Aasish Kumar Sharma

Performance Engineering & Benchmarking

2026-04-09

Revision Of Parts I And II

What we have measured so far

- Hardware ceilings of one SCC node ($\sim$7 TFLOP/s, $\sim$563 GB/s)
- Real STREAM TRIAD: 347 GB/s on a full 96-core node (62% of peak)
- The 8 rules of correct benchmarking

The natural next question

$\rightarrow$ *We can characterise one core. What happens with many?*

Part III – Practical: Scaling Study

Watch Amdahl bite, in real time.

- Walk through `mpi_pi.c`
- Live demo with 1, 2, 4 ranks
- Exercise 3: full strong + weak scaling sweep
- Interpret the curve and bridge to profiling (after lunch)

The Kernel: Parallel PI Via MPI

- **Numerical integration of $4/(1+x^2)$ on $[0,1]$. Embarrassingly parallel.**

```
double local_sum = 0.0;
for (long i = rank; i < N; i += size) {
    double x = h * ((double)i + 0.5);
    local_sum += 4.0 / (1.0 + x * x);
}
double local_pi = h * local_sum;
double pi = 0.0;
MPI_Reduce(&local_pi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

→ Each rank computes every P-th interval. Single `MPI_Reduce` at the end. Full source in `code/mpi_pi.c`.

Demo

Ranks - 1, 2, 4

```
make mpi_pi          # builds mpi_pi (needs module openmpi)
mpirun -np 1 ./mpi_pi 400000000     # ~0.6 s
mpirun -np 2 ./mpi_pi 400000000     # ~0.3 s  ( ~2x faster)
mpirun -np 4 ./mpi_pi 400000000     # ~0.16 s ( ~4x faster)
```

- Doubled the cores, halved the time – almost
- Not exactly half – that is the Amdahl tax

Exercise 3 – Strong And Weak Scaling On SCC (35 minutes)

Goal:

- Run `mpi_pi` on $1, 2, 4, 8, 16, 32, 64, 96$ ranks on a single SCC node,
- Plot the curve, interpret it.

```
cd ~/pchpc-perf/code
sbatch scaling.sbatch          # runs strong + weak sweep, writes results.csv
squeue --me
cat results.csv
python3 plot_scaling.py results.csv
ls scaling.png
```

  - Then fill in the worksheet tables for both strong and weak scaling.
  - Compute speedup $S(P) = T_1/T_P$ and efficiency $E(P) = S(P)/P$ for every rank count.

Reference Numbers From SCC "amp095" (96-Core Run)

Strong scaling ($N = 4 \times 10^8$):

Ranks	Wall (s)	Speedup	Efficiency
1	0.606	1.00	100%
2	0.307	1.97	99%
4	0.157	3.87	97%
8	0.084	7.20	90%
16	0.047	12.81	80%
32	0.027	22.78	71%
64	0.017	36.30	57%
96	0.012	**49.71**	**52%**

Weak scaling: wall stays $\sim$0.012 s across all ranks, efficiency $\sim$92% at 96.

Interpreting The Curve

The most important 5 minutes

- Strong-scaling efficiency at 96 ranks is 52% – is this good?
- Weak-scaling efficiency at 96 ranks is 92% – why higher?
- Strong: per-rank work shrinks; serial overhead grows proportionally
- Weak: per-rank work fixed; only the reduce tree grows ($\log P$)
- Both correct – they answer different questions

Back-fitting Amdahl

- From $S(96) = 49.71$, the implied serial fraction is $\sim$**0.98%**.
- Less than 1% of serial code costs us half the cores at full count.

Bridge To The Afternoon Session

What we measured today

- Hardware ceilings (Part I)
- Memory bandwidth, % of peak (Part II)
- Strong / weak scaling, speedup, efficiency (Part III)

What we did NOT measure

- *Which line* of code is causing the inefficiency
- *Where* in the call stack the time goes
- *Why* a cache miss happens

*We measured the **what**. Profiling tells you the **why**.*
Hand over to afternoon session at 13:00.

Take-Home Checklist

- Always measure on a compute node, never the login node
- Always warm up and take multiple repetitions
- Always use the result, or the compiler deletes the loop
- Always pin threads / processes to cores
- Always compare against a ceiling (peak FLOP/s or peak BW)
- Always run strong AND weak scaling for parallel codes
- Always record wall time, ranks, problem size, hostname in a CSV

The one sentence to remember

$\rightarrow$ *Measure on a compute node, warm up, repeat, pin, use the result, compare to peak.*

References – Foundational Papers

- G. M. Amdahl, *Validity of the single processor approach to achieving large scale computing capabilities*, AFIPS Conference Proceedings, 1967.
- J. L. Gustafson, *Reevaluating Amdahl's Law*, Communications of the ACM, 31(5):532–533, May 1988.
- S. Williams, A. Waterman, D. Patterson, *Roofline: An Insightful Visual Performance Model for Multicore Architectures*, Communications of the ACM, 52(4):65–76, April 2009.
- J. D. McCalpin, *Memory bandwidth and machine balance in current high performance computers*, IEEE TCCA Newsletter, December 1995. (STREAM benchmark)

References – Recommended Books

- G. Hager, G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press, 2010 – the standard textbook on the topics covered today.
- J. L. Hennessy, D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th edition, Morgan Kaufmann, 2017.
- V. Eijkhout, *Introduction to High Performance Scientific Computing*, freely available at `https://web.corral.tacc.utexas.edu/CompEdu/`
- T. Rauber, G. Rünger, *Parallel Programming for Multicore and Cluster Systems*, 3rd edition, Springer, 2023.

References – A Few Online Resources And Tools

- STREAM benchmark: https://www.cs.virginia.edu/stream/
- Top500 / Green500: https://www.top500.org/
- HPCG benchmark: https://www.hpcg-benchmark.org/
- LIKWID performance tools: https://github.com/RRZE-HPC/likwid
- GWDG SCC documentation: https://docs.hpc.gwdg.de/
- NHR Alliance training catalogue: https://www.nhr-verein.de/

Now You Know

How do you know if your job is using the cluster well?

Questions?

Aasish Kumar Sharma
aasish-kumar.sharma@gwdg.de
Institute of Computer Science, Georg-August-Universität Göttingen
GWDG mbH, Göttingen