High Performance Linpack (HPL) Benchmark on Raspberry Pi 4B (8GB) Beowulf Cluster

Dimitrios Papakyriakou
PhD Candidate
Department of Electronic Engineering
Hellenic Mediterranean University
Crete, Greece

Ioannis S. Barbounakis
Assistant Professor
Department of Electronic Engineering
Hellenic Mediterranean University
Crete, Greece

ABSTRACT
This paper focuses on a High Performance Linpack (HPL) benchmarking performance analysis of a state of the Art Beowulf cluster deployed with 24 Raspberry Pi's 4 (model B) (8GB RAM) computers with a CPU clocked at 1.5 GHz, 64-bit quad-core ARMv8 Cortex-A72. In particular, it presents the increased HPL performance of a Beowulf cluster with the use of the default microSD usage in all the RPi's in the cluster (SDCS2 64GB micro SDXC 100R A1 C10) compared to using a cluster set-up where the master-node uses a Samsung (1TB) 980 PCI-E 3 NVMe M.2 SSD and the slave-nodes uses each a (256GB) Patriot P300P256GM28 NVME M.2 2280).

Moreover, it presents the test results of a multithread execution of a C++ pi calculation program by using one to four cores in one RPi 4 B (8GB) using the above-mentioned microSD. In addition, it presents the test results of a multithread execution of a C++ with MPI (pi) calculation program by using 24 RPi's 4B with the above-mentioned microSD.

In terms of the HPL benchmarking performance testing of a Beowulf cluster where the NVMe M.2 SSD disks are used, RPi 4-B supports and deployed the option to use the entire SSD (MVMe) as a bootable external disk which the boot and root partition (where the actual HPL runs) is hosted in the external SSD. All of them are connected over two Gigabit switches (TL-SG1024D) in a parallel mode of operation so that to build a supercomputer.

Keywords
Raspberry Pi 4 cluster, Beowulf Cluster, Message Passing Interface (MPI), MPICH, BLAS, High Performance Linpack (HPL), Benchmarking HPL RPi clusters, Distributed Systems.

1. INTRODUCTION
The first Beowulf was developed in 1994 by Don Becker and Thomas Sterling at the Center of Excellence in Space Data and Information Sciences (CESDIS), a contractor to NASA at the Goddard Space Flight Center in Greenbelt, Maryland [1]. Beowulf is a way of building a supercomputer consisting a group of smaller computers which are connected and working together by a high-speed local area network (LAN) usually an Ethernet. A Beowulf Cluster in practice is usually a collection of generic computers, either stock systems or wholesale parts purchased independently and assembled, connected through an internal network. A Beowulf Cluster has two types of computers in the architecture, a master-node computer, and the slave-node computers. When a large problem or set of data is given to a Beowulf cluster, the master-node computer first runs a program that breaks the problem into small discrete pieces; it then sends a piece to each slave-node to compute. As slave-nodes finish their tasks, the master-node computer continually sends more pieces to them until the entire problem has been computed [2]. In order for the master-node and slave-node computers to communicate, some sort message passing control structure is required so as to control where the Message Passing Interface (MPI), is the most prominent programming model used in scientific computing today and the most commonly used. In this survey, the MPICH used which is a high performance and widely portable implementation of the MPI standard [3], [4].

Traditional high-performance computing (HPC) systems use CPUs for double-precision floating-point computing, while emerging supercomputing systems use CPUs, Graphics Processor Units (GPUs), and field-programmable gate array (FPGAs) for more powerful parallel computing [5], [6].

Supercomputing is measured in floating-point operations per second (FLOPS). In this manuscript the High-Performance Linpack (HPL) Benchmark for distributed-memory computers is used where HPL rely on an efficient implementation of the Basic Linear Algebra Subprograms (BLAS) [7], [8]. OpenBLAS library is an open-source optimized BLAS (Basic Linear Algebra Subroutines) library, which contains a set of routines that provide matrix/vector linear algebra functions.

Raspberry Pi (RPi) 4 Model B with 8GB ram "Figure 1" is used in the Beowulf cluster which is equipped with a CPU processor (64-bit quad-core ARMv8 Cortex-A72, 1.5 GHz) three times more powerful than the RPi 3B+ model [9], [10]. The low cost of the Raspberry Pi was the driving force to investigate a viable option for building a high-performance cluster computer and to study the Pi's ability to perform in a parallel clustering mode of operation.

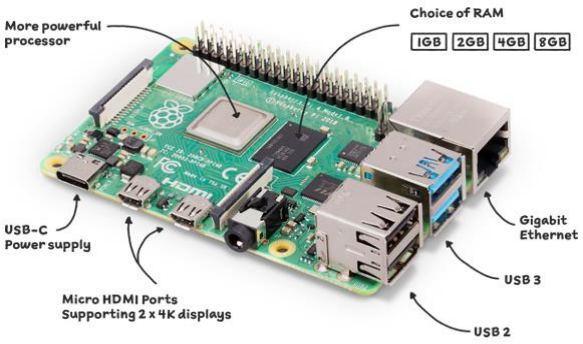

Figure 1: Single Board Computer (SBC) - Raspberry Pi 4 Model B [9].

2. SYSTEM DESCRIPTION

2.1 Hardware Equipment

The Beowulf cluster is composed of 24 Raspberry Pi 4's "Figure 2". One RPi 4B (8GB) is deployed to be the master (or head) node of the cluster, responsible for distributing jobs and resources and the rest 23 RPi's are simply the worker nodes obeying in the master node instructions. All the nodes are stacked together in four groups of 6 RPi's each, and are connected to two Gigabit switch (TL-SG1024D) where the maximum LAN network throughput for any individual node is 1000 Mbps. All of them are connected over two Gigabit switch (TL-SG1024D so that to build a kind of supercomputer. The whole cluster is powered by two switch-mode power supplies (60 Amp each) with 5V output boosted to 5.56V so as to adjust the voltage drop through the wires.

Each RPi requires a microSD card for booting and operating purposes. The size of each microSD card is 64GB. The used microSD is the Kingston SDCS2 64GB micro SDXC 100R A1 C10 with maximum 100MB/sec read (UHS-I Speed class 1 (U1).

In the second stage of benchmarking a Samsung (1TB) 980 PCI-E 3 NVMe M.2 SSD external disk is hosted in the master-node where the boot and root partition is in there and the slave-nodes host each one of a (256 GB) Patriot P300P256GM28 NVME M.2 2280 as external disk. At this stage of the test the external NVMe is the only bootable disk and the microSD is not used.

Figure 2: Deployment of the Beowulf Cluster with (24) RPi-4B (8GB)

2.2 Software Tools

The Operating System used to setup the RPi's in the cluster is the Raspbian GNU/Linux 10 (buster) which is one of the official supported Operating System (OS) with system 32 bit, and Kernel version 6.1.34-v8+ [11].

The 2nd Software Package we needed to install is the Message Passing Interface (MPI) where the MPICH is used. MPICH is a high performance widely portable implementation of the Message Passing Interface (MPI) which is the most widely used implementations of MPI in the world. The MPI is not a library but a standard for development of message-passing libraries based on recommendations of the MPI Forum. There are two prominent implementations of MPI that can be used on the Raspberry Pi. These are: OpenMPI and MPICH. In our case we use MPICH, which originally standing for Message Passing Interface Chameleon. It's an implementation of the MPI standard that supports C, C++ and FORTRAN applications. MPICH is a high performance and wide portable implementation of the Message Passing Interface (MPI) standard [12].

The 3th software package we needed to install is the (GNU Compiler Collection) GCC Fortran compiler which has optimization and multi-threading features. It's the default compiler suite in High Performance Computing (HPC).

The 4th SW package we needed to install so that to configure properly the cluster is the OpenBLAS which is a standard BLAS (Basic Linear Algebra Subprograms) library that is used to perform linear algebra operations.

The 5th SW package is the High Performance Linpack (HPL) [13]. The High Performance Linpack benchmark used to measure the performance of a HPC system. HPL is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers. The HPL benchmark is based on the original Linpack benchmark, measuring performance based on solving a system of linear equations using LU factorization [14], [15].

2.3 Design

The below mentioned "Figure 3" depicts the RPI cluster architecture diagram.

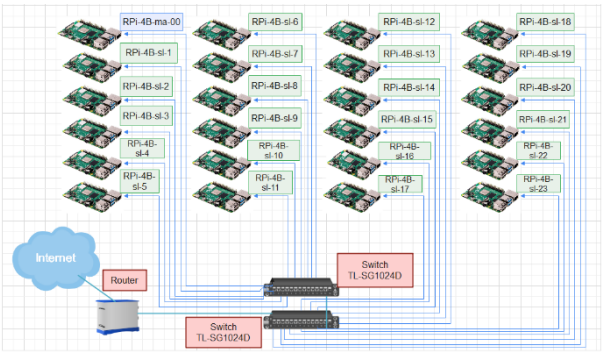

Figure 3: RPi-4B Beowulf cluster architecture diagram.

The Cluster design is composed of 24 Raspberry Pi's 4B with (8GB) memory connected to the 24-Port 1000 Mbps Ethernet switch. One out of the 24 Pi's is the master or head of the cluster and the rest 23 are slaves or workers. The network configuration is built with static addressing where each node has a static IP address and the configuration is in such a way where the master can only communicate to every node with secure shell.

In the first stage of the study the used microSD is the Kingston SDCS2 64GB micro SDXC 100R A1 C10 with maximum 100MB/sec read (UHS-I Speed class 1 (U1). The HPL benchmarking is run in the microSD card as the only bootable disk per master and slave nodes. The HPL benchmarking starts with one node and gradually the involved nodes are increased by three RPi's each time depicting the cluster performance in GFlops with the respective time.

In the second stage of the study, a Samsung (1TB) 980 PCI-E 3 NVMe M.2 SSD external disk is used for the master node where the theoretical maximum write-speed is up to 3000 MB/s and read-speed up to 3500 MB/s. For the rest of the slave nodes (or worker nodes) a (256 GB) Patriot P300P256GM28 NVME M.2 2280 is used as external disk where is the only bootable disk. The particular Patriot NVMe SSD has write-speed up to 1100 MB/s and read-speed 1700 MB/s. At this stage of the testing the external NVMe is the only bootable disks and the

microSD are not used since the RPi 4B support this feature. In this phase of the HPL benchmarking only the external NVMe disks are used where are mounted in USB3.0 in every RPi 4B. Hopefully, RPi 4B supports two USB 3.0 and two USB 2.0. The external SSDs (NVMe's) are connected to USB3.0, where the theoretical transfer speed of USB 3.0 is 4.8 Gbit/s (600MB/s) vs USB2.0 with 480 Mbit/s (60MB/s). In any case write-read capabilities of NVMe's exceeds by far the USB 3.0 data transfer throughputs. As a result, with the use of NVMe's it's expected much better performance of the cluster compared with related research [16].

3. MUNTITHREADED C++ and C++ with MPI

- *Multithreaded C++*. – When we refer to cores in a Central Processor Unit (CPU) we mean the hardware-based processing units within a CPU, while threads are the software-based instructions that can be processed by a CPU. The Raspberry Pi 4B (Broadcom BCM2711) comprises one socket (or chip) with 4 physical cores where each core can support 1 thread "Figure 4". On larger systems, it is common to see multiple threads supported per core. This is an example of simultaneous multi-threading (SMT), or Hyperthreading (HT) on Intel systems enabling multiple threads to run on a physical core. A multicore CPU allows multiple processes to execute simultaneously, or in parallel. A multi-threaded program, while capable of parallel execution, runs concurrently on a system with only a single CPU core. The primary goal of creating multi-threaded programs is to decrease the time of a program's execution. In a program which is perfectly parallelizable, it is possible to distribute the associated to program tasks, equally among all the threads. As a result, if we have a program named (A) which is equally distributed in (t) threads, eventually it will take approximately $(\frac{A}{t})$ time provided that it is executed on (t) cores meaning each core executes one thread.

```
pi@rpi4B-ma-0:~/mpi_testing $ lscpu
Architecture:            armv7l
Byte Order:              Little Endian
CPU(s):                  4
On-line CPU(s) list:     0-3
Thread(s) per core:      1
Core(s) per socket:      4
Socket(s):               1
Vendor ID:               ARM
Model:                   3
Model name:              Cortex-A72
Stepping:                r0p3
CPU max MHz:             1500.0000
CPU min MHz:             600.0000
BogoMIPS:                108.00
Flags:                   half thumb fastmult
vt vfpd32 lpae evtstrm crc32
pi@rpi4B-ma-0:~/mpi_testing $
```

Figure 4: RPi-4B cores and threads supported.

Modern C++ makes launching threads very easy. There is a very important caveat to keep in mind that code in parallel has access to the same memory, hence it is crucial to avoid having multiple threads write to the same memory location. In "Figure 6", we see the compilation and running a C++ code with one to four threads and the respective execution time. "Figure 5" presents the 4 CPU cores involved in code execution when 4 threads are called by the command in the Command Line Interface (CLI).

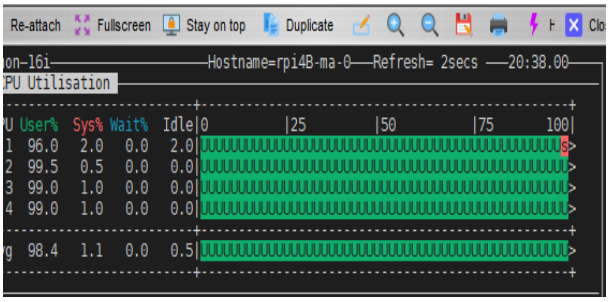

Figure 5: Multithreaded (4 threads) C++ execution time with 4 CPU cores involvement.

```
pi@rpi4B-ma-0:~/mpi_testing $ g++ -O2 pi-threads.cpp -o pi-threads -lpthread
pi@rpi4B-ma-0:~/mpi_testing $
pi@rpi4B-ma-0:~/mpi_testing $
pi@rpi4B-ma-0:~/mpi_testing $ ./pi-threads 1
Using 500000000 samples and 1 threads, pi is 3.14164 in 43.4 seconds
pi@rpi4B-ma-0:~/mpi_testing $
pi@rpi4B-ma-0:~/mpi_testing $
pi@rpi4B-ma-0:~/mpi_testing $ ./pi-threads 2
Using 500000000 samples and 2 threads, pi is 3.1415 in 25 seconds
pi@rpi4B-ma-0:~/mpi_testing $
pi@rpi4B-ma-0:~/mpi_testing $
pi@rpi4B-ma-0:~/mpi_testing $ ./pi-threads 3
Using 500000000 samples and 3 threads, pi is 3.14169 in 17.4 seconds
pi@rpi4B-ma-0:~/mpi_testing $
pi@rpi4B-ma-0:~/mpi_testing $
pi@rpi4B-ma-0:~/mpi_testing $ ./pi-threads 4
Using 500000000 samples and 4 threads, pi is 3.14156 in 14.1 seconds
pi@rpi4B-ma-0:~/mpi_testing $
```

Figure 6: Multithreaded C++ code execution time per (1 to 4) threads in one CPU.

"Figure 6" depicts the multithread execution time per threads and "Figure 7" depicts the respective speedup when we increase the CPU cores in the code execution from (1-4) cores.

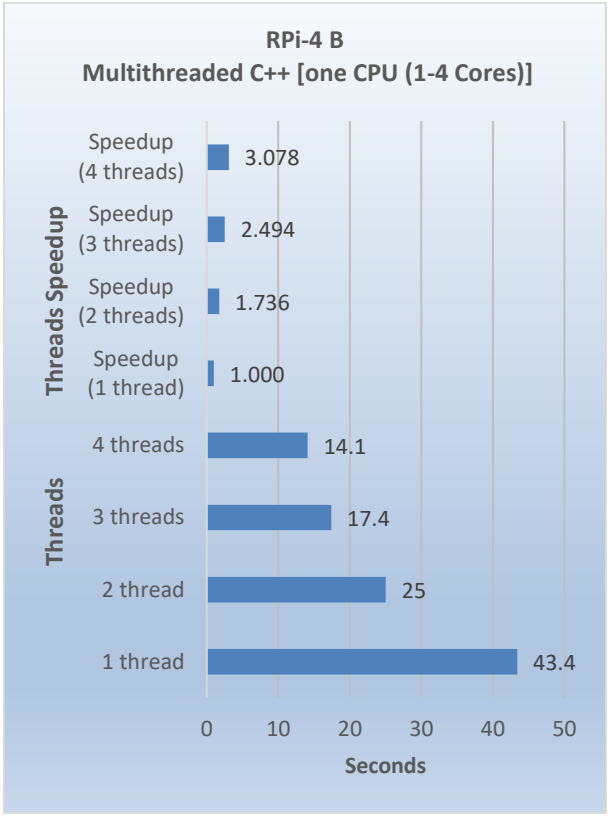

Figure 7: Multithreaded C++ execution time with Speedup performance by increasing CPU cores up to 4.

- *Multithreaded C++ with MPI in one CPU (one RPi-4B).* – The main disadvantage of multithreading is that it's dependent

and limited to the available number of relatively few computing cores on a single CPU. The way to overcome this is to split the computation across multiple physical computers. The solution is the Message Passing Interface (MPI) which is a library that allows different processes to communicate with each other primarily over the network but support multiple processes on the same physical computer as well. "Figure 8" presents an example of a C++ code execution time with the use of MPI and the CPU cores from one to four. "Figure 8" depicts the multithreaded C++ code execution time with the use of MPI per one to four threads (or processes) in one CPU (1 RPi 4B).

```
pi@rpi4B-ma-0:~/mpi_testing $ time mpiexec -np 1 ./pi-mpi
Using 500000000 samples and 1 processes, pi is 3.14159 in 43.4 seconds

real    0m43.458s
user    0m43.388s
sys     0m0.051s
pi@rpi4B-ma-0:~/mpi_testing $ time mpiexec -np 2 ./pi-mpi
Using 500000000 samples and 2 processes, pi is 3.14165 in 21.7 seconds

real    0m21.784s
user    0m43.482s
sys     0m0.053s
pi@rpi4B-ma-0:~/mpi_testing $ time mpiexec -np 3 ./pi-mpi
Using 500000000 samples and 3 processes, pi is 3.14164 in 14.5 seconds

real    0m14.528s
user    0m43.405s
sys     0m0.112s
pi@rpi4B-ma-0:~/mpi_testing $ time mpiexec -np 4 ./pi-mpi
Using 500000000 samples and 4 processes, pi is 3.14157 in 11 seconds

real    0m11.099s
user    0m43.751s
sys     0m0.126s
pi@rpi4B-ma-0:~/mpi_testing $
```

Figure 8: Multithreaded C++ code execution time with MPI per (1 to 4) processes in one CPU

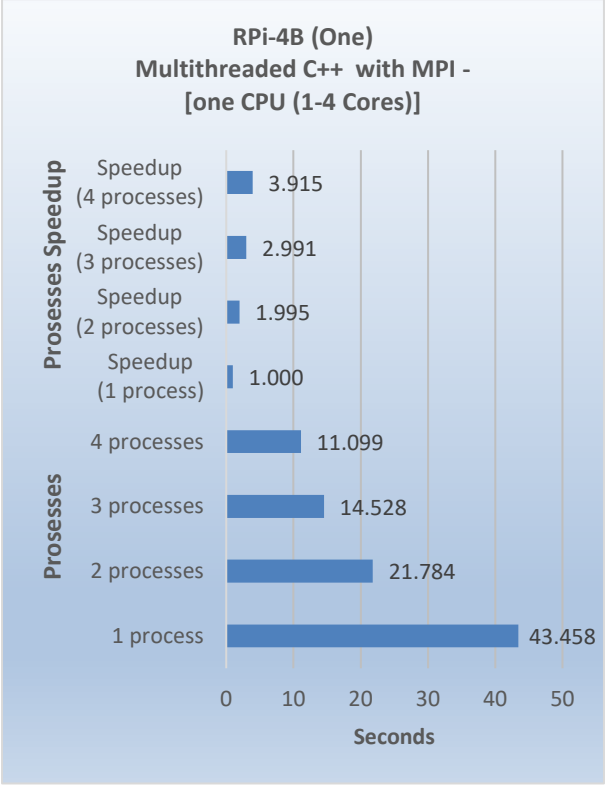

Figure 9: Multithreaded C++ code execution time with MPI Speedup performance per CPU cores up to 4

"Figure 9" depicts the performance results of the C++ code execution with MPI and the respective speedup performance when the involved CPU cores increased from one to four. By using 4 CPU cores (4 processes or threads, - one process per core) the performance is increased according to the "Figure 9". - *Multithreaded C++ with MPI in four CPUs (4 RPi's).* –

As an example, and in terms of multithreaded C++ code execution time with MPI with the involvement of 4 RPi's, that is to say, 4 CPUs with 4 threads (or processes) per CPU, "Figure 10", "Figure 11" shows the results. In particular:
Command "*time mpiexec -f machinefile -np 4 ./pi-mpi"* involves 1 CPU (1 RPi), 4 cores, (1 process per core).
Command "*time mpiexec -f machinefile -np 8 ./pi-mpi*" involves 2 CPUs (2 RPi's), 8 cores, (1 process per core).
Command "*time mpiexec -f machinefile -np 12 ./pi-mpi*" involves 3 CPUs (3 RPi's), 12 cores, (1 process per core).
Command "*time mpiexec -f machinefile -np 16 ./pi-mpi*" involves 4 CPUs (4 RPi's), 16 cores, (1 process per core).

```
pi@rpi4B-ma-0:~/mpi_testing $ time mpiexec -f machinefile -np 4 ./pi-mpi
Using 500000000 samples and 4 processes, pi is 3.14159 in 11 seconds

real    0m11.092s
user    0m43.787s
sys     0m0.054s
pi@rpi4B-ma-0:~/mpi_testing $ time mpiexec -f machinefile -np 8 ./pi-mpi
Using 500000000 samples and 8 processes, pi is 3.14169 in 5.49 seconds

real    0m5.888s
user    0m21.854s
sys     0m0.138s
pi@rpi4B-ma-0:~/mpi_testing $ time mpiexec -f machinefile -np 12 ./pi-mpi
Using 500000000 samples and 12 processes, pi is 3.14164 in 3.65 seconds

real    0m4.110s
user    0m14.655s
sys     0m0.151s
pi@rpi4B-ma-0:~/mpi_testing $ time mpiexec -f machinefile -np 16 ./pi-mpi
Using 500000000 samples and 16 processes, pi is 3.14159 in 2.75 seconds

real    0m3.245s
user    0m11.287s
sys     0m0.109s
pi@rpi4B-ma-0:~/mpi_testing $
```

Figure 10: Multithreaded C++ code execution time with MPI per (1 to 4) in 4 RPi's (4 – 16 CPU cores)

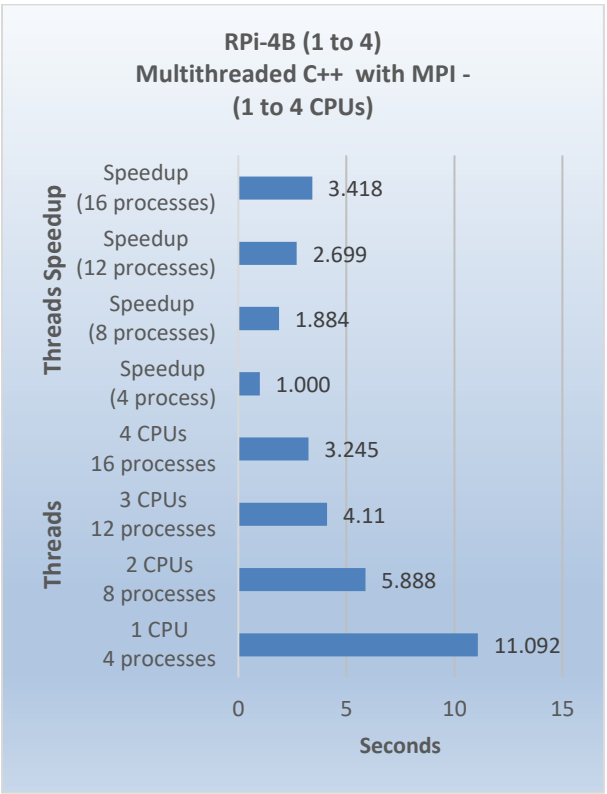

Figure 11: Multithreaded C++ code execution time with MPI and Speedup performance per RPi (4 – 16 CPU cores)

As an observation in terms of the multithreaded C++ code execution time per (1 to 4) threads in one CPU "Figure 6" and "Figure 7" contrary to the multithreaded C++ code execution time with MPI in one CPU (one RPi-4B) gives approximately

the same results with a remark of better performance with the use of MPI.

The multithreaded C++ code execution time with MPI per (1 to 4) RPi's (4 – 16 CPU cores) "Figure 10" and "Figure 11" in comparison with multithreaded C++ code execution time per (1 to 4) threads in one CPU (1 RPi 4B) and with multithreaded C++ code execution time with MPI in four CPU (four RPi-4B with 16 parallel processes) gives a striking better performance of about 70%. This is a justification that the parallel processing introduces much better performance in the multithreaded execution of a program and in particular in C++.

4 HIGH PERFORMANCE LINPACK BENCHMARK

The HPL benchmarking tool is a portable application working across various platforms and suitable for parallel workloads that are core-limited and memory intensive. Linpack is a floating-point benchmark that solves a dense system of linear equations in parallel and determines the upper bound of double precision floating point performance on a distributed parallel system. In other words, measures how fast a computer solves a random dense linear system of equations of order (n), $[A \times x = b;\ A \in R^{nxn};\ x, b \in R^n]$ by first computing the LU factorization [14], [15], with row partial pivoting of $[n\ by\ (n+1)]$ coefficient matrix $[A\ b] = [[\mathrm{L}, \mathrm{U}]\mathrm{y}]$. The left side of the equation $[A \times x = b;\ A \in R^{nxn};\ x, b \in R^n]$ comprises matrix (A), while the right-hand side is a vector (b), where the solution to the problem is established by calculating the factor (A). Provided a matrix (A), and vector (b), the HPL algorithm performs a (LU), factorization calculation through partial pivoting of rows of the matrix $(A\ b = [[L, U]]y])$, with the coefficient of $(n - by - n + 1)$, in order to solve a linear system with the order (n), in equation. To make sure that the load balancing is well-adjusted and the ability to scale to multiple computers, the results of calculation is allocated onto a two-dimensional $(P - by - Q)$ grid of processes and structured using block-cyclic organization. The matrix $(n - by - n + 1)$ coefficient is then segregated into $(NB - by - NB)$ blocks which in turn are intermittently distributed into the $(P - by - Q$ process grid [17]. Dense linear algebra calculations are applicable to many problems and is considered a good method to measure peak performance for a system. The data is distributed onto a two-dimensional grid of processes $(P\ by\ Q)$, according to the block-cyclic scheme, to ensure optimum load balance, as well as the scalability of the algorithm [18], [19]. To determine the scalability of the cluster, the problem size or matrix size (N) was kept constant and the number of processors was gradually varied from 4 (1 RPi) to 96 (24 RPi). In order to achieve the best performance possible, it is needed to define accurately some critical parameters in HPL.dat file, focusing especially in the Number of problems size (N), the Number of the block size (NBs) in the grid and the Number of process grids $(P \times Q)$ [20].

Briefly, the most important parameters in HPL.dat file that we had to configure are analyzed below:

Number of problems sizes (N). – Parameter (N) specifies the problem size and in other words, dictates the size of the matrix to be decomposed. The aim is to find the largest problem size that fits into the main memory of a specific cluster and for this reason, the main memory capacity for storing double precision (8 Bytes) numbers is calculated. A larger problem size engages more processing power in finding the solution and thus resulting in a higher computation speed. The max problem size is calculated as suggests: $N_{max} = Z\sqrt{m \times n}$, where (Z), is the reduction coefficient, taking values between (80-90) percent [21], (m) is the free memory in doubles for the machine with the least available free memory and (n) is the number of nodes. The mathematical expression can be seen as such:

$$\mathrm{N_{max}} = \sqrt{\left(\frac{\text{Memory in Gbytes} \times 1024^3 \times \text{No of Nodes}}{8}\right)} \times \mathrm{Z}, \quad [22].$$

The (N) in overall must consider the (80-90) % of the size of the total memory where (Z), is the reduction coefficient, taking values between (80-90) percent, and as a result we have below:

$$\mathrm{N} = \sqrt{\left(\frac{8\mathrm{GB} \times 1024^3 \times 24}{8}\right)} \times 90\% = 144476.78$$

During the optimization process it is needed to find the (NB) value that gives the best performance. In this case following testing (HPL testing in one RPi) with the range of [96, 104, 112, 120, 128… 256], it has been found that the $(NB = 224)$ gives the best performance and it was chosen that value. The optimization between (N) and (NB) follows as such:

we calculate $\left(\frac{144476.78}{224} = 644.9856\right)$ and next $(224 \times 645 = 144480)$. In any case a round up and round down (or little lower values than 90%) is applied for further optimization in terms of (N) so that not to face up System Memory saturation and then we put these parameters in the HPL.dat. The same logic and optimization take place when executing the benchmark with different values using all the nodes, in an order such as [1, 3, 6, 9, 12, 15, 18, 21, 24] nodes since there is different proportion of the systems' total memory usage within the cluster.

Number of block size (NBs). – NB is the block size which is used for data distribution and data reuse. The distribution size dictates the block size of the problem to be decomposed and distributed among the nodes. As a rule of thumb, small block sizes will limit the performance because there is less data reuse in the highest level of memory and more messaging. On the other hand, when block sizes are too big, there is a waste memory space and extra computation. the data distribution and for the computational granularity. Usually block sizes giving good results are within [96, 104, 112, 120, 128… 256] set of values.

Number of process grids $(P \times Q)$. – $(P \times Q)$ is the size of the grid where P (the number of process rows) and Q (the number of process columns) should be close to being a "square". According to the developers of the (HPL) [23], [24] the (P) and (Q) should be approximately equal, with Q slightly larger than P which is equal to the number of processors that the cluster has. $(P \times Q)$ is the total number of processes that the cluster runs per test phase meaning per involved RPi's nodes.

4.1 HPL Benchmark with microSD

The computing performance vs the number of nodes in the cluster when the microSD is used as bootable disk is depicted in "Table 1". The highest measured HPL performance of the Beowulf cluster with the use of microSD can be seen in "Figure 12" and is about 160 GFlops. Needless to say, that it was taken full attention on updating, and upgrading the RPi's with the latest software releases whereas Kernel version uploaded with the latest version as well (6.1.34-V8+).

Table 1. Beowulf cluster setup parameters and testing results with ≈ 90% System Memory Utilization and microSD disk usage

(≈90%) System Memory Utilization (microSD disk usage) (NB=128)					
N	**Nodes**	**Time (sec)**	**GFlops**	**Speedup**	**Scaling Efficiency (GFlops) (%)**
29440	1	1883.03	9.03	1	100
51200	3	3980.47	22.48	1.49	148.84
72320	6	5720.26	44.08	3.88	387.93
88448	9	7043.71	65.49	6.25	624.93
102144	12	8276.69	85.84	8.50	850.19
114176	15	9347.27	106.16	10.75	1075.12
125184	18	10329.64	126.61	13.01	1301.48
135168	21	11935.96	137.94	14.27	1426.90
144384	24	12491.43	160.64	16.78	1678.17

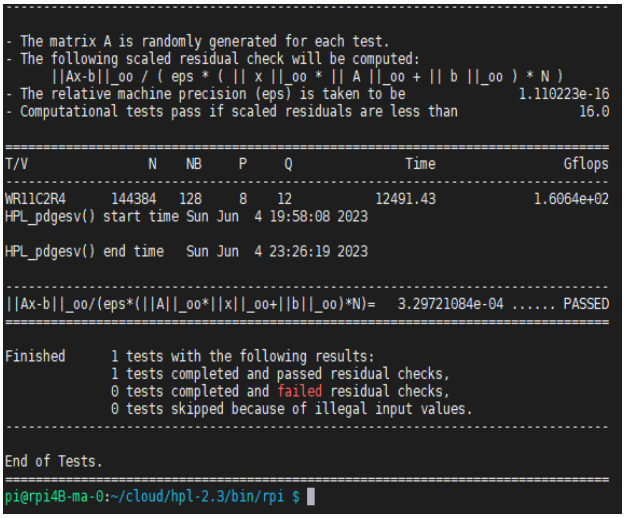

Figure 12: Highest measured HPL performance with (microSD) for the whole cluster (≈ 90% System Memory Utilization).

4.2 HPL Benchmark with NVMe M.2 SSD

The corresponding computing performance vs the number of nodes in the cluster when the NVMe SSD disk is used as bootable disk is depicted in "Table 2". The expected results by the use of NVMe SSDs was to see a better or a much better HPL performance with the restriction of the USB3.0 that RPi 4B supports and indeed there is a decent improvement.

Table 2. Beowulf cluster setup parameters and testing results with ≈ 90% System Memory Utilization and NVMe SSD disk usage

(≈ 90%) memory utilization (NVMe disk usage) (NB=224)					
N	**Nodes**	**Time (sec)**	**GFlops**	**Speedup**	**Scaling Efficiency (GFlops) (%)**
29440	1	1714.29	9.96	1	100
51200	3	3372.66	26.53	1.66	166.37
72320	6	4789.13	52.65	4.29	428.61
88448	9	6024.62	76.56	6.69	668.67
102144	12	7077.49	100.39	9.08	907.93
114176	15	8007.92	123.91	11.44	1144.08
125056	18	8856.18	147.68	13.83	1382.73
135168	21	9760.99	168.67	15.93	1593.47
144384	24	10747.4	186.71	17.75	1774.60

The highest measured value in GFlops when using the NVMe disk can be seen in "Table 2" and "Figure 13" as well which is 186.71 GFlops.

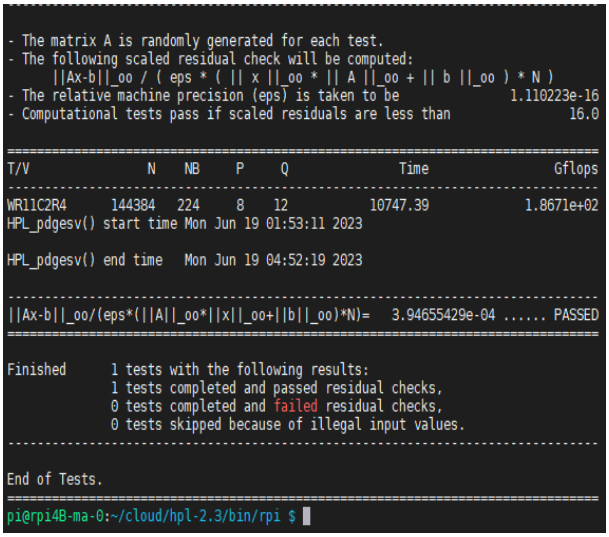

Figure 13: Highest measured HPL performance with (NVMe) for the whole cluster (≈ 90% System Memory Utilization).

4.3 Results and Conclusions

The summation of the results of this research per testing category is analyzed below:

- *Multithreaded C++.* – A multicore CPU allows multiple processes to execute simultaneously and as result the primary goal of creating multi-threaded programs is to decrease the time of a program's execution. "Figure 5", "Figure 6", "Figure 7", presents the results of the *Multithreaded C++* program

executed in a CPU (1 RPi 4B) with 4 cores. Whenever we add more cores in the program execution the performance gets higher reducing the execution time. The speedup time (performance) by using four cores of the CPU reaches three times better contrary to use only one core.

- *Multithreaded C++ with MPI in one CPU.* – Similar results and slightly better are depicted when we run a C++ program with the use of MPI. "Figure 8", "Figure 9" presents an example of a C++ code execution time with the use of MPI and the CPU cores from one to four. The speedup time (performance) by using four cores of the CPU using the MPI routines in the C++ program reaches almost four times better results contrary to use only one core (process or thread).

- *Multithreaded C++ with MPI in four CPUs (4 RPi's).* – "Figure 10", "Figure 11" shows the results of the *Multithreaded C++ with MPI in four CPUs (4 RPi's*) with remarkable results. One CPU (4 cores-4 processes) takes approximately 11 sec to execute the C++ program with MPI and 3.2 second when four RPi's (4 CPUs – 16 cores) are involved in the parallel program execution. Obviously, whenever more CPU cores are involved, there is a decent performance improvement.

- *HPL Benchmark with microSD vs NVMe SSD disks*. –

"Table 3" presents the scaling efficiency in (time) when the NVMe SSD disk is used in a Raspberry Pi 4B when HPL benchmarking testing is applied contrary to the condition when microSD is used as a bootable disk. Taking into account that the external SSDs (NVMe) are connected to USB3.0, where the theoretical transfer speed of USB 3.0 is 4.8 Gbit/s (600MB/s) the increased performance is more than decent compared to microSD write-read data throughput capabilities [24].

Table 3. Beowulf cluster scaling efficiency in terms of (time) between microSD disk vs NVMe disk

microSD disk vs NVMe disk usage Efficiency in (time)			
Nodes	**Time (microSD) (sec)**	**Time (NVMe) (sec)**	**Scaling Efficiency in time (%)**
1	1883.03	1714.29	100
3	3980.47	3372.66	15.27
6	5720.26	4789.13	16.28
9	7043.71	6024.62	14.47
12	8276.69	7077.49	14.49
15	9347.27	8007.92	14.33
18	10329.64	8856.18	14.26
21	11935.96	9760.99	18.22
24	12491.43	10747.39	13.96

The theoretical transfer speed of USB 3.0 is approximately 4.8 Gbit/s (600MBps) vs. USB 2.0 which is approximately 480 Mbit/s (60MBps) meaning more or less tenfold improvement. On the other side, sustained transfer speeds in real life for external hard drives are about 85MBps for USB 3.0 and about 22MBps for USB 2.0, so about a fivefold improvement but still a significant advancement in transfer speed.

From that perspective, -despite the use of the state of the arts NVMe SSDs – the HPL computing performance using the USB 3.0 is more than decent. The scaling efficiency in (time) is around (14-16 %) better performance in saving time in calculation.

Table 4. Beowulf cluster scaling efficiency in terms of (GFlops) between microSD disk vs NVMe disk

microSD disk vs NVMe disk usage Efficiency in (GFlops)			
Nodes	**GFlops (microSD)**	**GFlops (NVMe)**	**Scaling Efficiency in GFlops (%)**
1	9.03	9.96	100
3	22.48	26.53	18.02
6	44.08	52.65	19.44
9	65.49	76.56	16.90
12	85.84	100.39	16.95
15	106.16	123.91	16.72
18	126.61	147.68	16.64
21	137.94	168.67	22.28
24	160.64	186.71	16.23

On the other side, "Table 4" presents the scaling efficiency in (GFlops) -when the microSD disk is used- in a Raspberry Pi 4B when HPL benchmarking testing is applied contrary to the condition when NVMe SSD is used as a bootable disk. In this case, there is a decent performance improvement of about (16-22 %) in terms of GFlops.

"Figure 14", presents in a chart the HPL performance of the Beowulf cluster when the microSD is used. The scaling efficiency of the cluster starts with a 148% with 3 nodes (compared to one RPi) till 1678% with the use of 24 RPi's in the whole cluster "Table 1".

Moreover, "Figure 15", presents a chart with the HPL performance of the Beowulf cluster when the NVMe SSD disk is used. The scaling efficiency of the cluster starts with a 166% with 3 nodes (compared to one RPi) till 1774% with the use of 24 RPi's in the whole cluster "Table 2".

In overall, the increased performance in (time) and (GFlops) with the use of the NVMe SSD disks are good enough taking into account that the external SSDs are connected in the USB 3.0 of the raspberry Pi's and it is as expected based on USB 3.0 throughput capabilities.

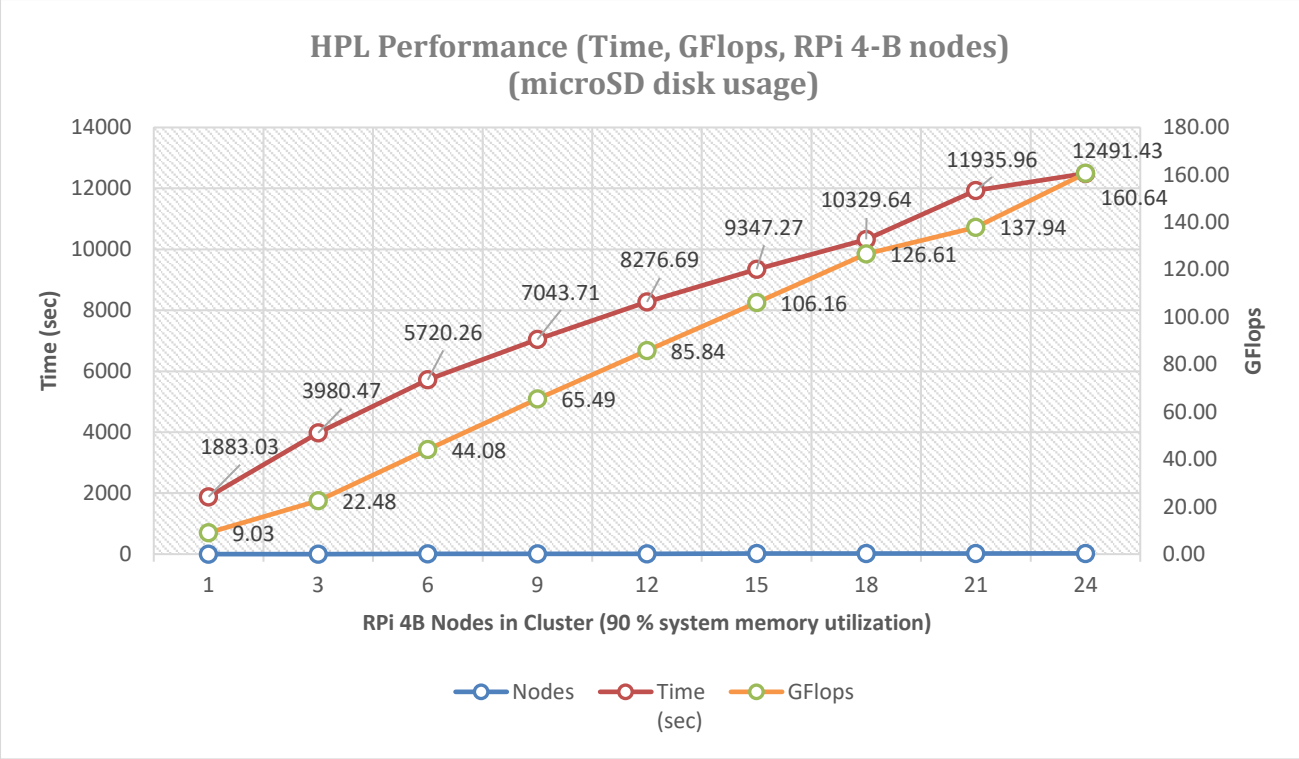

Figure 14: HPL Beowulf cluster performance results in GFlops with (≈ 90%) System Memory Utilization (microSD usage)

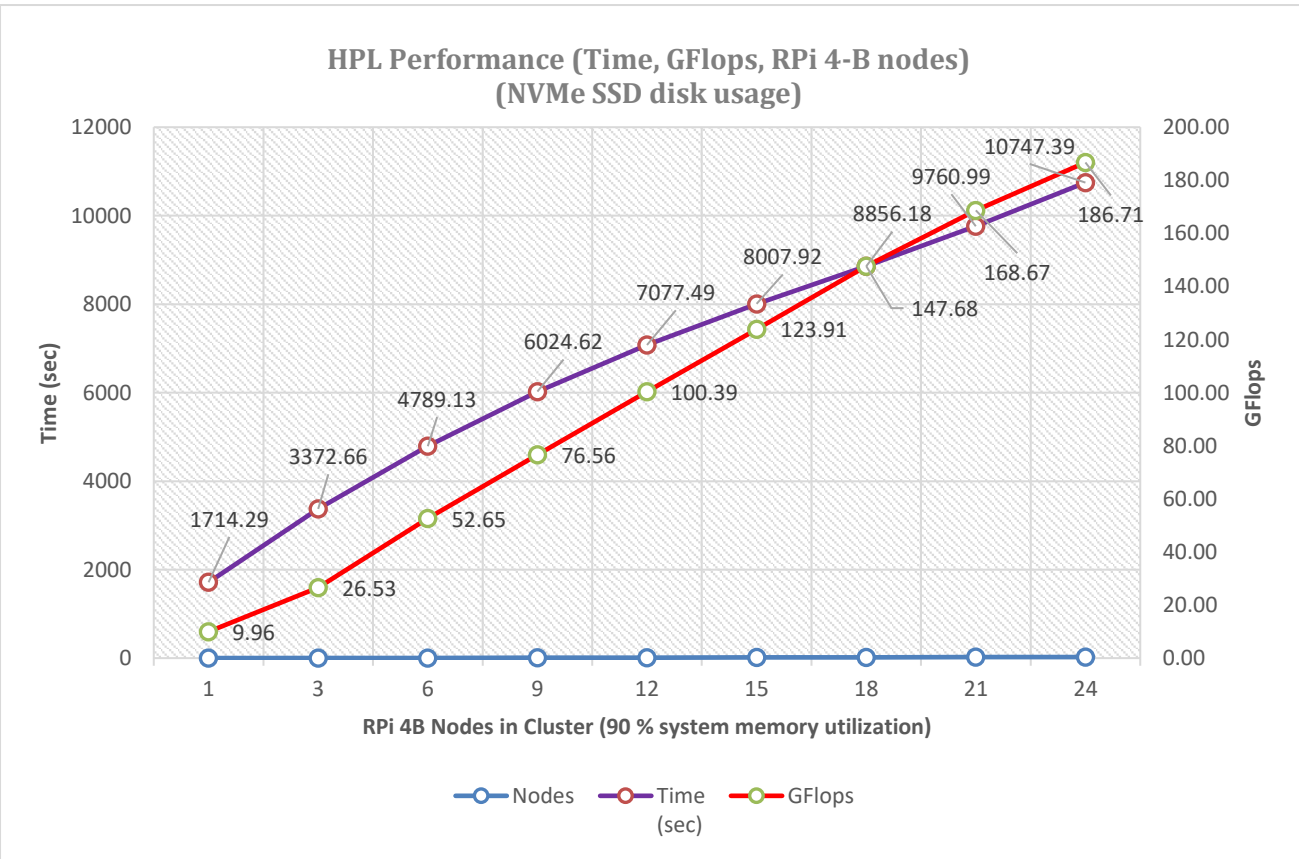

Figure 15: HPL Beowulf cluster performance results in GFlops with (≈ 90%) System Memory Utilization (NVMe SSD usage)

5. FUTURE WORK

The RPi 4B (8GB) ram introduced in overall very good performance results in the Beowulf cluster and it is intended by the authors to be used in different cluster architectures, such as, Hadoop, Spark and Kubernetes to evaluate the cluster performance. Moreover, there is a planning to run data mining algorithms and tested in such a clusters architecture to evaluate the performance when applying Big Data Analytics with Data Mining Algorithms in High Performance Computing systems.

6. ACKNOWLEDGMENTS

My sincere gratitude to Assistance Professor Ioannis S. Barbounakis for the precious guidelines, knowledge and contribution for the completion of this research.

In addition, a sincere appreciation to NETTOP company in Greece which is an Approved Reseller of Raspberry Pi's and donate some RPi's and supported this project to complete the Beowulf cluster deployment for the shake of the science [25].

7. REFERENCES

[1] Beowulf Computer Cluster. [Online]. Available: https://www.spacefoundation.org/space_technology_hal/beowulf-computing-cluster/.

[2] Sterling T. 2001. Beowulf clusters computing with Linux. Cambridge, Massachusetts: MIT Press.

[3] MPI. MPI Forum. [Online]. Available: http://mpi-forum.org/

[4] MPI. MPICH. [Online]. Available: https://www.mpich.org/

[5] Five Trends Shaping Next-gen, Data-intensive Supercomputing. [Online]. Available: https://www.huawei.com/en/huaweitech/publication/202202/data-intensive-supercomputing.

[6] An Analysis of System Balance and Architectural Trends Based on Top500 Supercomputers. [Online]. Available: https://dl.acm.org/doi/10.1145/3432261.3432263.

[7] High-Performance Linpack (HPL) benchmarking on UL HPC platform. [Online]. Available: https://ulhpc-tutorials.readthedocs.io/en/latest/parallel/mpi/HPL/

[8] Open Basic Linear Algebra Subprograms (OpenBLAS). [Online]. Available: http://sporadic.stanford.edu/reference/spkg/openblas.html.

[9] Raspberry Pi 4 Model B. [Online]. Available: raspberrypi.com/products/raspberry-pi-4-model-b/.

[10] Dimitrios Papakyriakou and Ioannis S Barbounakis. Benchmarking and Review of Raspberry Pi (RPi) 2B vs RPi 3B vs RPi 3B+ vs RPi 4B (8GB). International Journal of Computer Applications 185(3):37-52, April 2023.

[11] Raspberry Pi Operating System images. [Online]. Available: https://www.raspberrypi.com/software/operating-systems/#raspberry-pi-os-64-bit

[12] Message Passing Interface Chameleon MPICH. [Online]. Available: https://www.mpich.org/

[13] Netlib. HPL. [Online]. Available: http://www.netlib.org/benchmark/hpl/

[14] LU factorization. [Online]. Available: https://www.geeksforgeeks.org/l-u-decomposition-system-linear-equations/

[15] Mathematics. LU Decomposition of a System of Linear Equations. [Online]. Available: https://www.geeksforgeeks.org/l-u-decomposition-system-linear-equations/

[16] Dimitrios Papakyriakou, Dimitra Kottou and Ioannis Kostouros. (April 2018). "Benchmarking Raspberry Pi 2 Beowulf Cluster. International Journal of Computer Applications" 179(32):21-27.

[17] Petitet, A., R. C. Whaley, J. Dongarra, and A. Cleary. "HPL – A portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers." Accessed December 15, 2016

[18] Dunlop, D., Varrette, S. and Bouvry, P. 2010. Deskilling HPL, Vol. 6068 of Lecture Notes in Computer Science, Springer, Heidelberg, Berlin, 102–114.

[19] Luszczek, P., Dongarra, J., Koester, D., Rabenseifner, R., Lucas, B., Kepner, J., McCalpin, J., Bailey, D. and Takahashi, D. 2005. Introduction to the HPC Challenge Benchmark Suite, Technical Report, ICL, University of Tennessee at Knoxville.

[20] Netlib. HPL Tuning. http://www.netlib.org/benchmark/hpl/tuning.html#tips

[21] Dunlop, D., Varrette, S. and Bouvry, P. 2008. On the use of a genetic algorithm in high performance computer benchmark tuning, Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems, SPECTS 2008, Art. No.:4667550, 105-113

[22] Mathieu GAILLARD. (August 2022) How to compile HLP LINPACK on Ubuntu 22.04. [Online]. Available: https://www.mgaillard.fr/2022/08/27/benchmark-with-hpl.html

[23] HPL Frequently Asked Questions. [Online]. Available: http://www.netlib.org/benchmark/hpl/faqs.html

[24] Sindi, M. 2009. HowTo – High Performance Linpack (HPL), Technical Report, Center for Research Computing, University of Notre Dame

[24] Dimitrios Papakyriakou and Ioannis S Barbounakis. Benchmarking and Review of Raspberry Pi (RPi) 2B vs RPi 3B vs RPi 3B+ vs RPi 4B (8GB). International Journal of Computer Applications 185(3):37-52, April 2023.

[25] NETTOP. Approved and Official Raspberry Pi Reseller in Greece. [Online]. Available: https://nettop.gr/