Cluster Performance Benchmarking

High Performance LINPACK (HPL) on a 9-Node Raspberry Pi Cluster

Executive Summary

A 9-node Raspberry Pi 4 cluster was benchmarked using HPL. The single-node configuration (master alone, 4 cores) achieved a verified **29.52 GFLOPS** with HPL's residual check **PASSED**. Cluster-scale runs across all 9 nodes completed but consistently failed the residual check. Systematic diagnosis - eliminating network, MPI, BLAS, storage and RAM as causes - identified **worker undervoltage under HPL load** as the root cause, evidenced directly through Raspberry Pi firmware throttle flags. HPL's residual check correctly detected the silent data corruption that throttling caused, preventing the publication of an unverified GFLOPS figure.

1. Objective

Measure the sustained double-precision floating-point performance (in GFLOPS) of a distributed cluster using the industry-standard High Performance LINPACK (HPL) benchmark - the same workload used to rank the world's fastest supercomputers in the Top500 list.

2. Key Definitions

HPL (High Performance LINPACK)

A portable benchmark that solves a large dense system of linear equations Ax = b in double precision using LU factorisation with partial pivoting. Because the operation count is known (~ $2/3 \times N^3$ for matrix size N), HPL converts wall-clock time directly into floating-point performance. Critically, HPL ends every run with a residual check - if ||Ax - b|| / threshold > 16, the result is marked FAILED and the GFLOPS number is invalid.

GFLOPS

Giga **FL**oating-point **O**perations **P**er **S**econd. One GFLOPS = 10^9 arithmetic operations per second on real numbers.

MPI / MPICH

Message Passing Interface, the protocol HPL uses to coordinate work across nodes. We used MPICH after Open MPI 5.0.7 hit a launcher bug on this hardware.

BLAS / OpenBLAS

Basic Linear Algebra Subprograms - the library that does HPL's heavy arithmetic. We linked against OpenBLAS, configured to run single-threaded per MPI rank.

MPI Rank, P, Q

A **rank** is a single MPI process. **P x Q** is the 2D grid into which ranks are arranged; each rank owns a tile of the matrix. P x Q must equal the number of ranks (-np). Grids closer to square are more efficient. Our cluster ran 36 ranks (one per core across 9 nodes) in a 6 x 6 grid.

3. Cluster Setup

Role	Hostname	IP Address	Cores	RAM
Master	master	192.168.137.10	4	7.9 GiB
Worker	worker1	192.168.137.101	4	~900 MiB
Worker	worker2	192.168.137.102	4	~900 MiB
Worker	worker3	192.168.137.103	4	~900 MiB
Worker	worker4	192.168.137.104	4	~900 MiB
Worker	worker5	192.168.137.105	4	~900 MiB
Worker	worker6	192.168.137.106	4	~900 MiB
Worker	worker7	192.168.137.107	4	~900 MiB
Worker	worker8	192.168.137.108	4	~900 MiB

9 nodes, 36 cores total. Unmanaged gigabit Ethernet switch. Internet via macOS host with Internet Sharing. Heterogeneous memory: master 8 GB, workers ~1 GB - HPL's even partitioning means workers cap the matrix size.

4. Headline Result: Single-Node Benchmark

Configuration: 4 MPI ranks on the master node only, single-threaded OpenBLAS, N = 10,240, NB = 128, P x Q = 2 x 2.

```
T/V N NB P Q Time Gflops
--------------------------------------------------------------------------------
WR11C2C4 10240 128 2 2 24.26 2.9518e+01

||Ax-b||_oo / (eps * (||A||_oo * ||x||_oo + ||b||_oo) * N) = 4.59905368e-03 ... PASSED
```

Metric	Value
Throughput (verified)	29.52 GFLOPS
Wall time	24.26 seconds
Matrix size	N = 10,240
Block size	NB = 128
Process grid	P x Q = 2 x 2 (4 ranks)
Scaled residual	4.60 x 10^-3
Residual check	**PASSED**

Per-core efficiency

Theoretical peak for one Pi 4 node = 4 cores × 1.5 GHz × 4 FLOPs/cycle (NEON) = **24 GFLOPS**. Measured 29.52 GFLOPS exceeds this conservative estimate, indicating the Cortex-A72 delivers more than 4 FLOPs/cycle in dense DGEMM through fused multiply-add (FMA) pairings. Against a more optimistic 48 GFLOPS peak (8 FLOPs/cycle), efficiency is ~62 % - very respectable for an SBC.

5. Cluster-Scale Investigation

Extending the run to all 9 nodes (36 ranks, 6 x 6 grid) consistently produced FAILED residuals. Diagnostic runs were conducted in a controlled-elimination order:

Test	Ranks	Residual	Outcome
Master only	4 (P=2 Q=2)	4.6 x 10^-3	PASSED
Master + worker1	8 (P=2 Q=4)	1.67 x 10^9	FAILED
Worker1 only	4 (P=2 Q=2)	5.8 x 10^9	FAILED
All 9 nodes, N=10240	36 (P=6 Q=6)	1.32 x 10^9	FAILED
All 9 nodes, N=16896	36 (P=6 Q=6)	6.35 x 10^8	FAILED

Critical observation: master alone PASSED, worker1 alone FAILED. Same software, same network (none), same RAM type, same OpenBLAS, same binary. The difference lies in the hardware itself.

Causes systematically ruled out

Hypothesis	Test performed	Verdict
Network corruption (UCX/MPI bug)	Worker1 alone (no network) also FAILED	Ruled out
OpenBLAS install mismatch	md5sum libopenblas.so.0: master = worker1	Ruled out
Binary corruption in transit	md5sum xhpl: master = worker1	Ruled out
SD card silent corruption	Repeated md5sum of same file matched	Ruled out
Worker RAM corruption	dd + md5 of 400MB random data matched	Ruled out
OpenBLAS thread oversubscription	OPENBLAS_NUM_THREADS=1 did not help	Ruled out

6. Root Cause: Worker Undervoltage Under Load

The Raspberry Pi firmware exposes a real-time throttle/undervoltage status via *vcgencmd get_throttled*. The return value is a bitmask where bits 16/17/18 indicate "this condition has occurred since boot":

Bit	Hex	Meaning
16	0x10000	Under-voltage has occurred since boot
17	0x20000	ARM frequency cap has occurred since boot
18	0x40000	Throttling has occurred since boot

Throttle state across all workers (after first cluster runs)

Node	Flag	Decoded
worker1	0x50000	undervoltage + throttling occurred
worker2	0x50000	undervoltage + throttling occurred
worker3	0x70000	undervoltage + ARM cap + throttling occurred
worker4	0x50000	undervoltage + throttling occurred
worker5	0x70000	undervoltage + ARM cap + throttling occurred
worker6	0x70000	undervoltage + ARM cap + throttling occurred
worker7	0x70000	undervoltage + ARM cap + throttling occurred
worker8	0x70000	undervoltage + ARM cap + throttling occurred

Controlled experiment: throttling caused by HPL load

All workers were rebooted to clear the sticky throttle flags. After reboot all 8 workers reported *throttled = 0x0*. Worker1 then ran HPL alone for 67 seconds (N = 6912). Immediately after the run:

```
$ ssh worker1 "vcgencmd get_throttled"
throttled=0x50000
(undervoltage and throttling occurred during HPL load)
```

This is a direct cause-effect observation: the HPL workload itself draws enough current to push worker1's supply below 4.63 V, triggering the firmware's undervoltage protection and CPU throttling. The accompanying computational errors (||x|| = 8.65 in HPL's output, vs the expected ~1 for a correctly solved system) are the silent data corruption characteristic of unstable CPU operation under under-voltage.

7. Mitigation

The fix is hardware. Worker undervoltage under HPL load requires that each Pi receive a clean, stable 5.1 V at 3 A:

- Use the official Raspberry Pi 4 USB-C power supply (15.3 W) for each worker, with its hardwired thicker cable.
- Avoid powering Pis through USB hubs: even powered hubs typically deliver only 1 - 1.5 A per downstream port, insufficient for Pi 4 peak.
- Use short, thick USB-C cables. Long thin cables can drop 0.3 V at 3 A - enough to trigger the threshold.
- Verify by checking *vcgencmd get_throttled* = 0x0 after running HPL for > 1 minute.

• As a temporary software mitigation, capping each Pi's CPU frequency reduces current draw at the cost of ~15 % performance:

```
echo 1200000 | sudo tee /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

8. Analysis

Theoretical full-cluster performance

With adequately powered workers, the cluster's projected ceiling is:

```
Per-node measured: 29.52 GFLOPS
9 nodes × 29.52 GFLOPS = 265 GFLOPS naive ceiling
Realistic cluster efficiency over gigabit Ethernet: ~30 - 50 %
Expected verified cluster result: ~80 - 130 GFLOPS
```

Why the single-node result exceeded conservative peak

Our single-node measurement (29.52 GFLOPS) is 23 % above the conservative theoretical peak of 24 GFLOPS (4 cores × 1.5 GHz × 4 FLOPs/cycle). This is because the Cortex-A72 supports double-precision fused multiply-add (FMA) via NEON: a single instruction performs one multiply and one add, counted as 2 FLOPs but completing in 1 cycle. With FMA pairing, the effective rate can reach 8 FLOPs/cycle, giving an optimistic peak of 48 GFLOPS - against which our 29.5 GFLOPS is a healthy 62 % efficiency.

9. Why the Residual Check Matters

HPL has reported up to 11.9 GFLOPS in failed multi-node runs. Without the residual check, these figures would have looked like plausible benchmark numbers. The check exists precisely to prevent this: any system that produces fast-but-wrong answers is not actually doing the work the benchmark claims to measure. In our case, the residual check detected silent data corruption caused by CPU undervoltage, exposing a hardware infrastructure problem that would otherwise have gone unnoticed and propagated into any subsequent computational use of the cluster.

This is a real engineering finding, not a coursework artefact. In production HPC sites, the residual check (and analogous correctness checks in other benchmarks) is the primary line of defence against what is known industry-wide as **silent data corruption**, a serious and well-documented issue in large-scale computing.

10. Conclusion

HPL 2.3 was successfully built, deployed and executed across a 9-node Raspberry Pi cluster. The verified single-node throughput is **29.52 GFLOPS** (PASSED), achieving 62 % efficiency against the Cortex-A72's theoretical peak. Cluster-scale runs failed HPL's residual check; systematic diagnosis identified worker undervoltage under load as the root cause, with direct evidence from the firmware's throttle status. The cluster is software-ready: with adequate power delivery to the workers, a verified multi-node result on the order of 100 GFLOPS is expected.

Reference: HPL home page - https://www.netlib.org/benchmark/hpl/

Reference: vcgencmd documentation - https://www.raspberrypi.com/documentation/computers/os.html#vcgencmd