammad Ridzwan Ramli, Wan Mazlina Wan Mohamed, and Siti Nur Amalina Mohd Halidi. "The Evolution of Induced Drag of Multi-Winglets for Aerodynamic Performance of NACA23015." *Journal of Advanced Research in Fluid Mechanics and Thermal Sciences* 93, no. 2 (2022): 100-110. https://doi.org/10.37934/arfmts.93.2.100110

[28] Masatsuka, K. "I do like CFD. VOL. 1: Governing equations and exact solutions." Katate Masatsuka, Lulu. com (2009).

[29] Puri, Kunal, and Prabhu Ramachandran. "Approximate Riemann solvers for the Godunov SPH (GSPH)." *Journal of Computational Physics* 270 (2014): 432-458. https://doi.org/10.1016/j.jcp.2014.03.055

[30] M. S. A. Paisal, 'Numerical simulation of flow over airfoil using open source algorithm', Universiti Tun Hussein Onn Malaysia, Batu Pahat, 2016.

[31] Sankar, N. L., and Y. Tassa. "Compressibility Effects on Dynamic Stall of an NACA 0012 Airfoil." *AIAA Journal* 19, no. 5 (1981): 557-558. https://doi.org/10.2514/3.50976

[32] Ladson, Charles L., Acquilla S. Hill, and William G. Johnson Jr. Pressure distributions from high Reynolds number transonic tests of an NACA 0012 airfoil in the Langley 0.3-meter transonic cryogenic tunnel. No. NASA-TM-100526. 1987.

[33] Gregory, Nigel, and C. L. O'reilly. "Low-speed aerodynamic characteristics of NACA 0012 aerofoil section, including the effects of upper-surface roughness simulating hoar frost." (1970).