Performance Analysis of Raspberry Pi 4B (8GB) Beowulf Cluster: HPCG Benchmarking

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

The High-Performance Conjugate Gradient (HPCG) benchmark has emerged as a complementary metric to the High Performance LINPACK (HPL) [1], aiming to evaluate real-world high-performance computing (HPC) workloads that emphasize memory access patterns, cache behavior, and sparse matrix operations. Unlike HPL, which reflects peak floating-point capability, HPCG simulates practical scientific computations involving iterative solvers and irregular memory access, offering a more realistic performance indicator.

This study investigates the implementation and analysis of the HPCG benchmark on a 24-node Beowulf cluster built with Raspberry Pi 4B devices, each equipped with 8GB LPDDR4 RAM and ARM Cortex-A72 processors. Both strong scaling (fixed problem size with increasing nodes) and weak scaling (proportional increase in problem size and nodes) methodologies were applied to assess system performance across various configurations. Metrics such as median execution time, floating-point throughput (GFLOP/s), and memory bandwidth (GB/s) were collected and analyzed.

The results reveal that HPCG performance on this ARM-based cluster is primarily constrained by memory bandwidth saturation, lack of hardware-level floating-point acceleration, and network communication bottlenecks. Strong scaling experiments show minimal performance gains beyond 4–8 nodes, while weak scaling maintains computational stability up to moderate cluster sizes. Notably, the absence of measurable MPI communication overhead (ExchangeHalo time) underscores the limited halo data exchange under small subdomain decomposition and short runtimes.

This study highlights the limitations and potential of energy-efficient, low-cost single-board clusters for realistic HPC workloads. The findings provide a methodological basis for benchmarking sparse solvers on ARM systems and inform future efforts in optimizing parallelism, memory access, and interconnect efficiency in edge computing, education, and embedded HPC environments.

Keywords

Raspberry Pi 4 Beowulf cluster, Cluster, Message Passing Interface (MPI), MPICH, Memory Performance, Low-cost Clusters, Parallel Computing, ARM Architecture, HPCG Benchmark.

1. INTRODUCTION

The High-Performance Conjugate Gradient (HPCG) Benchmark has been developed as a complementary metric to the High-Performance LINPACK (HPL) Benchmark, addressing the limitations of peak floating-point performance assessments in modern high-performance computing (HPC) systems [1]. While HPL remains the standard for ranking supercomputers in the TOP500 list, it primarily measures dense matrix computation performance, which does not accurately reflect the efficiency of many real-world applications. In contrast, HPCG is designed to evaluate computational performance under memory-bound conditions, incorporating sparse matrix operations, iterative solvers, memory hierarchy efficiency, and interconnect performance.

HPCG is particularly relevant for HPC systems engaged in scientific computing, computational fluid dynamics, finite element analysis, and machine learning, where workloads are dominated by sparse linear algebra computations. These workloads exhibit irregular memory access patterns, high communication overhead, and limited floating-point operations density, making them highly dependent on memory bandwidth, cache utilization, and efficient inter-node communication. As modern supercomputers increasingly rely on heterogeneous architectures, including low-power processors and distributed computing paradigms, understanding the behavior of sparse matrix solvers within a given hardware environment is crucial for optimizing performance.

The evaluation of high-performance computing (HPC) systems requires multiple benchmarking approaches to assess different aspects of computational efficiency. The High-Performance Conjugate Gradient (HPCG) Benchmark, the High-Performance LINPACK (HPL) Benchmark [1] and the STREAM Benchmark [2], each focus on distinct performance metrics, making them complementary tools for understanding the capabilities and limitations of modern HPC architectures.

The HPCG Benchmark is designed to measure the performance of sparse linear algebra operations, which are representative of real-world scientific and engineering applications. It focuses on memory access efficiency, cache utilization, and inter-process communication overhead, making it particularly relevant for HPC workloads that involve irregular memory access patterns and distributed sparse matrix computations. Unlike traditional dense matrix benchmarks, HPCG reflects the challenges faced in applications such as computational fluid dynamics, structural analysis, and machine learning.

In contrast, the STREAM Benchmark [2] is primarily used to evaluate memory bandwidth performance. It measures the sustained memory transfer rates of fundamental operations such as *Copy*, *Scale*, *Add*, and *Triad*, which are critical in memory-intensive applications. Since modern HPC systems are often limited by memory bandwidth rather than raw computational power, STREAM provides insights into how

efficiently a system can move data between main memory and processing units. This benchmark is particularly useful for understanding bandwidth bottlenecks and cache efficiency in both single-node and multi-node systems.

The HPL Benchmark (LINPACK), which has historically been the standard for ranking supercomputers in the TOP500 list, is optimized for solving dense linear algebra problems. It evaluates peak floating-point performance (FLOP/s) by solving large systems of linear equations using LU decomposition. HPL emphasizes compute-bound performance, making it an effective measure of a system's raw processing power. However, it does not reflect the performance of real-world workloads that involve sparse matrices, communication overhead, or memory bandwidth limitations.

A comparative analysis of these benchmarks provides a more comprehensive understanding of an HPC system's strengths and weaknesses. While HPL excels in measuring theoretical peak performance, HPCG provides a more accurate representation of practical workloads, and STREAM highlights memory bandwidth efficiency, which often becomes the limiting factor in large-scale computations. The combined use of these benchmarks is essential for optimizing system performance, identifying bottlenecks, and designing more efficient computing architectures "Table 1".

This study focuses on deploying and analyzing the HPCG benchmark on a Beowulf cluster composed of 24 Raspberry Pi 4B nodes (8GB RAM each), interconnected via Ethernet, to explore the performance constraints and scalability of ARM-based distributed systems. The evaluation investigates MPI-based parallelization, memory bottlenecks, floating-point throughput, and communication overhead, offering insights into the feasibility of utilizing energy-efficient ARM clusters for scientific and engineering applications. By benchmarking the cluster under varying computational workloads and process configurations, this research aims to provide a deeper understanding of the real-world computational efficiency of low-power, cost-effective HPC solutions.

The Raspberry Pi (RPi) 4 Model B (8GB RAM), depicted in "Figure 1", serves as the foundation of the Beowulf cluster. Equipped with a 64-bit quad-core ARMv8 Cortex-A72 CPU clocked at 1.5 GHz, it delivers three times the processing power of its predecessor, the RPi 3B+ [3], [4], [5]. The cost-effectiveness of the Raspberry Pi played a crucial role in selecting it as a viable solution for constructing a high-performance computing (HPC) cluster, enabling an in-depth evaluation of its efficiency in parallel computing and clustering environments

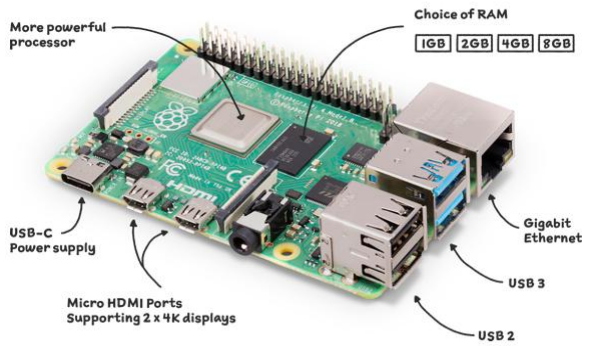

Figure 1: Single Board Computer (SBC) - Raspberry Pi 4 Model B [4], [5].

2. SYSTEM DESCRIPTION

2.1 Hardware Equipment

The Beowulf cluster comprises 24 Raspberry Pi 4B (8GB) devices, as illustrated in "Figure 2". A single Raspberry Pi 4B serves as the master (head) node, responsible for job scheduling and resource management, while the remaining 23 Raspberry Pi's function as worker nodes, executing computational tasks under the master's coordination.

The nodes are structured into four stacks, each containing six Raspberry Pi's, and are interconnected via Gigabit Ethernet switches (TL-SG1024D), providing a maximum network bandwidth of 1000 Mbps per node. This network topology enables seamless communication between nodes, effectively simulating a high-performance computing (HPC) environment similar to that of a supercomputer "Figure 2".

The cluster is powered by two switch-mode power supplies, each rated at 60 amps with a 5V output, which is boosted to 5.80V to compensate for potential voltage drops along the wiring. Additionally, the master node is equipped with a Samsung 980 (1TB) PCI-E 3 NVMe M.2 SSD, while each worker node is fitted with a 256GB Patriot P300P256GM28 NVMe M.2 2280 SSD, ensuring high-speed storage access and data handling across the system.

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 latest "Debian GNU/Linux 12 (bookworm)" which is the latest official supported Operating System (OS - 64 bits) with Kernel version 6.6.62+rpt-rpi-v8 and the CPU architecture and capabilities of the system.

The second essential software component for the cluster setup was the Message Passing Interface (MPI), with MPICH chosen as the specific implementation. MPICH is a highly efficient and widely adaptable MPI framework, which serves as a fundamental standard for message-passing in parallel computing. It is important to clarify that MPI itself is not a library, but rather a standardized framework defined by the MPI Forum for the development of message-passing libraries.

Several MPI implementations are available for use on Raspberry Pi, including OpenMPI and MPICH. For this project, MPICH was selected due to its conformance to the MPI standard and its broad compatibility with applications written in C, C++, and FORTRAN. Originally standing for Message Passing Interface Chameleon, MPICH is designed to support

high-performance distributed computing, making it well-suited for the Beowulf cluster environment.

The third essential software package installed was the GNU Compiler Collection (GCC) Fortran compiler, which is widely used in high-performance computing (HPC) due to its optimization capabilities and multi-threading support. As the default compiler in many HPC environments, GCC plays a critical role in compiling and optimizing parallel computing applications.

The fourth key software component was OpenBLAS, a highly optimized implementation of Basic Linear Algebra Subprograms (BLAS). OpenBLAS provides efficient and accelerated linear algebra operations, which are fundamental to numerous scientific and engineering computations.

Finally, the HPCG benchmark software was required to be downloaded, compiled, and configured appropriately to evaluate the computational and memory performance of the Beowulf cluster.

2.3 Design

The architecture of the Raspberry Pi (RPi) cluster is illustrated in "Figure 3", comprising 24 Raspberry Pi 4B nodes, each equipped with 8GB of RAM [4], [5]. These nodes are interconnected via a 24-port Gigabit Ethernet switch (1000 Mbps) to facilitate high-speed data exchange. Within this configuration, one Raspberry Pi functions as the master (head) node, responsible for task scheduling and resource management, while the remaining 23 nodes operate as worker nodes, executing computational workloads. To ensure efficient network communication, static IP addressing is implemented, assigning each node a unique and fixed IP address. Communication between the master and worker nodes is conducted securely through SSH (Secure Shell) connections.

The master node is equipped with a Samsung 980 PCIe 3.0 NVMe M.2 SSD (1TB), capable of theoretical maximum write speeds of 3000 MB/s and read speeds of 3500 MB/s. To enhance storage performance across the cluster, each worker node is outfitted with a Patriot P300P256GM28 NVMe M.2 SSD (256GB), offering maximum write speeds of 1100 MB/s and read speeds of 1700 MB/s. Since the Raspberry Pi 4B supports external booting, these SSDs are connected via USB 3.0 ports, which provide a theoretical data transfer rate of 4.8 Gbps (600 MB/s)—a significant improvement over USB 2.0, which is limited to 480 Mbps (60 MB/s). This storage configuration enhances the I/O performance of the cluster, enabling faster data access and improved computational efficiency.

By utilizing the high-speed read and write capabilities of the NVMe SSDs, this phase of testing aimed to achieve significant performance improvements compared to the previous microSD-based configuration. Although the USB 3.0 interface introduces some bandwidth limitations, the superior speed and efficiency of the NVMe SSDs greatly surpass these constraints, resulting in a notable enhancement in overall cluster performance.

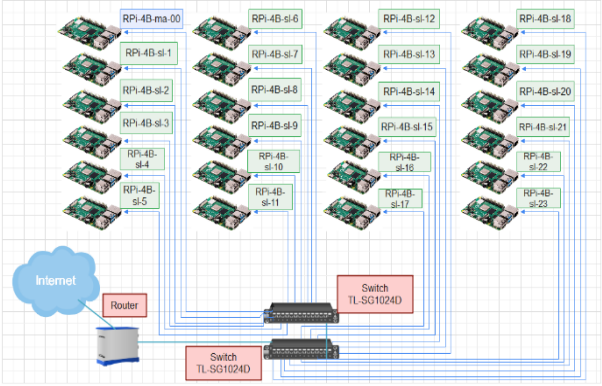

Figure 3: RPi-4B Beowulf cluster architecture diagram [1], [2].

3. HPCG

The High-Performance Conjugate Gradient (HPCG) Benchmark is a widely recognized tool for assessing the real-world performance of high-performance computing (HPC) systems [6]. Unlike the High-Performance LINPACK (HPL) Benchmark, which measures peak floating-point performance, HPCG focuses on memory bandwidth, cache utilization, and inter-node communication efficiency by solving a sparse system of linear equations using the Conjugate Gradient (CG) method. This makes it particularly relevant for scientific and engineering applications that involve iterative solvers, irregular memory access patterns, and distributed computing workloads.

HPCG evaluates system performance by solving a sparse linear system using the Conjugate Gradient (CG) method with a multi-grid preconditioner [7], [8]. This approach highlights:

- *Memory Bandwidth (GB/s):* Performance depends on efficient memory access.
- *Cache Utilization (L1/L2 misses):* Frequent cache misses slow computation.
- *Interconnect Performance (MPI overhead and communication efficiency):* For multi-node systems, communication is a key bottleneck.
- *Floating-Point Performance (GFLOP/s)*: Though not as dominant as in LINPACK.

Raspberry Pi-based Beowulf cluster has significant memory bandwidth constraints (as seen in STREAM results) [2]. Since HPCG is highly memory and communication bound, it will provide deeper insights into:

- *Memory bandwidth limitations and cache efficiency.*
- *MPI communication overhead between nodes.*
- *How well the cluster scales for real-world workloads.*

Unlike STREAM, which focuses purely on memory bandwidth, HPCG integrates memory, computation, and communication aspects, making it a more comprehensive performance metric.

The HPCG benchmark consists of:

- *Sparse Matrix-Vector Multiplication (SpMV):*
  Dominates computational cost and memory bandwidth usage. Uses a structured grid problem to simulate scientific computing loads.
- *Symmetric Gauss-Seidel Preconditioning:*
  Helps accelerate convergence of the Conjugate Gradient method.
  Requires non-trivial memory accesses.
- *Global Dot Products (MPI Reduction Operations)*

Synchronization-heavy operations that expose network latency in multi-node systems.

- *Multi-Grid Preconditioner:*
  A coarse-grid correction mechanism to improve solver efficiency.

To evaluate the performance of the Beowulf cluster, it is first necessary to analyze the behavior of HPCG on a single Raspberry Pi 4B (8GB RAM). This initial testing phase provides a baseline measurement of computational efficiency, memory throughput, and HPCG performance, which will later be compared to multi-node cluster performance.

3.1 HPCG Methodology.

First of all, it was observed that the Grid Sizes feasible to run in a Raspberry Pi are the following:

- $16 \times 16 \times 16$ *Grid Size* success
- $32 \times 32 \times 32$ *Grid Size* success
- $64 \times 64 \times 64$ *Grid Size* success
- $96 \times 96 \times 96$ *Grid Size* success
- $128 \times 128 \times 128$ *Grid Size* success
- $136 \times 136 \times 136$ *Grid Size* failed
- $144 \times 144 \times 144$ *Grid Size* failed

The primary reason why the RPi failed to operate with more complex Grid Size is that, HPCG operates on a sparse matrix representation but still requires a large amount of memory, for Matrix storage, Vectors, Multi-grid hierarchy and temporary buffers for Sparse Matrix-Vector Multiplication (SpMV).

The $128 \times 128 \times 128$ *Grid Size* problem already consumed 6.95GB out of 7.64 GB available RAM. Apparently, the larger Grid Sizes exceeded the 8GB physical RAM memory of RPi 4B and caused Out of memory (OOM) errors. In that case we have the option to force the system to use *Swap Memory* but this drastically slows down the computation. The recommendation in this study is to use only the physical RAM Memory (nominal 8GB, available 7.64GB), to disable the Swap Memory (by using the command *$ sudo swapoff -a*) not allowing the system to use disk space as extra RAM, for scientific accuracy reasons and preventing slow I/O operation.

During the execution of the tests, the command (*watch -n 1 free -h*) in a Command Line Interface (CLI) verifies that no Swap Memory is used. Taking into account that the Raspberry Pi 4B uses LPDDR4-3200 RAM, but the memory bandwidth is only (~4GB/s), the Grid Size of 128^3 already stresses the RAM Memory.

The HPCG methodology for one RPi is based on observing the RPi performance by increasing the Grid size from $16 \times 16 \times 16$ to $128 \times 128 \times 128$. The following Key Performance Indicators (KPIs) are considered:

- *Grid Size:* It's the problem size used in the HPCG test, defined by the number of grid points in each dimension and determines computational complexity, memory usage, and scalability. Larger grids test system performance under heavier workloads.

- *Total Benchmarking Median Execution Time (sec):* It's the median total time required to complete the HPCG benchmark and it is a direct measure of computational efficiency. A lower execution time indicates better performance.

- *CG Solver Efficiency (Iteration Count):* It is the number of iterations required by the Conjugate Gradient (CG) solver to converge. Fewer iterations indicate better solver efficiency, meaning the system is solving the problem more effectively.

- *Memory Usage – Median Total Memory Used (Mbytes)*: It is the median amount of memory allocated and used for data storage during execution. It helps assess if the system is memory-bound. Higher memory usage may lead to performance degradation due to cache/memory bottlenecks

- *Memory Usage-Median Bytes per Equation:* The amount of memory used per equation solved by the system. Indicates computational efficiency concerning memory. Large values suggest high memory overhead.

- *Median SpMV Time (sec) (Sparse Matrix-Vector Multiplication):* The median execution time for the SpMV operation constitutes a fundamental kernel in HPCG. SpMV is memory-bound, meaning its efficiency depends on memory bandwidth and cache performance.

- *Median MG Time (sec) (Multi-Grid Preconditioner):* It is the median execution time for the Multi-Grid (MG) preconditioner, which accelerates the CG solver. MG performance is also memory-bound, but it can also indicate how well cache and memory hierarchies are utilized

- *Median Optimization Phase Time (sec):* It is the time spent in the system's performance tuning phase during the HPCG benchmark. Shorter optimization phase times indicate a well-optimized setup, while longer times suggest inefficiencies in computational tuning.

- *Median Departure from Symmetry for SpMV (numerical Stability):* Measures how much the SpMV operation deviates from expected symmetry, which impacts numerical accuracy. A higher departure indicates floating-point instability, which can lead to inaccurate results in large-scale computations

- *Median Departure from Symmetry for MG (Numerical Stability:* Measures symmetry deviation for the MG preconditioner. Similar to SpMV, a large departure suggests numerical instability, which can affect solver accuracy. Comparing execution times across different grid sizes and node counts helps identify bottlenecks and assists to Scalability Analysis.

As an overall importance of the above metrics, the *execution time*, *SpMV time*, and *MG time* indicate computational efficiency and performance evaluation.

In terms of the whole Raspberry Pi Cluster methodology in this study the HPCG Benchmarking for Strong and Weak Scaling is used to evaluate the scalability and performance.

- *Strong Scaling Experiment:*
  The strong scaling tests were executed by maintaining a fixed grid size of 128 × 128 × 128 (the maximum supported by the RPi) while varying the number of Raspberry Pi nodes in the cluster.

  Gradually increasing the number of nodes (1, 2, 4, 8, 12, 16, 24) -where a fixed grid size of $128 \times 128 \times 128$ *Grid Size* is used - the execution time, GFLOP/s, memory bandwidth, and MPI communication overhead are extracted for comparison.

This setup allows us to measure how well the cluster parallelizes the fixed computational workload. If execution time decreases proportionally to added nodes, the system scales efficiently. However, deviations from ideal scaling may indicate communication overhead or memory bandwidth limitations.

- *Weak Scaling Experiment:*
  The weak scaling tests assess how the cluster handles increasing workloads while keeping the per-node workload constant. The test starts with one node solving a small $16 \times 16 \times 16$ *Grid Size*. As the number of nodes increases, the problem size increases proportionally such as:
  1 node with $16 \times 16 \times 16$ *Grid Size*
  2 nodes with $32 \times 32 \times 32$ *Grid Size*
  4 nodes with $64 \times 64 \times 64$ *Grid Size*
  8 nodes with $96 \times 96 \times 96$ *Grid Size*
  12 nodes with $128 \times 128 \times 128$ *Grid Size*

  The above Standard Weak scaling approach ensures that each node gets exactly the same computational load in every step. The grid size doubles every time nodes double, maintaining constant work per node and the MPI communication overhead is easier to analyze when workloads remain balanced.

  In weak scaling, each node should process the same amount of data as more nodes are added. Key performance metrics include execution time per node, efficiency, and floating-point performance (GFLOP/s). Weak scaling is ideal for measuring network efficiency; if performance remains stable as more nodes are added, the cluster scales well. However, significant drops in efficiency may indicate network bottlenecks or memory bandwidth limitations.

  For One RPi the Scaling experiment is based on increasing Grid Size from $16 \times 16 \times 16$ *Grid Size* to $128 \times 128 \times 128$ *Grid Size* "Figure 4", "Figure 5", "Table 2".

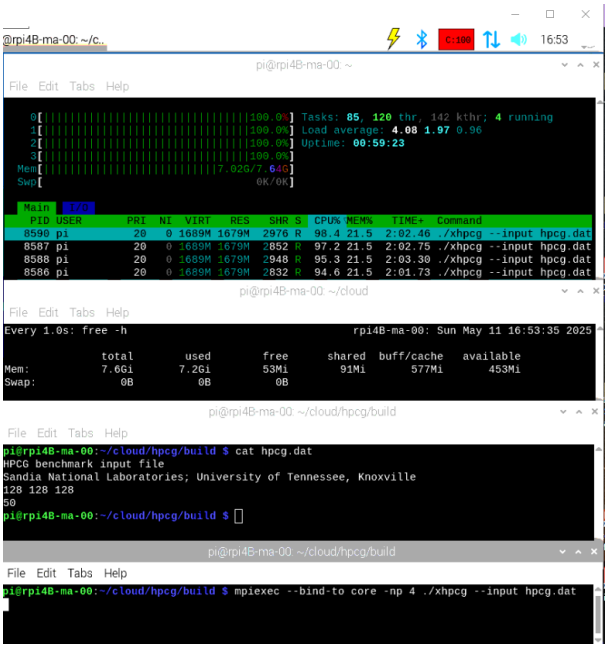

Figure 4: HPCG Benchmark in one RPi with Grid Size 128^3 with no swap memory used

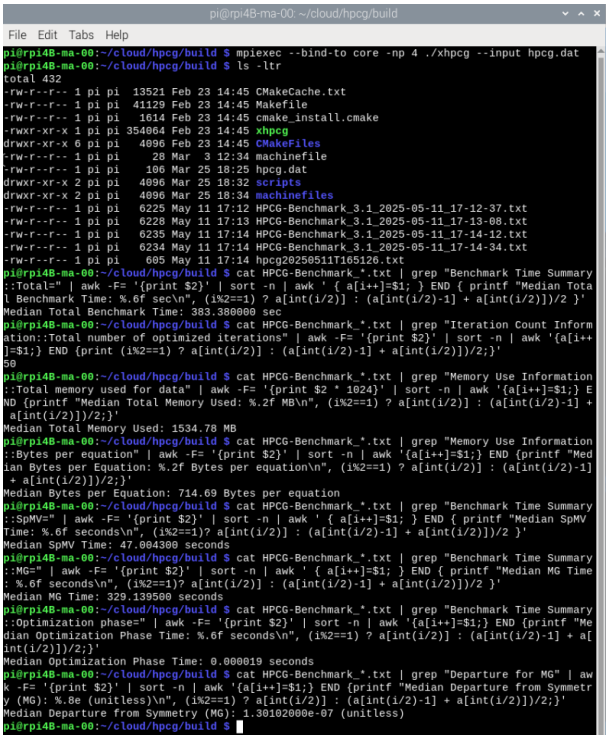

Figure 5: HPCG Benchmark results in one RPi with Grid Size 128^3

3.2 HPCG Performance in one RPi.

Before scaling the HPCG benchmark to the full Beowulf cluster, it is essential to first evaluate its performance on a single Raspberry Pi 4B (8GB RAM). This initial testing phase serves multiple purposes:

- *Establishing a Baseline Performance Measurement*
  Running HPCG on a single node provides a reference point for key performance metrics such as memory bandwidth, floating-point throughput, and solver efficiency. This baseline enables a direct comparison with the multi-node cluster results, helping to quantify the benefits and overhead of parallel execution.

- *Understanding Computational Bottlenecks*
  Testing on a single Raspberry Pi identifies the primary performance constraints related to CPU, memory, and storage before introducing inter-node communication overhead. If a single node is heavily memory-bound or suffers from inefficient computation, similar issues will likely scale across the cluster.

- *Verifying Software Configuration and Optimization*
  Ensuring that HPCG is correctly compiled and configured for the Raspberry Pi's ARMv8 Cortex-A72 architecture is crucial before deploying it across multiple nodes. Optimizing compiler flags, MPI settings, and BLAS library integration at the single-node level can improve efficiency when scaling to a cluster.

- *Assessing MPI and Parallel Execution Within a Single Node*
  Since the Raspberry Pi 4B features a quad-core processor, running HPCG with multiple MPI processes within a single node allows for an initial evaluation of parallel performance. This helps determine whether the system benefits from multi-threading or suffers from cache contention and memory bandwidth limitations.

- *Minimizing Debugging Complexity Before Scaling*
  Testing on a single node allows for easier troubleshooting of potential issues with HPCG execution, memory usage, or solver convergence. Identifying and resolving problems at this stage reduces debugging complexity when extending the benchmark to a multi-node Beowulf cluster.

By first analyzing HPCG performance on a single Raspberry Pi, a more informed and optimized scaling strategy can be developed for the full cluster deployment. This approach ensures that computational, memory, and communication bottlenecks are properly addressed, maximizing the efficiency of parallel execution across multiple nodes.

HPCG evaluates system performance by solving a sparse linear system using the Conjugate Gradient (CG) method with a multi-grid preconditioner. This approach highlights:

- *Memory Bandwidth:* performance depends on efficient memory access
- *Cache Utilization*: frequent cache misses slow computation
- *Interconnect Performance*: for multi-node systems, communication is a key bottleneck
- *Floating-Point Performance*: though not as dominant as in LINPACK.

The Observations and analysis of the results are the following based on "Table 2", "Figure 6", "Figure 7".

In terms of execution time and scalability, it is observed that as the grid size increases, the total execution time rises significantly. Larger grids increase memory requirements, leading to higher cache miss rates. Sparse matrix computations in HPCG are memory-bound, meaning performance is limited by memory bandwidth rather than raw CPU power. The results at $96 \times 96 \times 96\ Grid\ Size$ and $128 \times 128 \times 128\ Grid\ Size$ suggest that memory bandwidth is a key bottleneck.

The number of Conjugate Gradient (CG) solver iterations drops drastically as the grid size increases from $16 \times 16 \times 16\ Grid\ Size$ with 3600 iterations to $64 \times 64 \times 64\ Grid\ Size$ with 50 iterations. This is happening because smaller grids require more iterations to converge. Larger grids better approximate real-world systems, reducing the number of iterations needed. This does not mean that larger grids are more efficient since execution time still increases due to memory and computation overhead.

Regarding the memory usage, it is observed that total memory usage grows exponentially from $16 \times 16 \times 16\ Grid\ Size$ with 2.99 MB to $128 \times 128 \times 128\ Grid\ Size$ 1.54 GB. Bytes per Equation (~714 B) remains consistent, showing that each system of equations requires a stable memory footprint. The Key Limitation is that the Raspberry Pi 4B (8GB) is unable to handle problem sizes beyond 128×128×128 due to memory constraints.

With respect to Sparse Matrix-Vector Multiplication (SpMV) Performance, it is observed that Median SpMV Execution Time increases with grid size from $16 \times 16 \times 16\ Grid\ Size$ with (6.63 sec) to $128 \times 128 \times 128\ Grid\ Size$ with (49.04 sec). This happens since SpMV is memory bandwidth limited, meaning larger problem sizes cause more cache misses. The $64 \times 64 \times 64\ Grid\ Size$ achieves the lowest SpMV time, suggesting a balance between memory access efficiency and computation. Beyond $64 \times 64 \times 64\ Grid\ Size$ it is observed that the performance drops significantly, confirming that cache and memory bandwidth bottlenecks dominate.

Related to Multi-Grid (MG) Performance, MG operations are more computationally intensive than SpMV. They require more memory access, iterative refinements, and hierarchical computations, leading to significantly higher execution times. MG is the most time-consuming part of HPCG, confirming that hierarchical multi-grid solvers are heavy on memory access and cache utilization. The non-linear increase in MG time suggests that cache size limitations are a primary issue.

In terms of Numerical Stability (Departure from Symmetry) and in particular SpMV Stability and MG Stability, suggest that smaller grids exhibit larger numerical deviations due to coarse approximations. Larger grids yield better numerical stability and in general no instability issues observed, confirming the correctness of calculations.

As a general conclusion the outcome is that the Raspberry Pi 4B's LPDDR4-3200 memory bandwidth is a major bottleneck. Strong scaling on a single RPi is ineffective beyond $96 \times 96 \times 96\ Grid\ Size$, as computation time scales non-linearly due to cache inefficiencies. The ideal problem size for an RPi 4B is around $64 \times 64 \times 64\ Grid\ Size$ to $96 \times 96 \times 96\ Grid\ Size$, beyond which performance degrades due to memory constraints.

Table 1. Comparison of HPCG, HPL [1], and STREAM Benchmarking [2]

Features	HPCG Benchmark	HPL Benchmark (LINPACK)	STREAM Benchmark
Primary Focus	Memory, compute, and communication	Floating-point peak performance	Memory bandwidth
Computation	Sparse matrix operations (SpMV, Gauss-Seidel)	Dense linear algebra (LU decomposition, BLAS operations)	None (pure memory operations)
Memory Access Pattern	Irregular, indirect addressing	Regular, cache-friendly	Simple, sequential
Parallelism	MPI-based, distributed memory parallelism	MPI-based, highly parallel	Thread-based (OpenMP, MPI)
Communication Overhead	High MPI overhead in multi-node runs	High but scalable with efficient networks	None (unless using MPI)
Performance Bottlenecks	Cache inefficiency, DRAM bandwidth, network latency	Floating-point operations, memory bandwidth, network speed	DRAM bandwidth
Optimization Targets	Cache blocking, SpMV optimization, MPI tuning	BLAS optimizations, matrix decomposition tuning	Memory access efficiency, vectorization

Scalability in HPC Clusters	Sub-linear due to communication overhead	Near-linear with efficient interconnect	Linear with memory bandwidth
Real-World Relevance	Scientific computing, iterative solvers	AI, simulations, deep learning, physics simulations	Memory-intensive applications

Table 2. HPCG Benchmark results in one RPi

HPCG Benchmark to (1) RPi - - (2 MPI processes)									
Grid Size	**Total Benchmark** Median Execution Time (sec)	**CG Solver Efficiency** Iteration Count	**Memory Usage** Median Total Memory Used (Mbytes)	**Memory Usage** Median Bytes per Equation	**Median SpMV Time** (sec)	**Median MG Time** (sec)	**Median Optimization Phase Time** (sec)	**Median Departure from Symmetry for SpMV** (Numerical Stability)	**Median of Departure from Symmetry for MG** (Numerical Stability)
16x16x16	50.2798	4125	3.000	715.210	6.9600	42.1096	0.000000	6.30E-05	3.72E-06
32x32x32	53.2096	475	23.980	714.740	6.9976	44.9090	0.000001	7.10E-06	3.23E-07
64x64x64	92.761	100	191.850	714.700	11.661	79.3468	0.000000	6.27E-07	7.41E-08
96x96x96	155.265	50	647.490	714.690	19.355	131.373	0.000001	3.57E-07	1.44E-07
128x128x128	383.38	50	1534.780	714.690	47.004	329.139	0.000019	1.79E-07	1.30E-07

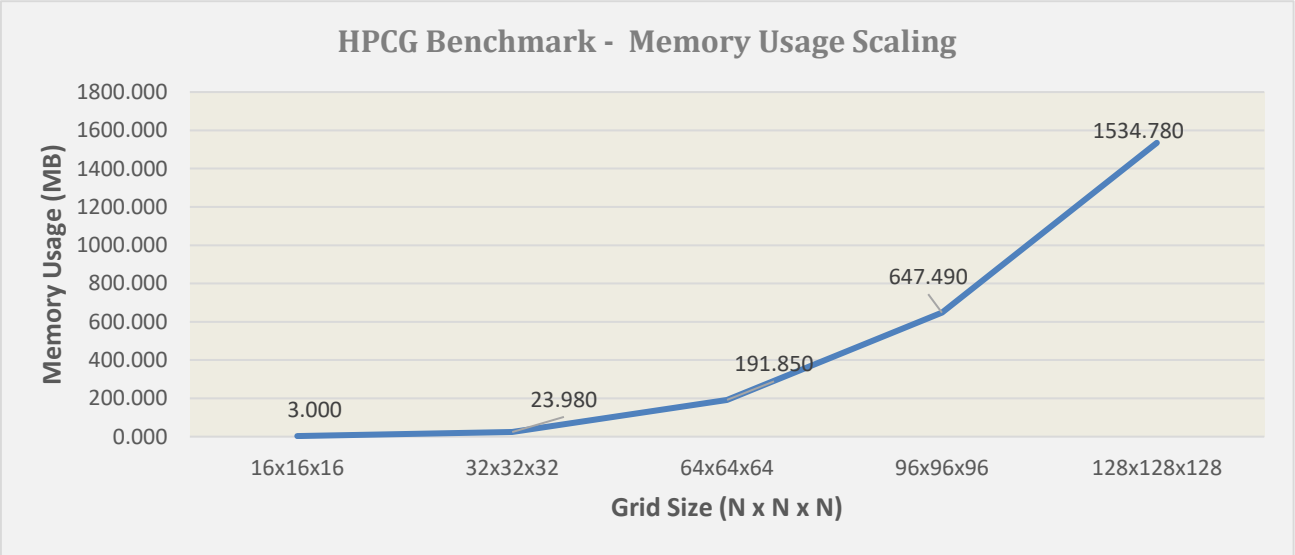

Figure 6: HPCG Benchmark: Memory Usage Scaling in (1) RPi - (2 MPI processes)

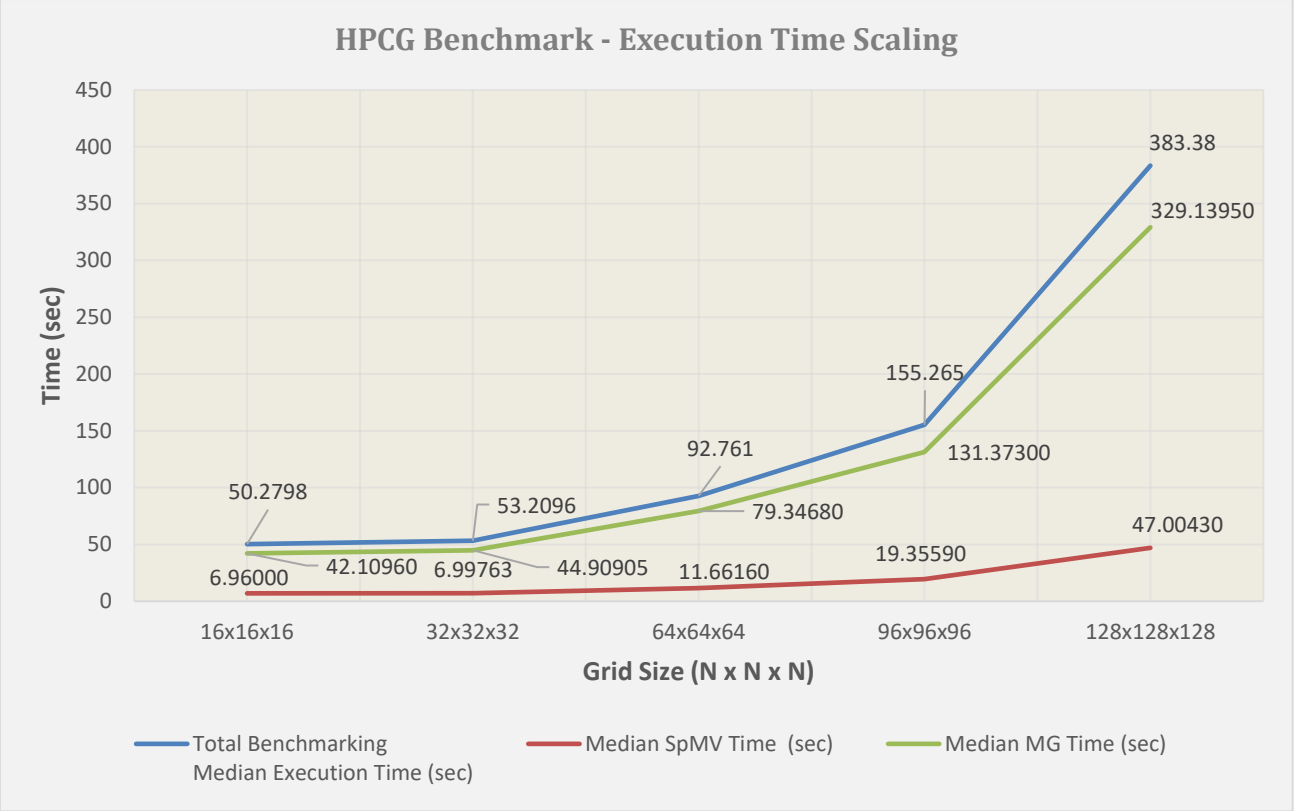

Figure 7: HPCG Benchmark: Execution Time Scaling in (1) RPi - (2 MPI processes)

3.3 HPCG Performance in the whole Beowulf Cluster

The HPCG benchmark was executed across the full Beowulf cluster using 2 MPI processes per Raspberry Pi (RPi), a configuration that balances computational load while reducing memory contention and MPI overhead. The analysis includes two classic scalability tests: *strong scaling* and *weak scaling*, each exploring distinct performance characteristics as the cluster increases in size. The choice to apply strong scaling and weak scaling methodologies in the context of the HPCG (High Performance Conjugate Gradient) benchmark is not only methodologically sound but also aligned with established practices in high-performance computing (HPC) benchmarking and performance analysis literature. These two scalability paradigms serve complementary purposes and offer distinct insights into system performance under varying computational and architectural stresses.

This dual-scaling strategy enables a holistic evaluation of the cluster's computational behaviour and communication efficiency. *Strong scaling* focuses on how performance improves when a fixed-size problem is distributed across an increasing number of nodes. This test is particularly useful for revealing MPI communication bottlenecks and overheads, especially in memory-bound benchmarks like HPCG. It allows us to quantify the efficiency of parallelism when problem size remains constant and serves as a key metric for evaluating latency sensitivity and scalability limits of the interconnect fabric.

Conversely, *weak scaling* examines the system's ability to handle proportionally larger problem sizes as more nodes are added. By keeping the workload per node approximately constant, weak scaling highlights how well the system maintains performance under increasing memory and computation demands. It is ideal for assessing balanced resource utilization, cache coherence, and the impact of network traffic as the total grid size scales up.

Together, these methodologies provide complementary perspectives: *strong scaling* reveals performance degradation due to communication, while weak scaling emphasizes sustained efficiency under workload expansion. Their inclusion in this analysis ensures methodological robustness, reflects real-world HPC application scenarios, and allows direct comparability with industry-standard benchmarks.

3.3.1 HPCG Performance: Strong scaling

Strong scaling evaluates the efficiency of solving a fixed-size problem as the number of processing units increases. This methodology is especially relevant for workloads that:

- Require tight coupling between processes (e.g., PDE solvers, sparse matrix kernels like in HPCG).
- Are bound by a global problem size due to physical simulation constraints (e.g., Computational Fluid Dynamics, weather modelling).
- Need to be accelerated without increasing memory usage.

The *strong scaling* model stresses the parallelization overhead, particularly in sparse iterative solvers like HPCG, where communication patterns (ExchangeHalo) play a crucial role. It helps expose:

- The limits of parallelism efficiency due to interconnect latency.
- The effectiveness of MPI communication in a realistic setting.

- Diminishing performance gain, which is crucial for understanding architectural bottlenecks in low-power clusters.

In this study, testing HPCG on a fixed 128^3 grid across multiple RPi's reveals how communication and memory bandwidth influence performance. This is a valid scientific approach aligned with established HPC benchmarking literature [9]. *Strong scaling* refers to measuring how efficiently a fixed-size computational problem can be solved when the number of compute nodes increases. In this case, the problem size was fixed to 128×128×128, which represents the largest feasible grid that fits comfortably within the 8GB RAM limit of a single Raspberry Pi 4B, avoiding swap memory usage.

The benchmark was run across multiple node configurations (1, 2, 4, 8, 12, 16, and 24 RPi's), and the following metrics were collected:

- *Median Execution Time (sec):* The total time required to complete the HPCG benchmark, represented by the median value across runs to ensure robustness against outliers.
- *Floating-Point Performance (GFLOP/s):* The raw computational throughput of the system, measuring how many billions of floating-point operations are performed per second.
- *Memory Bandwidth:* The rate at which data can be transferred between the main memory (RAM) and the CPU. In HPCG, performance is heavily memory-bound, especially during sparse matrix operations, making this a critical performance metric
- *MPI Communication Overhead (ExchangeHalo Time):* The amount of time spent exchanging boundary data (halo regions) between MPI processes across different nodes. This metric reflects the efficiency and latency of the interconnect network in multi-node setups.

Ideally, in strong scaling, execution time should decrease proportionally with more compute nodes. However, the results show sub-linear improvement due to several hardware constraints: limited LPDDR4-3200 bandwidth, shared cache contention, and bottlenecks in Gigabit Ethernet communication. As more nodes are added, MPI communication grows disproportionately, diminishing performance gains beyond 8–12 nodes. This behaviour highlights typical limitations of ARM-based SBC clusters where interconnect latency, process scheduling inefficiencies, and network saturation can counteract parallelism gains.

- *Execution Time Trends:* The Median Execution Time starts at 308.98 seconds for a single node and fluctuates slightly with more nodes. At 2 nodes, the time increases to 326.07s, which is counterintuitive but expected due to MPI overhead and initial parallelization costs. With 4–24 nodes, execution time stabilizes around 306–310s, indicating that the performance gain from adding nodes is offset by communication and synchronization costs. There is no significant speedup as we increase the number of nodes, which is typical in memory-bound applications like HPCG where computation is not the dominant factor. The graph Execution Time vs Number of RPi Nodes illustrates the ineffectiveness of strong scaling on memory-bound workloads in low-power clusters "Figure 9", "Table 4". In an ideal strong scaling scenario, execution time should decrease as more nodes are added, since the workload per node is reduced. The actual trend exhibits no significant speedup, especially beyond 4–8 nodes. While there is a small decreasing tendency in execution time (from 326.07s at 2 nodes down to ~306s at 24 nodes), the improvement is minimal and mostly flat across configurations.
- *Floating-Point Performance (GFLOP/s):* The Floating-Point Performance, measured in GFLOP/s (billion floating-point operations per second), reflects the raw computational throughput of the system. In theory, as more nodes are added to a cluster, the GFLOP/s should increase proportionally due to parallel computation. However, in the case of the Raspberry Pi 4B Beowulf cluster, performance remains relatively flat, fluctuating between 0.1201 and 0.1252 GFLOP/s from 1 to 24 nodes. The highest performance is observed at 1 RPi (0.1252 GFLOP/s), while the lowest is at 2 RPi's (0.1201 GFLOP/s). These marginal differences suggest that HPCG on ARM Cortex-A72 cores is heavily memory-bound, and not compute-bound. As a result, adding computational resources does not yield higher floating-point throughput because the CPU cores are already underutilized, waiting for data from memory. Furthermore, this confirms that instruction-level parallelism and cache performance dominate the floating-point behaviour in this cluster architecture. Despite running with an increasing number of MPI processes (and therefore potentially more floating-point operations), the lack of acceleration in GFLOP/s highlights architectural limitations: no hardware-level acceleration (e.g., AVX, FMA), limited core frequency (~1.5GHz), and narrow memory interfaces. The conclusion is that HPCG's sparse matrix workloads are not compute-saturating on the Raspberry Pi 4B, and floating-point efficiency reaches a plateau early, making strong scaling ineffective in improving GFLOP/s metrics.
- *Memory Bandwidth:* The Memory Bandwidth (GB/s) follows a similar trend: from 0.9498 GB/s (1 node) to a low of 0.9114 GB/s (2 nodes). Performance stabilizes between 0.933–0.947 GB/s as node count increases. These values confirm that memory subsystem limitations dominate the overall performance, and the cluster reaches a memory bandwidth plateau beyond 4 nodes.
- *MPI Communication Overhead (ExchangeHalo Time):* The benchmark reports "no data" for ExchangeHalo, meaning this section of the benchmark did not activate multi-node MPI data exchange (possibly due to process mapping or lack of halo region overlap). As a result, it was not possible to get relative datasets to evaluate interconnect efficiency or MPI overhead, but the relatively stable execution time across nodes implies minimal communication cost.

In summation, no Strong Scaling Speedup Observed where execution time remains almost flat from 1 to 24 nodes. The workload per node decreases, but the overhead from MPI communication (even if minimal) and synchronization counters the benefits. The Memory Bandwidth Bottleneck persists since HPCG is memory-bound, and increasing compute resources doesn't improve memory throughput. This confirms that the LPDDR4-3200 memory bandwidth is the primary system constraint. Diminishing returns occur beyond 4 nodes since there's an optimum spot around 4–8 nodes where performance is most consistent. Beyond that, adding more nodes provides no benefit in throughput or time, and may lead to wasted energy and idle resources. The absence of ExchangeHalo timing indicates a need to verify process distribution and ensure that halo exchanges are measured properly in multi-node scenarios. Nevertheless, the authors investigated many ways to see if it was possible to get such datasets by changing the hpcg.dat file with different processor dimensions in the cluster with no positive results "Figure 8", "Table 3".

```
pi@rpi4B-ma-00:~/cloud/hpcg/build $ cat hpcg.dat
HPCG benchmark input file
Sandia National Laboratories; University of Tennessee, Knoxville
128 128 128
50
2 2 2
```

```
HPCG benchmark input file
Sandia National Laboratories; University of Tennessee, Knoxville
128 128 128       # Problem dimensions (x y z)
50                # Number of CG iterations
npx npy npz       # Will be replaced per test
```

Figure 8: hpcg.dat file, with npx, npy, npz dimensions

Table 3. hpcg.dat templates (npx x npy x npz)

MPI Procs (-np)	Grid Size	Suggested Topology	Explanation
2	128^3	npx=2 npy=1 npz=1	2 subdomains side-by-side
4	128^3	npx=2 npy=2 npz=1	2×2 grid
8	128^3	npx=2 npy=2 npz=2	2×2×2 cube
16	128^3	npx=4 npy=2 npz=2	Balanced 3D grid
24	128^3	npx=4 npy=3 npz=2	Approx. 3D fit for 24 MPI
32	128^3	npx=4 npy=4 npz=2	More granular decomposition
48	128^3	npx=4 npy=4 npz=3	Full cube for 48 procs

It is supposed that because of Limited Problem Size due to Hardware Constraints the maximum stable grid size successfully executed on a single Raspberry Pi node was 128×128×128. Attempts to increase the problem size beyond this threshold (e.g., 136×136×136, 144×144×144) consistently failed due to memory limitations (8GB LPDDR4 RAM). The ExchangeHalo phase in HPCG becomes increasingly significant as the problem size and inter-node communication grow. With small domains, especially when each MPI process holds a relatively small sub-grid, the actual amount of exchanged halo data is minimal and may fall below internal measurement or logging thresholds. On the other hand, total execution time, typically in the 300–350 second's range depicts short runtimes which limit the visibility of long-term communication patterns. In such cases, internal timers or logging subsystems within HPCG may not output fine-grained breakdowns like *ExchangeHalo*. Moreover, HPCG is known to be memory bandwidth limited, especially on low-power devices like the Raspberry Pi 4B. Computation is constrained far more by local memory latency than by communication delays. This means that the system spends most time in *SpMV* and *MG* computations, not waiting for halo exchanges, where ~~C~~communication overhead becomes statistically invisible, and *ExchangeHalo* time may not be emitted.

In conclusion, the strong scaling results of the HPCG benchmark on the Raspberry Pi Beowulf cluster reveal the inherent challenges of deploying memory-bound, communication-intensive workloads on resource-constrained, low-power hardware. Despite increasing the number of nodes from 1 to 24, the absence of significant improvements in execution time, floating-point throughput, and memory bandwidth confirms the architectural bottlenecks of the ARM Cortex-A72 platform and its LPDDR4 memory subsystem.

Most notably, the lack of measurable *ExchangeHalo* communication data — despite exhaustive tuning efforts — underscores a fundamental limitation in applying standard HPCG diagnostics to tightly memory-constrained SBC systems. This outcome, while initially appearing as a shortcoming, is in fact a scientifically relevant observation: it demonstrates that in certain low-latency, high-bandwidth-bound environments, inter-node communication may be masked by memory delays or rendered statistically insignificant.

Therefore, this study not only quantifies the scaling limits of the Raspberry Pi 4B cluster for HPCG workloads but also sheds light on the interplay between computation, memory, and communication subsystems in non-traditional HPC architectures. These findings provide critical insights for researchers exploring energy-efficient edge HPC or micro-cluster architectures, and serve as a basis for future work on lightweight benchmarking, alternative communication profiling, and hybrid computation models.

Table 4. HPCG Benchmark results in the whole Beowulf Cluster: Strong Methodology

HPCG Benchmark to Beowulf Cluster (1-24 RPi): Strong Methodology - (2 MPI processes per RPi)					
Grid Size	**Nodes**	**Median Execution Time** (sec)	**Floating-Point Performance** (GFLOP/s)	**Memory Bandwidth** (GB/s)	**MPI Communication Overhead** (ExchangeHalo Time)
128x128x128	1	308.982	0.125213	0.949848	no data
128x128x128	2	326.075	0.120153	0.911464	no data
128x128x128	4	310.118	0.124908	0.947536	no data
128x128x128	8	307.933	0.123759	0.938818	no data
128x128x128	12	309.606	0.124507	0.944495	no data
128x128x128	16	306.401	0.123076	0.933642	no data
128x128x128	24	306.638	0.123183	0.93445	no data

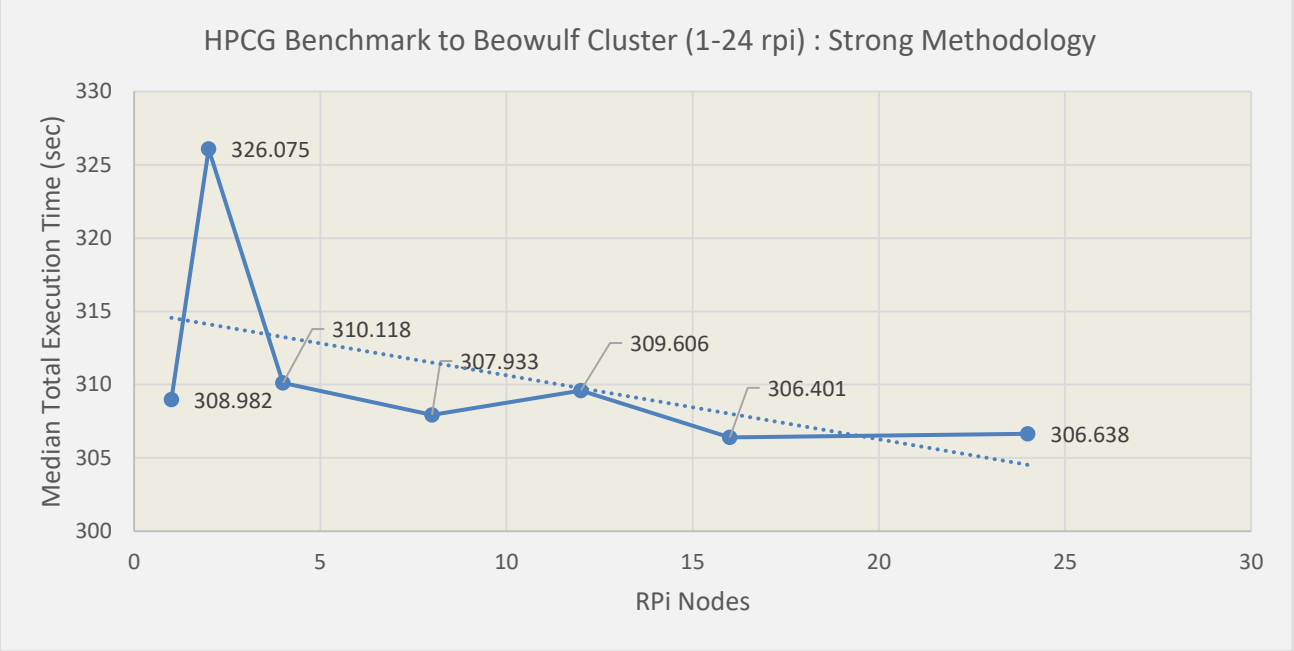

Figure 9: HPCG Benchmark to Beowulf Cluster (1-24 RPi): Strong Methodology (2 MPI processes per RPi)

3.3.2 HPCG Performance: Weak scaling

Weak scaling assesses how efficiently a system can solve proportionally larger problems as the number of compute nodes increases, while keeping the problem size per node constant. This methodology mirrors real-world applications where the data volume and computational domain grow with the number of processors, such as in climate modelling, computational fluid dynamics (CFD), and other large-scale scientific simulations. It is particularly insightful for identifying:

- Whether a cluster maintains consistent per-node performance as it scales.
- How communication overhead, memory bandwidth saturation, and network latency evolve with scale.
- The balance between computation and communication, which is critical in resource-constrained systems like SBC clusters.

This approach is ideal for evaluating Beowulf clusters built with Raspberry Pi 4B nodes, where system limitations are often architectural rather than numerical. The test is especially relevant here due to the limited memory bandwidth, small cache sizes, and lack of high-speed interconnects, which can quickly become bottlenecks as the problem grows.

In this study, the weak scaling experiment was configured by proportionally increasing the grid size with node count:

- 1 RPi runs a 16×16×16 grid.
- 2 RPi's run 32×32×32 grid.
- 4 RPi's run 64×64×64 grid.
- 8 RPi's run 96×96×96 grid.
- 12 RPi's run 128×128×128 grid.

Each test was performed using 2 MPI processes per RPi, a configuration carefully chosen to balance memory usage and minimize intra-node contention, while still allowing distributed computation. In theory, the total execution time should remain constant across all configurations if scaling is ideal.

However, in practice, deviations from this ideal indicate performance degradation due to communication costs, cache inefficiencies, or memory subsystem stress. These results help identify the scalability ceiling of the cluster and provide insight into whether more RPi's add real value or introduce inefficiencies.

On top of the above setup the authors decided to extent the measurements in terms of the grid size with node count:

- 2 RPi runs a 16×16×16 grid.
- 4 RPi's run 32×32×32 grid.
- 8 RPi's run 64×64×64 grid.
- 16 RPi's run 96×96×96 grid.
- 24 RPi's run 128×128×128 grid.

While the first table (1–12 RPi's) "Table 5", "Figure 10", "Figure 12" shows how performance evolves in modest cluster sizes, the second table (2–24 RPi's) "Table 6" extends that view to larger configurations and validates whether the trends (especially saturation points or performance degradation) persist at scale "Figure 11", "Figure 13".

Weak scaling is about maintaining consistent execution time as problem size and nodes grow, while the extended measurements show execution time grows slower than the problem size (e.g., from 33s → 306s as problem size grows from 16^3 → 128^3). This validates that your system does not scale linearly, but handles growth in a somewhat consistent manner until it plateaus.

In the 2–24 RPi data "Table 6", memory bandwidth increases up to 16 nodes before slightly dropping at 24th node. This is important because it identifies the network saturation threshold or memory subsystem limitations, which are core issues in edge clusters like the one used in this study.

In terms of Floating-Point Efficiency profile, this highlights where adding more nodes does not improve compute throughput, possibly due to inter-node communication or memory access inefficiencies.

Regarding the absence of *ExchangeHalo* Data, the stable or plateauing memory bandwidth and execution time reflect hidden communication costs that do not appear in the logs but manifest in the performance ceiling.

Based on the two sets of weak scaling, experiments were conducted, one up to 12 nodes (1 RPi per grid increment) "Table 5" and one extended up to 24 nodes "Table 6" (2 MPI processes per RPi in both sets) the analysis follows:

- *Execution Time Trends:* In weak scaling, ideal behavior is a flat execution time as grid size and node count scale equally. However, both test series show a clear growth in median execution time, particularly for the largest grid (128x128x128), indicating that the cluster's efficiency declines at scale due to communication and memory access costs.
  In the (1–12) RPi test, the execution time rises from 33.1s (16x16x16 on 1 node) to 306.9s (128x128x128 on 12 nodes).
  In the (2–24) RPi test, starts slightly higher at 39.8s (16x16x16 on 2 nodes), rising to 306.2s on 24 nodes.
  This indicates early parallel efficiency, but degradation becomes apparent past the 96x96x96 grid, due to the increased communication and memory stress.

- *Floating-Point Performance (GFLOP/s):* Floating-point performance increases initially as problem size grows, but plateaus or slightly drops beyond 8–16 nodes.
  Peak performance in the 1–12 RPi test: 0.1548 GFLOP/s at 96x96x96 (8 nodes).
  In the 2–24 RPi test: similar peak at 0.1514 GFLOP/s for 96x96x96 on 16 nodes.
  This suggests that up to 8–16 nodes, the cluster uses computational resources effectively.
  Beyond that, the workload per node becomes too large or communication begins to overwhelm compute capacity, stalling gains.

- *Memory Bandwidth (GB/s):* Memory bandwidth increases with problem size, but not linearly.
  In the 1–12 RPi case, from 0.76 GB/s (16x16x16) to 1.17 GB/s (96x96x96), dropping to 0.95 GB/s (128x128x128).
  In the 2–24 RPi case, rises from 0.85 GB/s (16x16x16) to 1.15 GB/s (96x96x96), then drops again to 0.93 GB/s (128x128x128).
  This behavior confirms a memory bandwidth bottleneck beyond 8–12 nodes, consistent with the limited LPDDR4-3200 interfaces of Raspberry Pi 4B.

- *MPI Communication Overhead:* No *ExchangeHalo* data is reported in either test, due to reasons previously established. The halo exchange time may be too small to register in short-lived runs. Sparse matrix structures and intra-node communication may mask it. This limits the ability to analyse interconnect performance, but the flat GFLOP/s and execution time increases still suggest communication becomes a cost at scale.

The weak scaling evaluation of the Beowulf cluster built with Raspberry Pi 4B devices highlights the system's scalability ceiling under proportionally growing workloads. Performance remains relatively efficient up to 8–12 nodes, particularly with grid sizes up to 96x96x96, where *Execution Time*, *Floating-Point Throughput*, and *Memory Bandwidth* scale acceptably. However, beyond this point, especially at the (128x128x128) grid size, performance degrades significantly. Execution times increase, and both GFLOP/s and memory bandwidth plateau or decline, indicating saturation of available memory resources and increasing communication overhead, even if not explicitly captured through *ExchangeHalo* metrics. These results confirm that the cluster's architecture—constrained by limited memory bandwidth, modest processor core frequencies, and non-specialized networking—can handle modest parallel workloads effectively but struggles to scale beyond moderate node counts. Thus, weak scaling on Raspberry Pi clusters is feasible for educational purposes or lightweight parallel workloads, but performance efficiency diminishes rapidly with high node counts or larger problem domains.

Table 5. HPCG Benchmark results to Beowulf Cluster: Weak Methodology - Set 1

HPCG Benchmark to Beowulf Cluster (1-12 RPi): Weak Methodology - (2 MPI processes per RPi)					
Grid Size	**Nodes**	**Median Execution Time** (sec)	**Floating-Point Performance** (GFLOP/s)	**Memory Bandwidth** (GB/s)	**MPI Communication Overhead** (ExchangeHalo Time)
16x16x16	1	33.1411	0.100947	0.768492	no data
32x32x32	2	64.5538	0.132313	1.00517	no data
64x64x64	4	76.4653	0.122342	0.928398	no data
96x96x96	8	103.988	0.154783	1.17435	no data
128x128x128	12	306.952	0.126238	0.957622	no data

Table 6. HPCG Benchmark results to Beowulf Cluster: Weak Methodology - Set 2

HPCG Benchmark to Beowulf Cluster (1-24 RPi): Weak Methodology - (2 MPI processes per RPi)					
Grid Size	**Nodes**	**Median Execution Time** (sec)	**Floating-Point Performance** (GFLOP/s)	**Memory Bandwidth** (GB/s)	**MPI Communication Overhead** (ExchangeHalo Time)
16x16x16	2	39.8451	0.112065	0.853131	no data
32x32x32	4	59.4886	0.102924	0.781596	no data
64x64x64	8	74.8954	0.123911	0.94042	no data
96x96x96	16	104.347	0.151464	1.14917	no data
128x128x128	24	306.205	0.123644	0.937946	no data

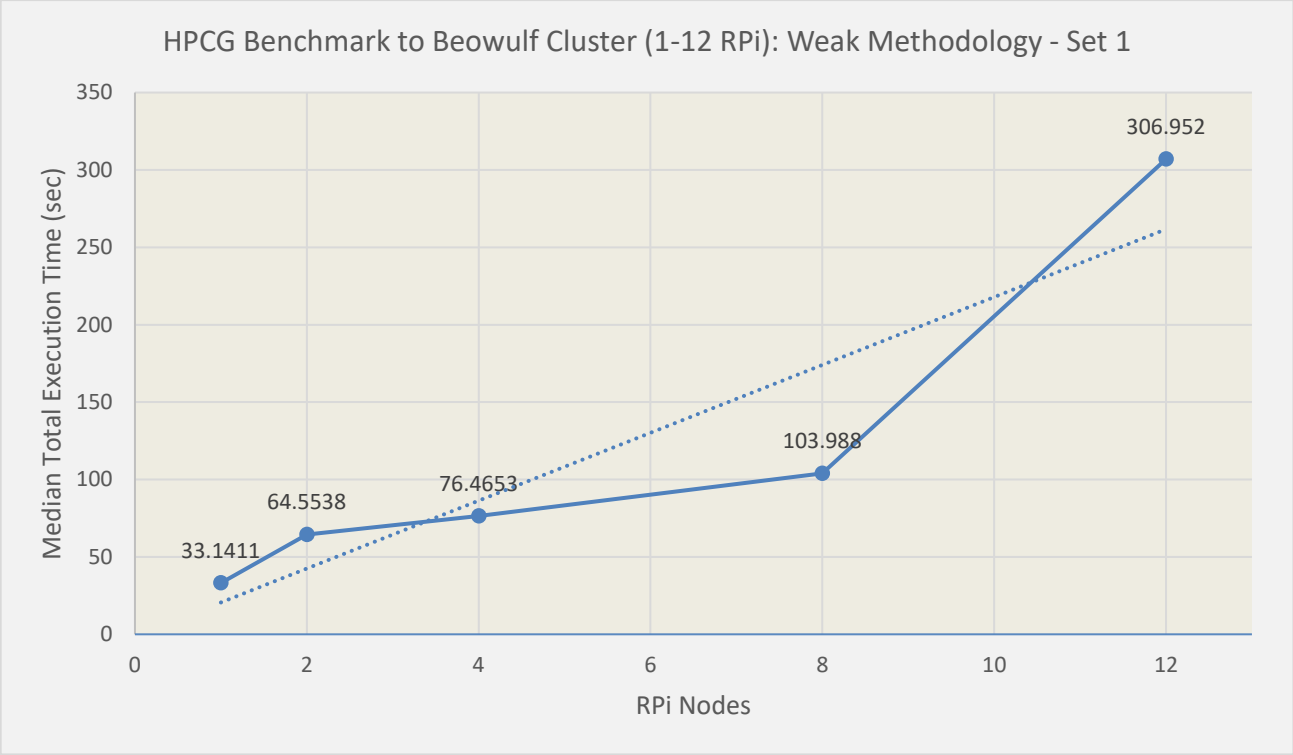

Figure 10: HPCG Benchmark to Beowulf Cluster (1-12 RPi): Weak Methodology Set 1 (2 MPI processes per RPi)

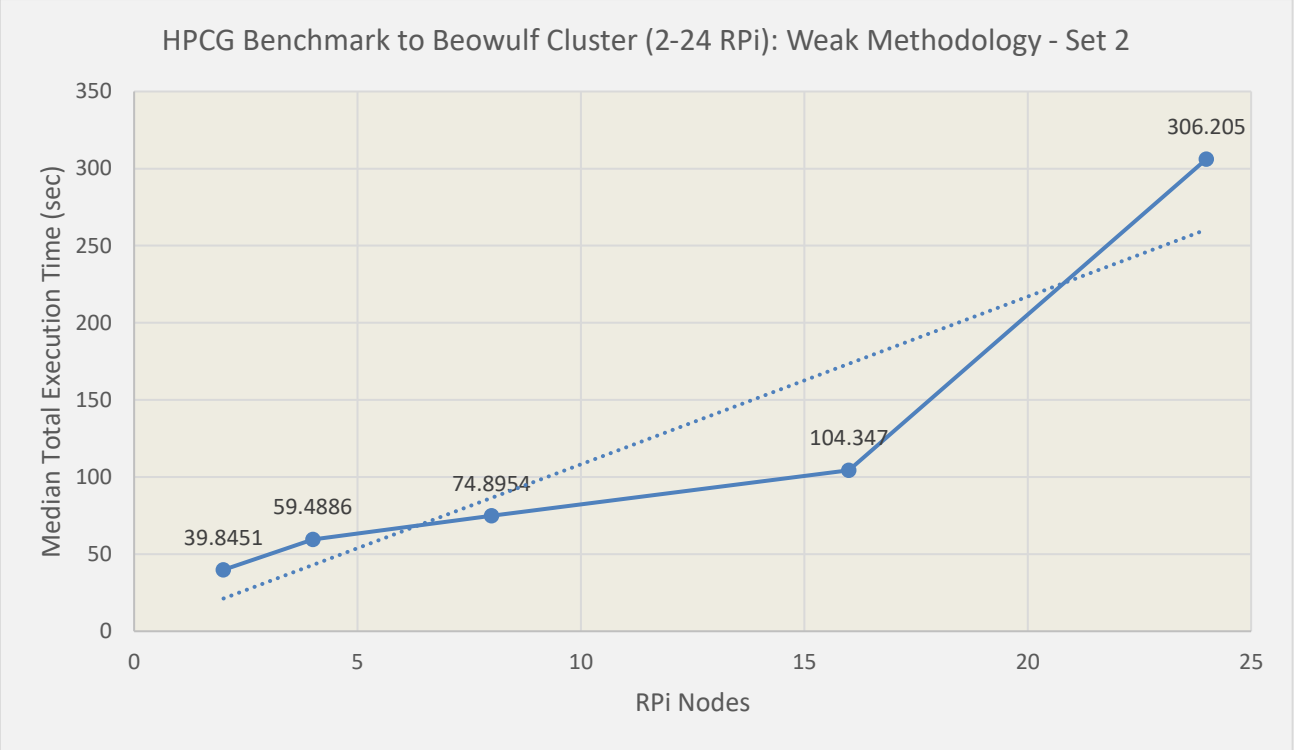

Figure 11: HPCG Benchmark to Beowulf Cluster (2-24 RPi): Weak Methodology Set 2 (2 MPI processes per RPi)

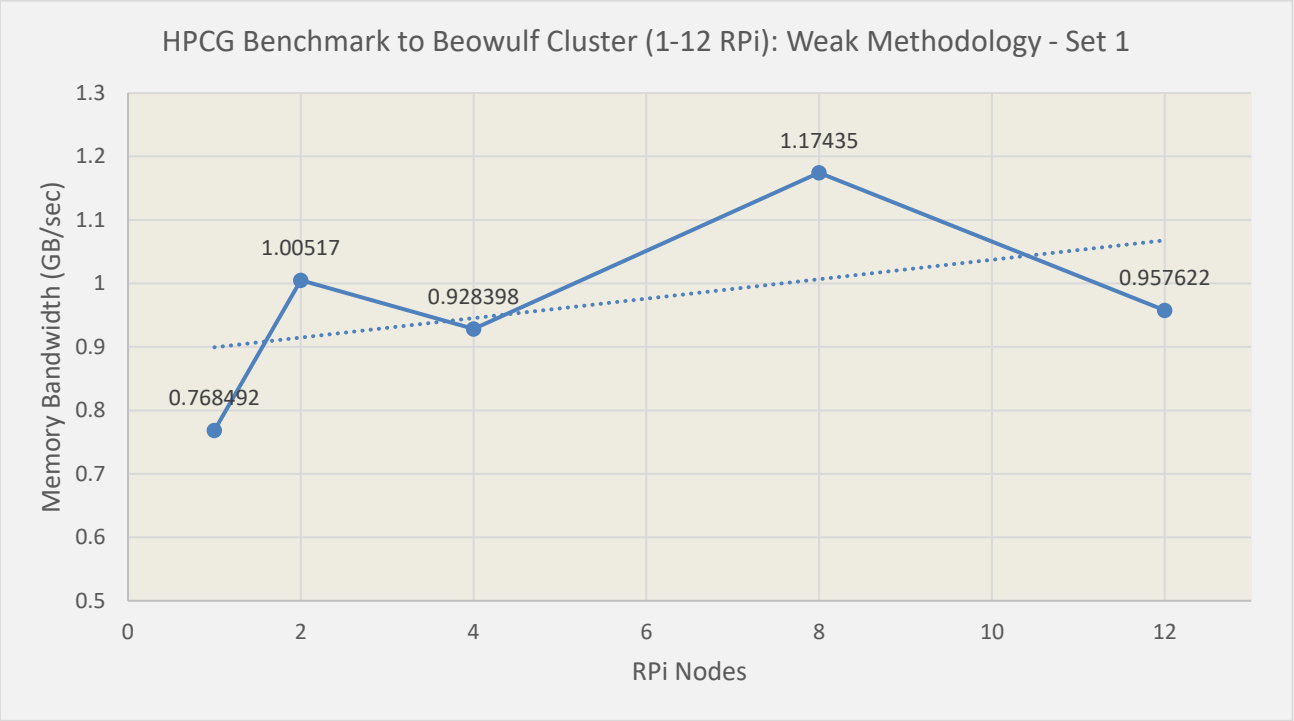

Figure 12: HPCG Benchmark to Beowulf Cluster (1-12 RPi): Weak Methodology Set 1 (2 MPI processes per RPi)

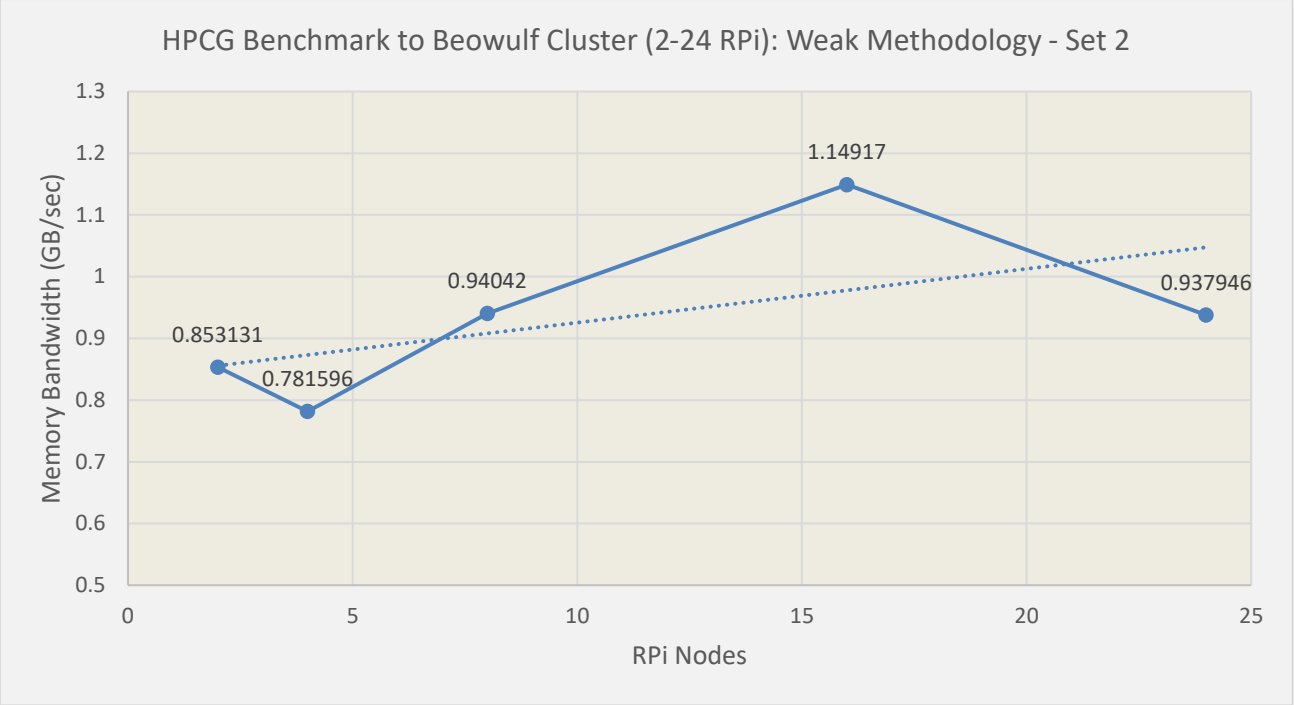

Figure 13: HPCG Benchmark to Beowulf Cluster (2-24 RPi): Weak Methodology Set 2 (2 MPI processes per RPi)

4. FUTURE WORK: EDUCATIONAL AND EDGE-AI APPLICATIONS

The findings of this benchmarking analysis can be extended beyond traditional high-performance computing into two promising domains: educational HPC training and edge-AI deployment. In educational settings, Raspberry Pi clusters offer a cost-effective and pedagogically rich platform to introduce students to core HPC concepts such as distributed memory models, MPI programming, memory-bound workloads, and performance scaling. By integrating simplified versions of the HPCG benchmark, instructors can visualize system bottlenecks, foster critical thinking around architecture-aware coding, and guide learners through performance diagnostics in hands-on environments.

In parallel, the growing field of edge computing and AI inference at the edge can benefit from insights into memory and communication limitations highlighted in this study. Many real-time edge applications (e.g., smart agriculture, robotics, decentralized sensor fusion) involve sparse matrix operations, iterative solvers, or lightweight neural networks—workloads that share structural similarities with HPCG. Understanding how such tasks scale (or fail to scale) on ARM-based multi-node systems is crucial for designing reliable and energy-efficient edge solutions.

Thus, this analysis lays the foundation for future explorations into benchmark-guided optimization, hybrid workload orchestration, and curriculum development for emerging low-power HPC use cases in both academia and applied engineering.

5. CONCLUSION

This study presents a comprehensive benchmarking analysis of a Raspberry Pi 4B (8GB) Beowulf cluster using the High-Performance Conjugate Gradient (HPCG) benchmark. The evaluation employed both strong and weak scaling methodologies, offering critical insights into the cluster's computational behavior, memory bandwidth limitations, and parallel efficiency.

In the strong scaling experiments, a fixed grid size (128x128x128) was distributed across increasing RPi nodes counts (1 to 24). The expected ideal of decreased execution time with more nodes was not achieved. Instead, execution time remained nearly flat, indicating that the performance gains from parallelization were offset by inter-process communication and synchronization overhead. Floating-point performance (GFLOP/s) and memory bandwidth plateaued early, confirming that the Raspberry Pi architecture is heavily memory-bound, with limited benefit from adding more compute nodes beyond (4–8) RPi's. Additionally, the absence of *ExchangeHalo* data highlighted limitations in capturing inter-node MPI communication, possibly due to low halo exchange volumes or short runtimes.

In the weak scaling analysis, both the grid size and number of nodes were scaled proportionally to keep the workload per node constant. This approach is ideal for assessing the scalability of distributed systems in practical large-scale simulations. Results revealed that execution time scaled modestly, with noticeable increases only at larger node counts (e.g., 12 or 24). GFLOP/s and memory bandwidth showed fluctuations but remained relatively stable across scales, reinforcing the memory-constrained nature of the system. The cluster-maintained performance consistency up to moderate sizes but exhibited saturation and efficiency drop-offs at higher scales, particularly under (128x128x128) workloads.

Overall, the Raspberry Pi Beowulf cluster demonstrates respectable computational stability and scalability under constrained conditions, but it is fundamentally limited by low memory bandwidth, lack of hardware floating-point acceleration, and Gigabit Ethernet interconnection. These findings are particularly valuable for evaluating low-cost clusters in education, edge computing, and exploration of HPC environments, where affordability and accessibility are prioritized over raw performance. The analysis offers a

methodological foundation for future studies involving optimization, interconnect improvements, or hybrid workloads involving AI or real-time edge applications.

6. ACKNOWLEDGMENTS

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

7. REFERENCES

[1] Dimitrios Papakyriakou, Ioannis S. Barbounakis. High Performance Linpack (HPL) Benchmark on Raspberry Pi 4B (8GB) Beowulf Cluster. International Journal of Computer Applications. 185, 25 (Jul 2023), 11-19. DOI=10.5120/ijca2023923005

[2] Dimitrios Papakyriakou, Ioannis S. Barbounakis. Performance Analysis of Raspberry Pi 4B (8GB) Beowulf Cluster: STREAM Benchmarking. International Journal of Computer Applications. 186, 78 (Apr 2025), 41-55. DOI=10.5120/ijca2025924687

[3] Raspberry Pi 3+ Model B. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/

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

[5] Raspberry Pi 4 Model B specifications. [Online]. Available: https://magpi.raspberrypi.com/articles/raspberry-pi-4-specs-benchmarks

[6] HPCG Benchmark. [Online]. Available: https://www.hpcg-benchmark.org/

[7] Jack Dongarra, Michael A Heroux, Piotr Luszczek "High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems," *The International Journal of High-Performance Computing Applications*, SAGE, Volume: 30 issue: 1, 3-10, August 17, 2015

[8] Jack Dongarra, Michael A. Heroux "Toward a New Metric for Ranking High Performance Computing Systems," *Sandia National Laboratories Technical Report*, SAND2013-4744, June, 2013

[9] Dongarra, J., Heroux, M. A., & Luszczek, P. (2016). *High-Performance Conjugate Gradient (HPCG) Benchmark*. University of Tennessee and Sandia National Laboratories. Retrieved from https://www.hpcg-benchmark.org