Aasish Kumar Sharma

Performance Engineering & Benchmarking

2026-04-09

Course Context And Disclaimer

Welcome

- Topic: Performance engineering and benchmarking on a real cluster

What this session will cover

- A practical introduction to measuring HPC code correctly
- Concepts and standard tools that work on *any* Linux cluster
- Hands-on exercises on the GWDG SCC cluster

What this session will NOT cover

- Vendor-tool tutorial
- Profiler / tracer tools and techniques – covers afternoon session
- MPI or accelerator details

Slides may be recorded; please notify the instructor if you object to your name or voice being captured.

Learning Objectives

By 12:00 you will be able to:

- Sketch the architecture of an HPC cluster
- Compute the theoretical peak FLOP/s and memory bandwidth of a node
- Distinguish compute-bound from memory-bound code
- State Amdahl’s and Gustafson’s laws
- Define speedup, parallel efficiency, strong scaling, weak scaling
- Run a microbenchmark correctly (warmup, repetitions, pinning)
- Interpret a scaling curve and know when to stop optimising

Time Plan – 09:00 To 12:00

Time	Block	Content
09:00 – 09:10	Welcome	Title, intro, objectives, plan
09:10 – 09:55	**Part I**	Performance Engineering + demo + Exercise 1
09:55 – 10:05	*Break*	10 minute break
10:05 – 10:55	**Part II**	Benchmarking: rules, demos, Exercise 2
10:55 – 11:05	*Break*	10 minute break
11:05 – 11:55	**Part III**	Practical: scaling study (Exercise 3)
11:55 – 12:00	Wrap-up	Bridge to afternoon profiling/tracing session

One question, three parts

→ *How do I know if my job is using the cluster well?*

Part I – Introduction to Performance Engineering

Why measure?
What are the ceilings?
What do the laws say?

- Lecture: 25 minutes
- Live demo: 10 minutes
- Exercise 1: 10 minutes

Yesterday You Learned MPI – Today's Question Is Harder

Is your parallel program any *good*?

- Your job runs in 10 minutes on 32 cores instead of 5 hours on 1 core
- Is that good? Could it be 5 minutes? 1 minute?
  - If yes, how? If not, why not?
- Is "Faster than before" is same as "fast"?
- Today you will learn to tell the difference,
  - With proper measurements, not opinions.

Why Performance Matters In HPC

- HPC time is expensive
  - A node-hour is real money
- HPC time is shared
  - Your inefficiency is somebody else's wait time
- HPC time is measurable
  - We can know exactly how much we wasted
- Most scientific code uses less than 10% of the hardware it runs on
  - Hager & Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press, 2010.
- Performance engineering closes that gap on purpose

What We Do?

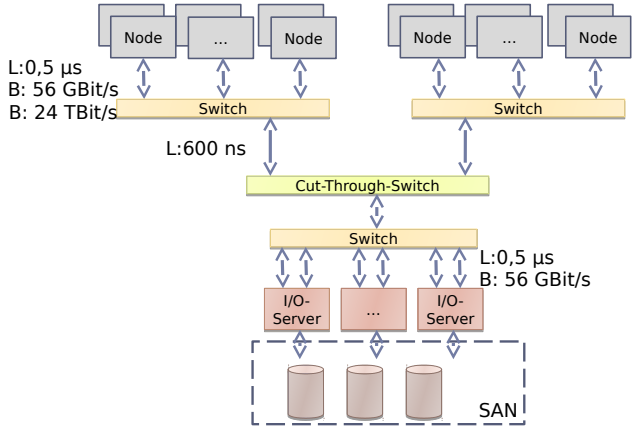
Before we measure, we have to know the hardware

- Performance lives in the silicon and the wires
- Same code runs at different speeds on different hardware
- Different hardware has different ceilings
- Before asking “is my code good?” ask “what does this hardware allow?”

The next slides build a top-down picture of an HPC cluster

Cluster → node → socket → NUMA domain → cache → core

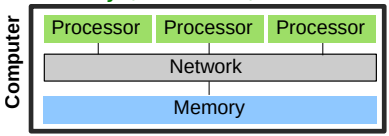
HPC Cluster – Compute, Storage, And Network



- Many compute nodes joined by a fast network,
- Storage provided by Shared parallel filesystem,
- Today we know the hardware and measure one of a compute node.

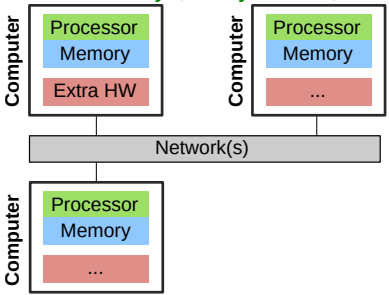
Two Memory Paradigms In HPC

Shared memory (one node)



- All cores see one address space
- Programmed with OpenMP / pthreads
- Limited to one node

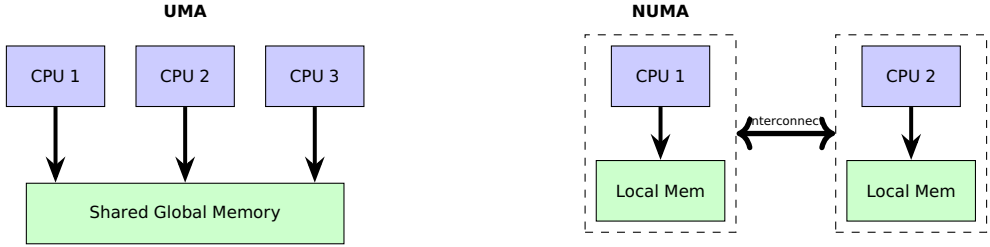
Distributed memory (many nodes)



- Each node has its own RAM
- Communication via MPI
- Scales to thousands of nodes

Today's exercises live *inside one node* – shared memory, multiple NUMA domains.

UMA vs NUMA – Inside A Single Node



- **UMA** – one shared bus, all memory equally far – does not scale past a few cores
- **NUMA** – memory split into domains; local fast, remote slower
- All modern multi-socket HPC nodes are NUMA
- The SCC `amp` node has 4 NUMA domains

Performance Engineering Ideas

Idea 1 – Time Alone Is Not A Metric

- "My job took 12 minutes." So what?
- Compared to what – yesterday? a friend's laptop? the Top500?
- Slow on slow hardware $\neq$ slow on fast hardware

To interpret time, we need a reference

- The reference is the **theoretical peak** of the hardware:
  - Peak **FLOP/s** – arithmetic ceiling
  - Peak **memory bandwidth (GB/s)** – data-movement ceiling

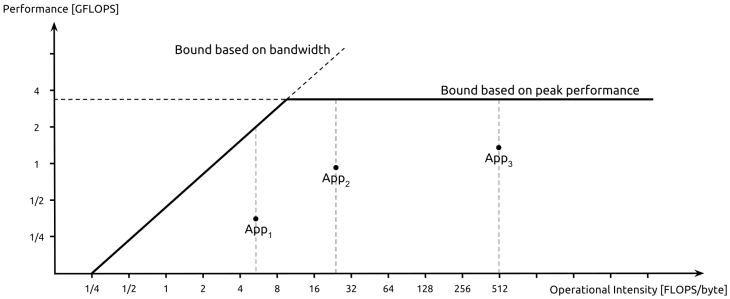
Performance Engineering Ideas (Contd..)

Idea 2 – Every Code Has Two Ceilings

- Every loop is bounded by either **compute** (FLOP/s) or **bandwidth** (GB/s)
- Which one bites depends on **arithmetic intensity** = flops / bytes
- Optimising the wrong ceiling is wasted effort

Source: Williams, Waterman & Patterson, “Roofline: An Insightful Visual Performance Model for Multicore Architectures”, CACM, April 2009.

The Roofline Model – The Full Picture



- X-axis: arithmetic intensity (flops per byte)
- Y-axis: achievable performance (GFLOP/s)
- Sloped line = bandwidth ceiling; flat line = compute ceiling
- Your kernel sits below the lower of the two – never above

Image: Giu.natale / Wikipedia (CC BY-SA).

Why Most Real HPC Code Is Memory-Bound

- CPUs grew FLOP/s much faster than memory bandwidth
- Typical Xeon machine balance – $\sim$10–20 flops per byte
- Most scientific kernels – below 5 flops per byte
- Stencils, sparse matvec, BLAS-1, FFT – all memory-bound
- Dense matrix-matrix multiply (BLAS-3) – compute-bound (rare)
- This is why we will spend more time on bandwidth than on FLOP/s

Source: Hager & Wellein (2010), Chapter 1; STREAM benchmark history (McCalpin 1995).

Performance Engineering Ideas (Contd..)

Idea 3 – Parallelism has a ceiling: Amdahl

Amdahl's Law (1967)

$$\text{Speedup}(P) = \frac{1}{s + (1-s)/P} \quad \xrightarrow[P\to\infty]{} \quad \frac{1}{s}$$

where s = serial fraction, P = number of cores.

- 5% serial $\Rightarrow$ max speedup = 20$\times$, no matter how many cores
- Doubling cores never quite halves the time
- Serial code is a tax you pay forever
- **Amdahl's question:** "Can I solve *my problem* faster?"

Source: G. M. Amdahl, AFIPS Conference Proceedings, 1967.

Amdahl's Measure In Numbers – 5% Serial Kills Your Performance

Cores	$s = 0$	$s = 0.05$	$s = 0.10$	$s = 0.20$
1	1.0	1.00	1.00	1.00
2	2.0	1.90	1.82	1.67
4	4.0	3.48	3.08	2.50
8	8.0	5.93	4.71	3.33
16	16.0	9.14	6.40	4.00
32	32.0	12.55	7.80	4.44
96	96.0	16.55	9.34	4.76
∞	∞	20.00	10.00	5.00

At $s = 0.05$, going from 32 to 96 cores buys only $1.32\times$ extra speedup – for $3\times$ the resources.

Gustafson's Law (1988) – The Optimist

The Setup – A Different Question

- Do not fix the problem size. Fix the *time*. Grow the problem with P:

$$\text{Scaled Speedup}(P) = s + (1 - s) \cdot P$$

- Linear in P – no asymptotic ceiling.
- In real HPC we usually *do* grow the problem when we get more cores
- Finer mesh, longer simulation, larger dataset
- **Gustafson's question:** "Can I solve a *bigger* problem in the same time?"

Source: J. L. Gustafson, "Reevaluating Amdahl's Law", CACM, May 1988.

Strong Vs Weak Scaling – Two Laws, Two Questions

Strong Scaling – Amdahl Law

- Fixed total problem size
- Add cores; time should drop
- Limited by serial fraction
- “Solve my problem faster”

Weak Scaling – Gustafson Law

- Fixed work per core
- Add cores; problem grows
- Limited by communication
- “Solve a bigger problem”

Concrete Example: Weather Forecast

- Strong: same 1° resolution, more cores $\Rightarrow$ result in 30 min instead of 2 h
- Weak: more cores $\Rightarrow$ run a 0.25° resolution in the same 2 h

Speedup And Parallel Efficiency

Speedup

$$S(P) = \frac{T_1}{T_P}$$

- Wall time on 1 core divided by wall time on P cores. Ideal: $S(P) = P$.

Parallel efficiency

$$E(P) = \frac{S(P)}{P}$$

- Fraction of ideal speedup achieved. Ideal: $E(P) = 1.00$.

$\rightarrow$ Always report *both* the **wall time** and the **efficiency**, with the rank count.

Bridge – From Theory To Your Node

Measure A Node

- We talked about peak performance – what is the peak *here*?
- Most students never look at the node they run on
- We are going to look – it takes 60 seconds
- Once you have done it, you will do it on every cluster you ever touch

Four standard tools, available on *any* Linux cluster

- `lscpu`
- `numactl -H`
- `free -h`
- Pen + Paper

Live Demo – Inspect A Compute Node On SCC

Step 1. Get to a compute node (*never* the login node):

```
ssh u<id>@login-mdc.hpc.gwdg.de
module load gcc/14.2.0 openmpi/5.0.6
srun -p medium -N 1 -n 1 --time=00:30:00 --pty bash
hostname
```

Step 2. Inspect the silicon and the memory layout:

```
lscpu
numactl -H
free -h
```

What "lscpu" Tells Us About An SCC "amp" Node

Model name	Intel Xeon Platinum 9242 @ 2.30 GHz
Sockets	2
Cores per socket	48
Threads per core	1 (HT off – clean for benchmarking)
Physical cores per node	**96**
Base clock	2.30 GHz
Vector ISA	AVX-512 (8 doubles per SIMD register)
L1d / L2 / L3 cache	32 KB / 1 MB / ∼36 MB
NUMA domains	4 (24 cores each)
RAM per node	∼382 GB

Now we have everything we need to compute the peak.

Computing Peak FLOP/s – Live On The Whiteboard/Dashboard

Per-core peak (double precision)

$$\text{peak/core} = \text{clock} \times \text{FMA units} \times \text{SIMD width} \times 2$$

- Clock = 2.30 GHz
- FMA units = 2 (Cascade Lake)
- SIMD width = 8 doubles (AVX-512)
- $\times 2$: FMA = multiply + add = 2 ops

$$\text{peak/core} = 2.30 \times 2 \times 8 \times 2 = \mathbf{73.6}\ \text{GFLOP/s}$$

$$\text{peak/node} = 73.6 \times 96 = \sim \mathbf{7.07}\ \text{TFLOP/s}$$

Computing Peak Memory Bandwidth

From the CPU spec sheet

$$\text{Peak BW} = \text{channels} \times \text{sockets} \times \text{width} \times \text{MT/s}$$

For Xeon 9242 (12 channels per socket, DDR4-2933, 8 B per transfer):

$$12 \times 2 \times 8 \times 2933 \approx \mathbf{563}\ \text{GB/s per node}$$

Reality check

- Real STREAM TRIAD on this node measures $\sim$**347 GB/s = 62% of peak**.
- That ratio is your performance budget.

Measure Memory In 30 seconds

NUMA Node

- This node has 4 NUMA domains
  - Each set of 24 cores has its own local memory
- Local access is fast;
  - Cross-domain access is slower
- **First-touch policy:** a memory page lives where it was first written
- If thread 0 alone initialises an array, it sits on NUMA 0
- Other 3 domains then read remotely $\Rightarrow$ bandwidth collapses 3–4$\times$
- Lesson: **initialise in parallel, the same way you compute in parallel**

Exercise 1 – Know Your Node (10 minutes)

Goal:

- For the SCC node you are allocated,
  - Find the hardware spec and compute its theoretical peak FLOP/s and peak memory bandwidth.
  - Write your answers in `worksheet.md`.

```
ssh u<id>@login-mdc.hpc.gwdg.de
module load gcc/14.2.0 openmpi/5.0.6
srun -p medium -N 1 -n 1 --time=00:10:00 --pty bash
lscpu
numactl -H
free -h
```

- Compute peak FLOP/s per core, then per node
- Compute peak memory bandwidth per node
- Answer: how many NUMA domains? Why does that matter?

Wrap-Up Of Part I

What you now know

- Every cluster has two ceilings: peak FLOP/s and peak memory BW
- For SCC `medium`: ∼7.07 TFLOP/s and ∼563 GB/s per node
- Most HPC code is memory-bound, not compute-bound
- Amdahl bounds strong scaling; Gustafson describes weak scaling
- Speedup, efficiency, strong/weak scaling are how we report parallelism

10-minute break – back at 10:05.