



SURF

Performance analysis

Maxim Masterov

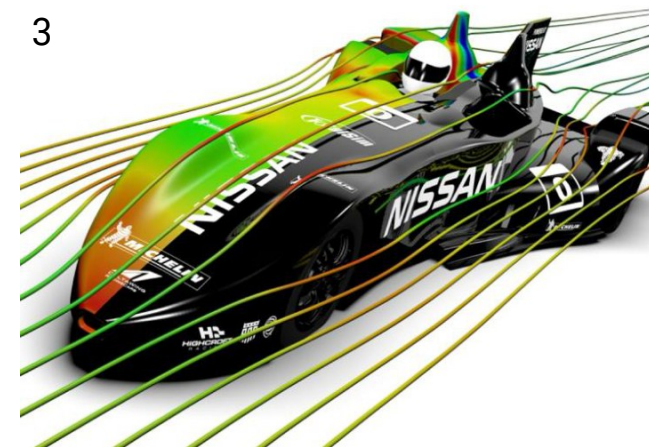
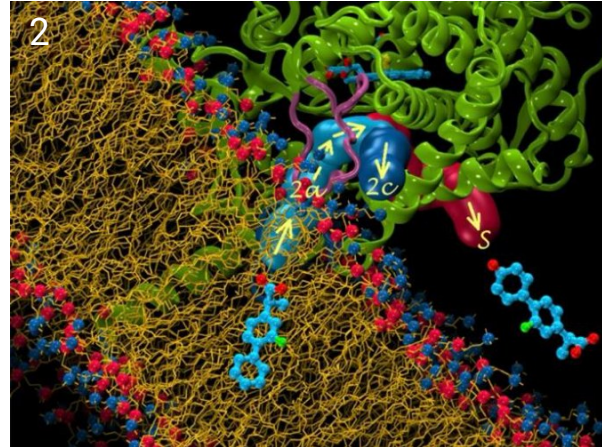
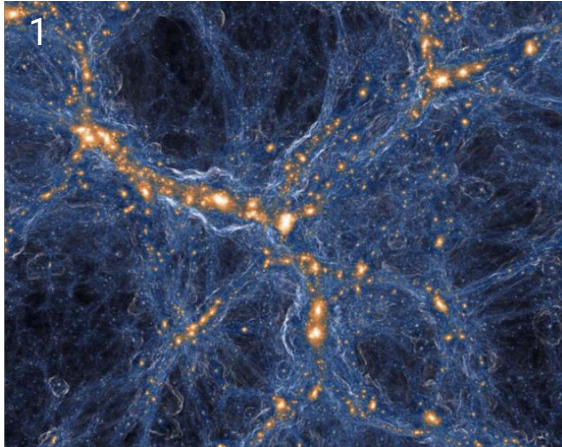
15.05.2025

Adapted and updated by Rohit Goswami

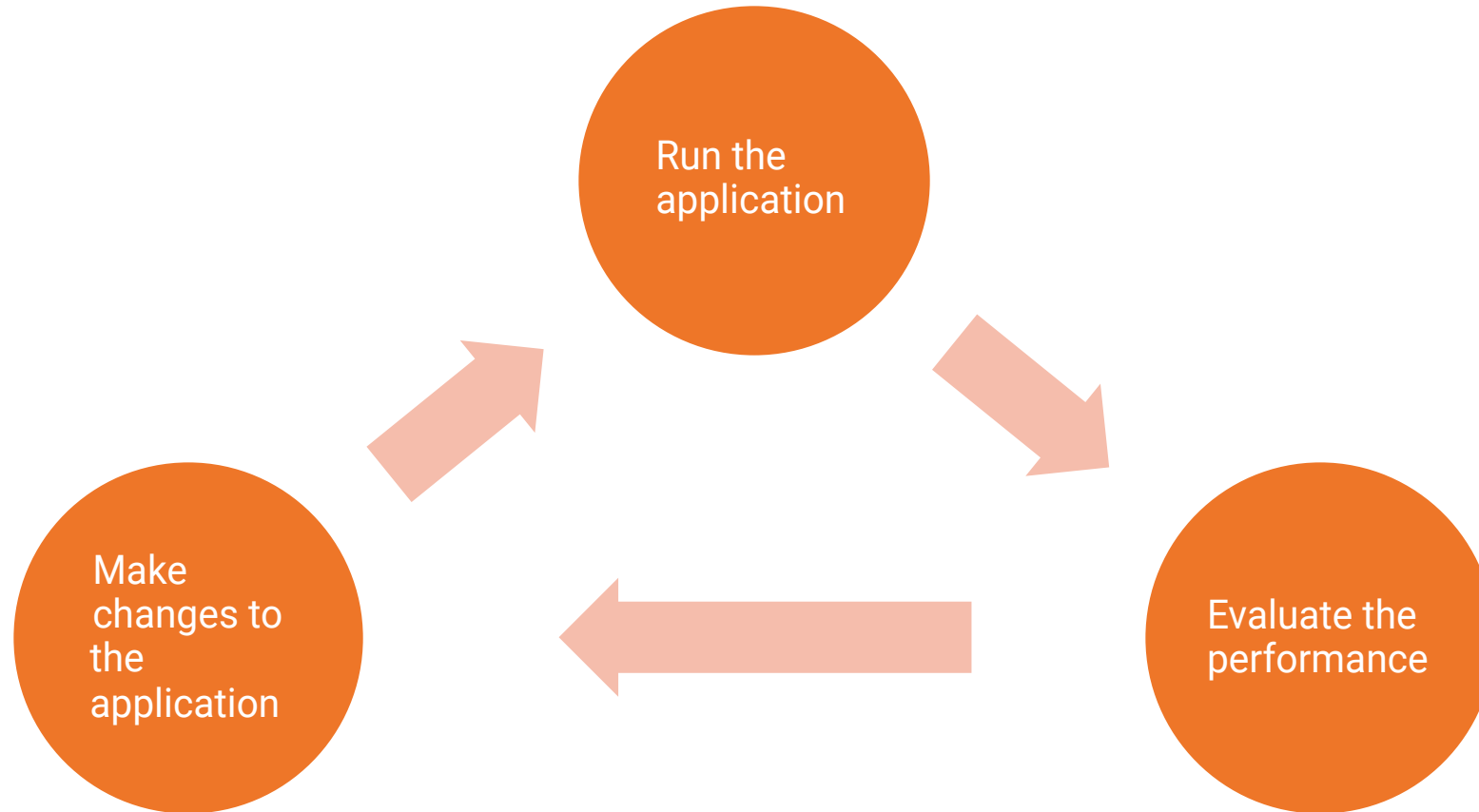
Performance analysis

| Overview of HPC applications

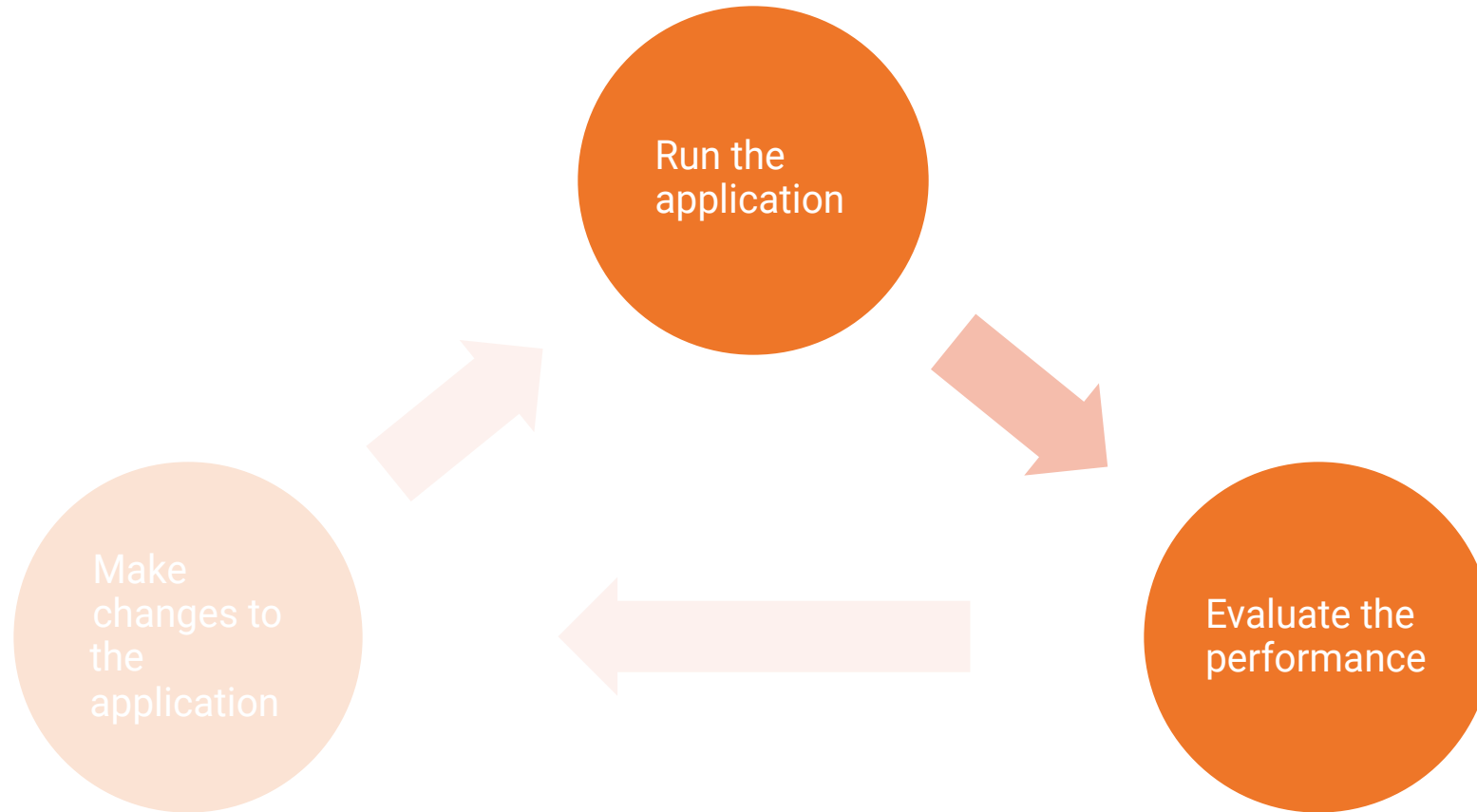
- HPC applications are software designed to exploit the power of supercomputers or high-performance clusters to solve complex problems
- HPC applications perform intensive computations, run communications between parallel processes, and manage input/output operations on a high-performance file system



| A life cycle of an HPC application

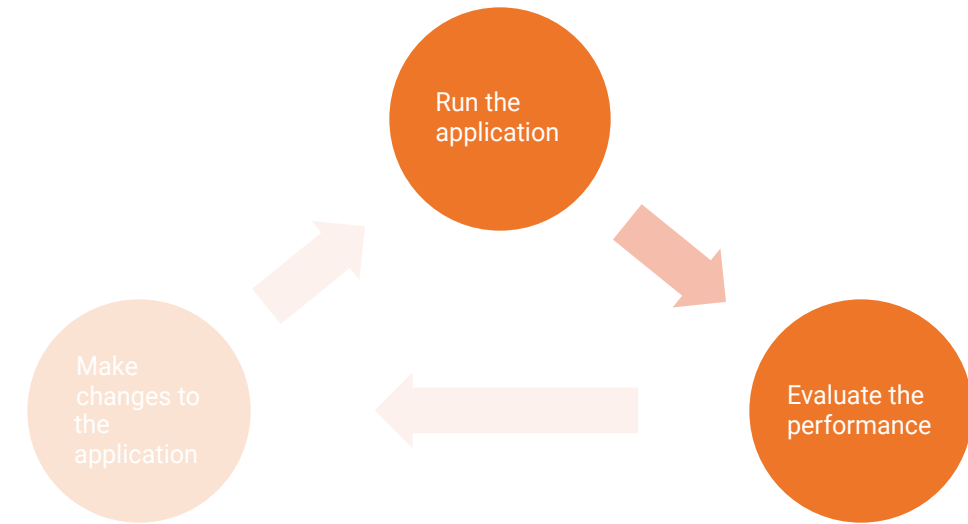


| A life cycle of an HPC application



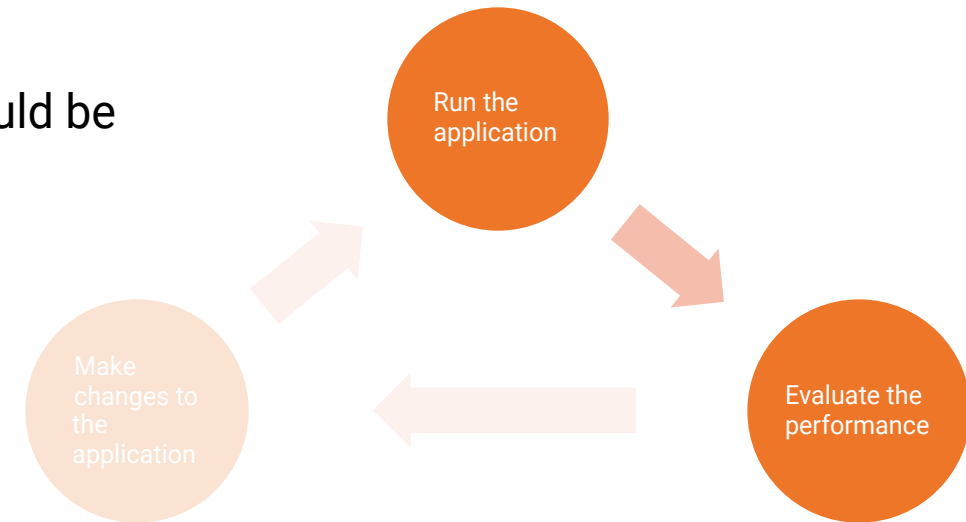
| How to evaluate the performance?

- Using tools, e.g. Score-P, Scalasca, Nsight Systems, AMD uProf
- Measuring the execution time
- Measuring the memory footprint
- Tracking the network load
- Comparing the performance with similar applications/models
- Comparing the performance across different hardware
- Using “gut feeling”



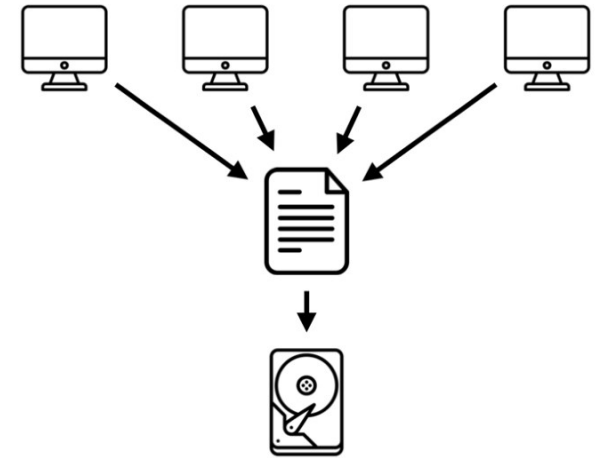
| Performance analysis

- An application consists of multiple levels of complexity
- Each level may have certain effect on the performance and should be evaluated individually:
 - IO
 - **Communications**
 - **Computations**
 - **Memory footprint**



| Performance analysis - IO

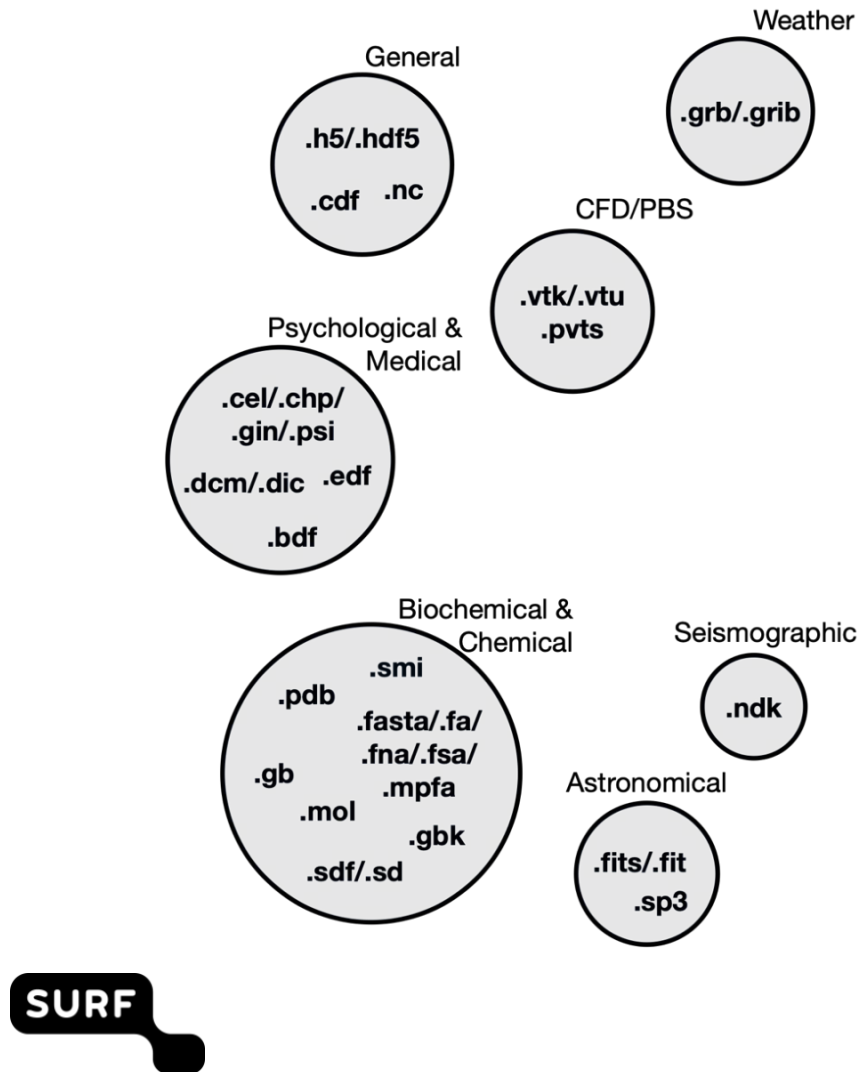
- Parallel vs sequential filesystems
- We can measure read/write speed
- Data can be ASCII or binary
- Different file formats help, as some are sequential, some are parallel
- The easiest way to evaluate performance is to measure the elapsed time of IO operations



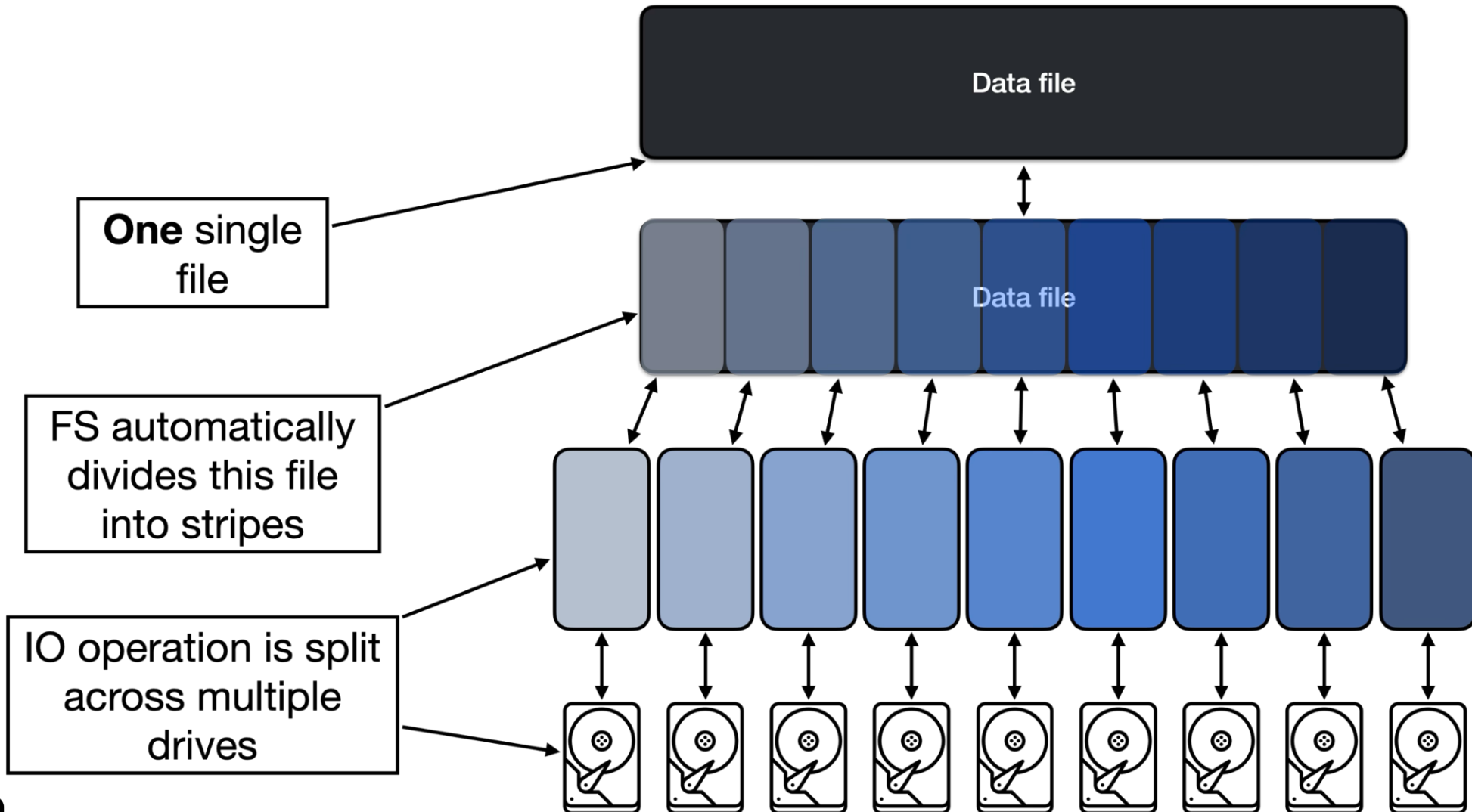
10M vector of 64-bit FP values on NVME file system

Format	IO operation	Speed, [MB/s]
ASCII	Read	29.7
	Write	24.9
Binary	Read	2666.4
	Write	1586.5

| Performance analysis - IO



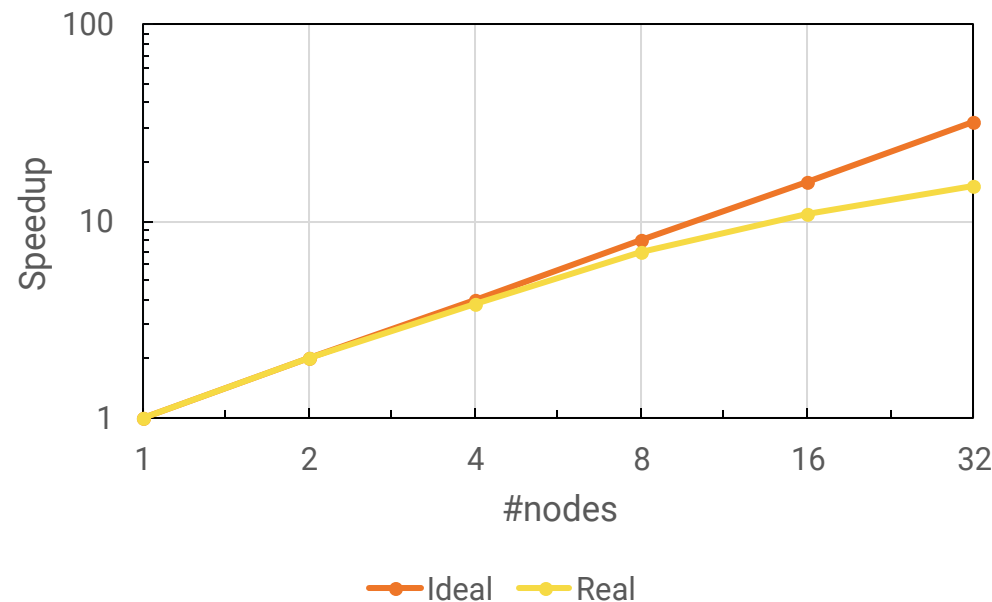
| Performance analysis - IO



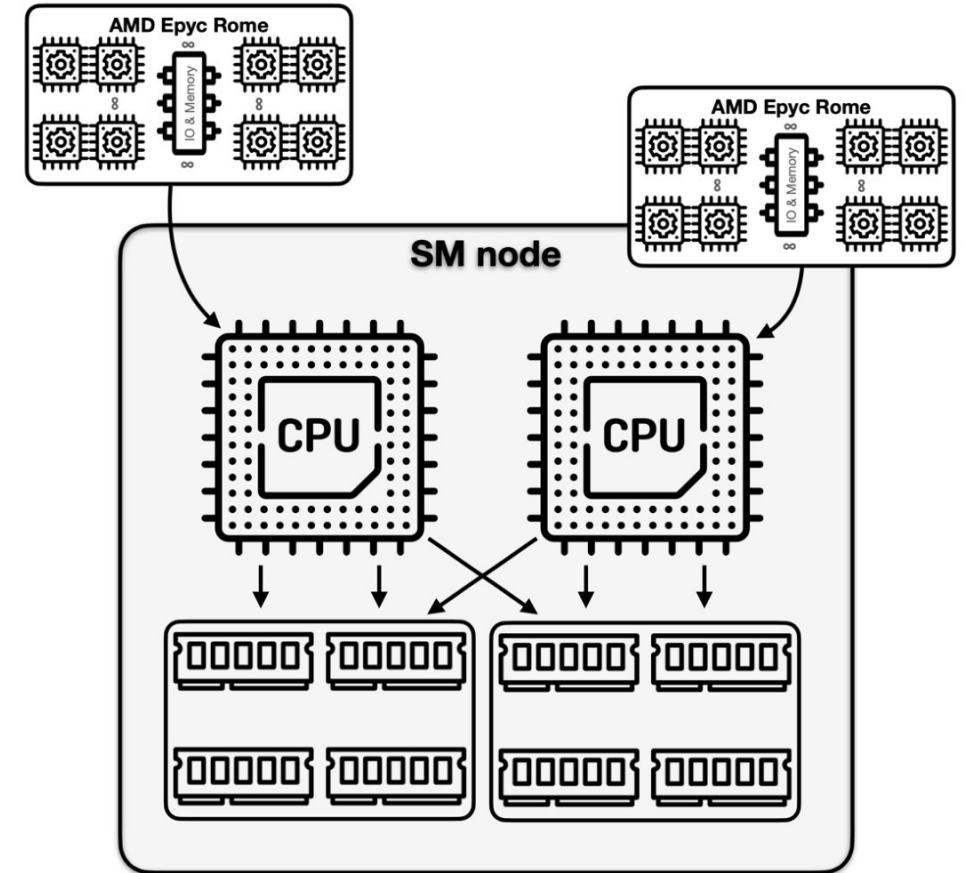
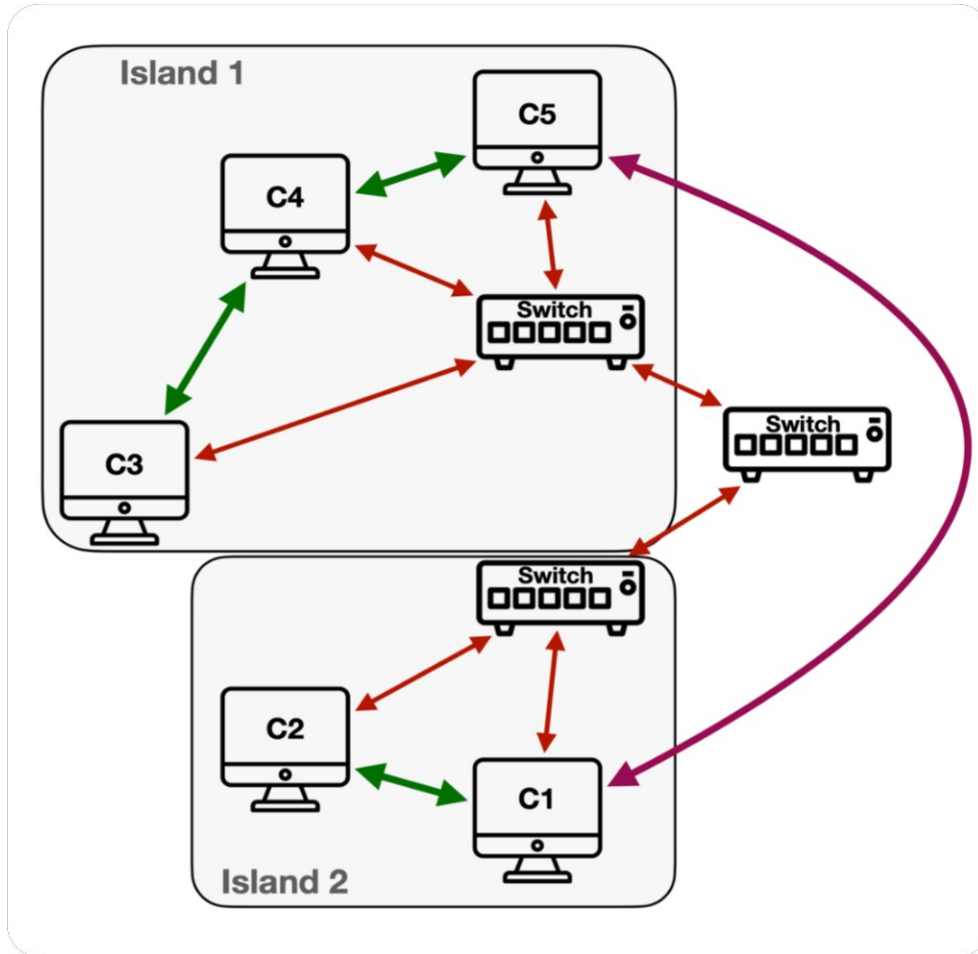
SURF

| Performance analysis - Communications

- Leads to a decrease in performance as the number of active processes increases
- Optimisation may require knowledge of the network structure
- Some frameworks allow for run-time tuning of the communication algorithms (e.g. with MPI tune)
- Communication cost depends on message sizes, topology and synchronization.

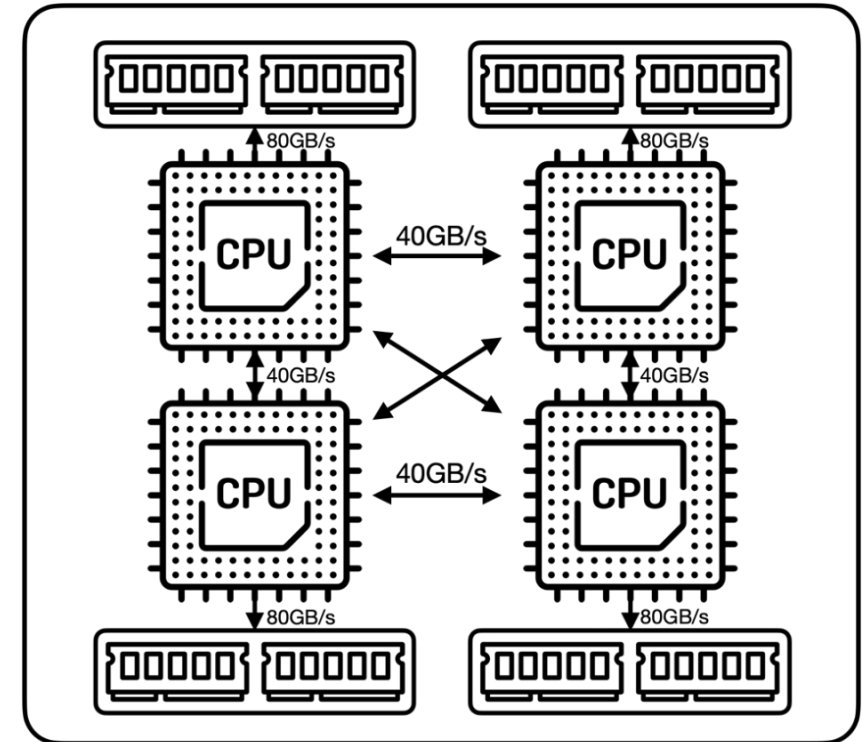
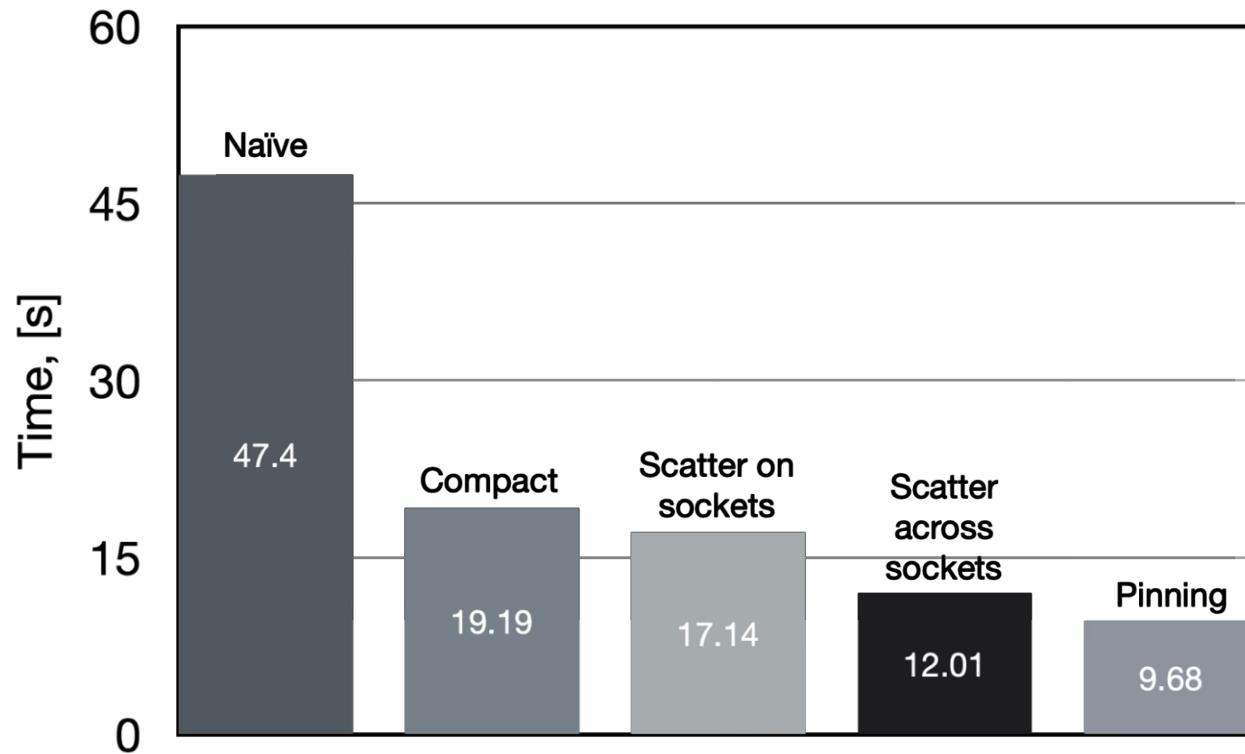


Performance analysis - Communications



Performance analysis - Communications

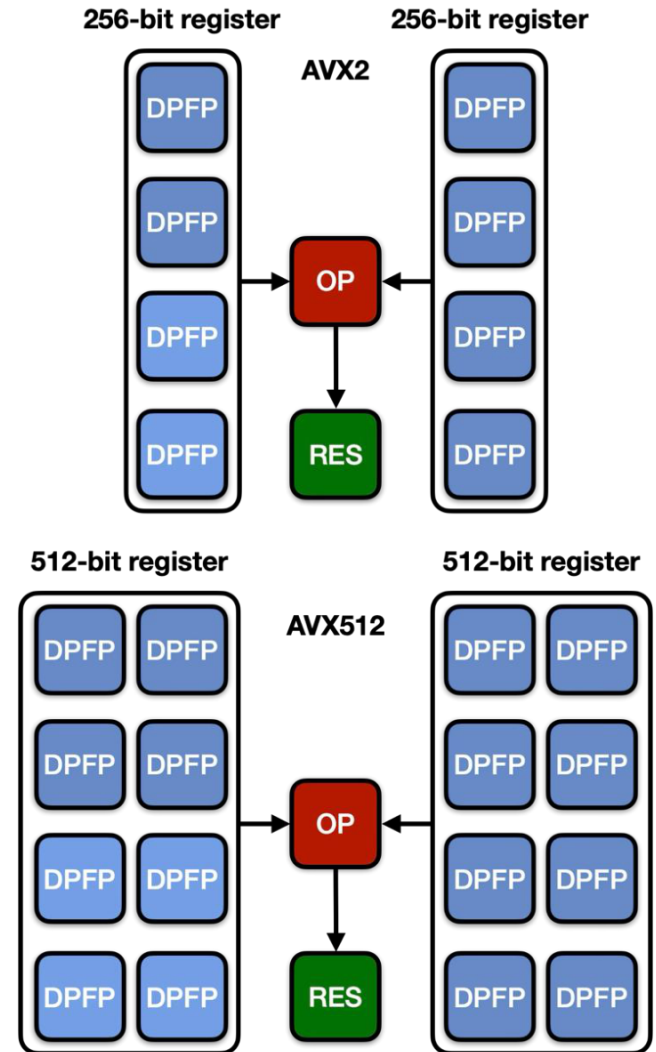
QX-simulator, C++, 32 threads



Performance analysis - Computations

- **Dozens** of packages that allow to analyse performance of sequential and parallel codes
- Different tools suit different parallelisation strategies (threaded, distributed, GPU)
- Many tools use **hardware counters** that **have to be enabled explicitly on Snellius**
- Ideal speedup is P for P processors at fixed total work.

```
#SBATCH --constraint=hwperf  
#SBATCH --exclusive
```

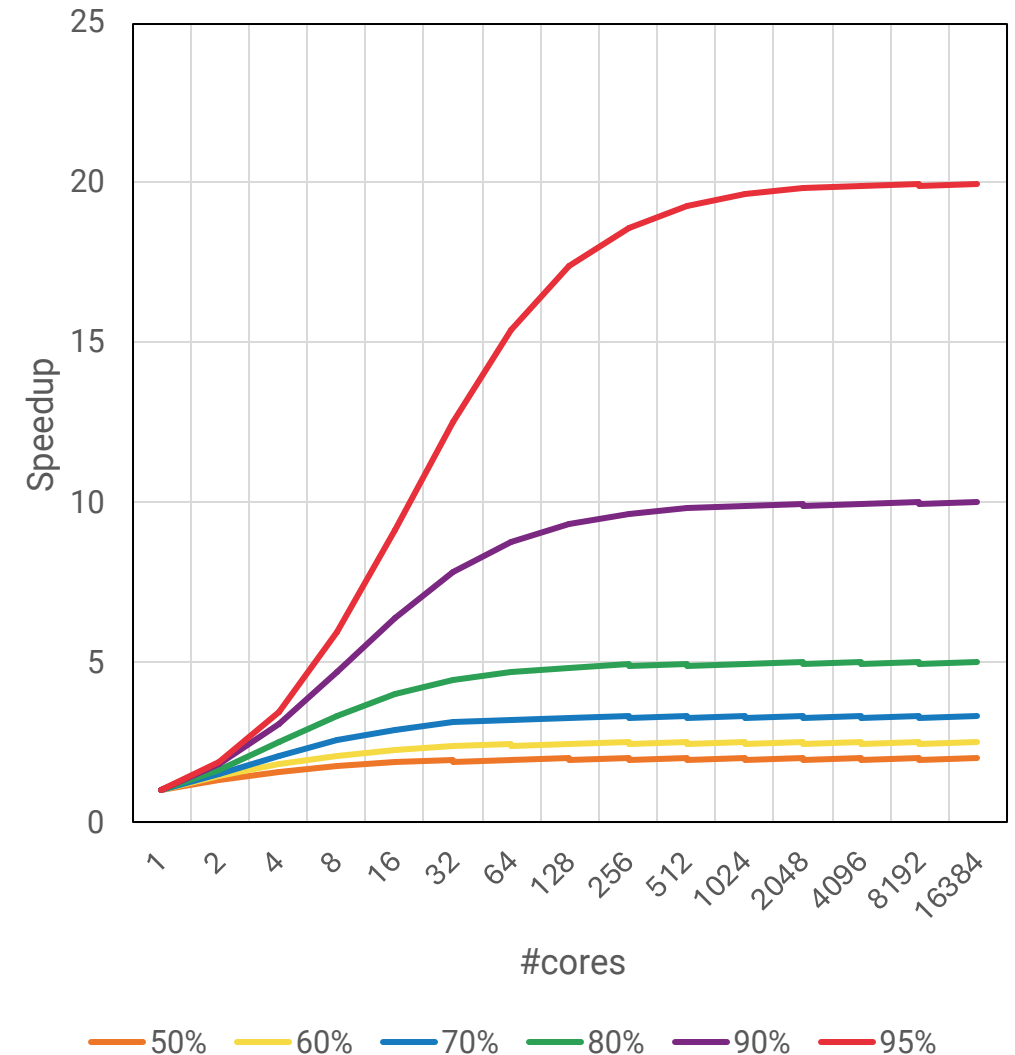


Strong and weak scaling

- **Strong scaling:** the problem size is **fixed**, the number of processors is **increased**
- **Amdahl's law** - the speedup is limited by the fraction of the serial part of the software that is not amenable to parallelization

$$speedup = \frac{1}{s + p/N}$$

- s: proportion of the execution time spent on the serial part
- p: the share spent on the parallel part
- N: number of processes

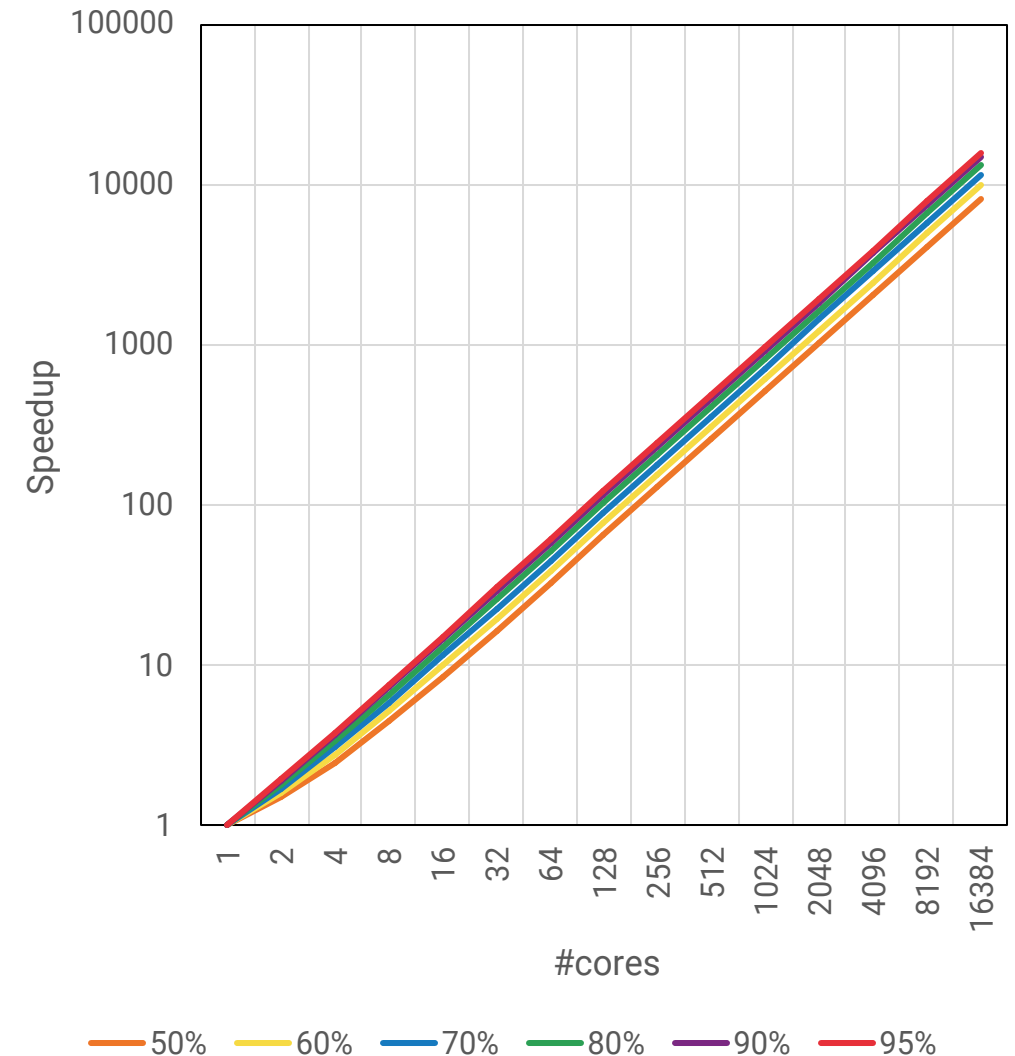


Strong and weak scaling

- **Weak scaling:** the problem size **increases** with the number of processes
- **Gustafson's law** - the parallel part scales linearly with the amount of resources, while the serial part stays fixed as the problem grows

$$speedup = s + p \cdot N$$

- s : proportion of the execution time spent on the serial part
- p : the share spent on the parallel part
- N : number of processes



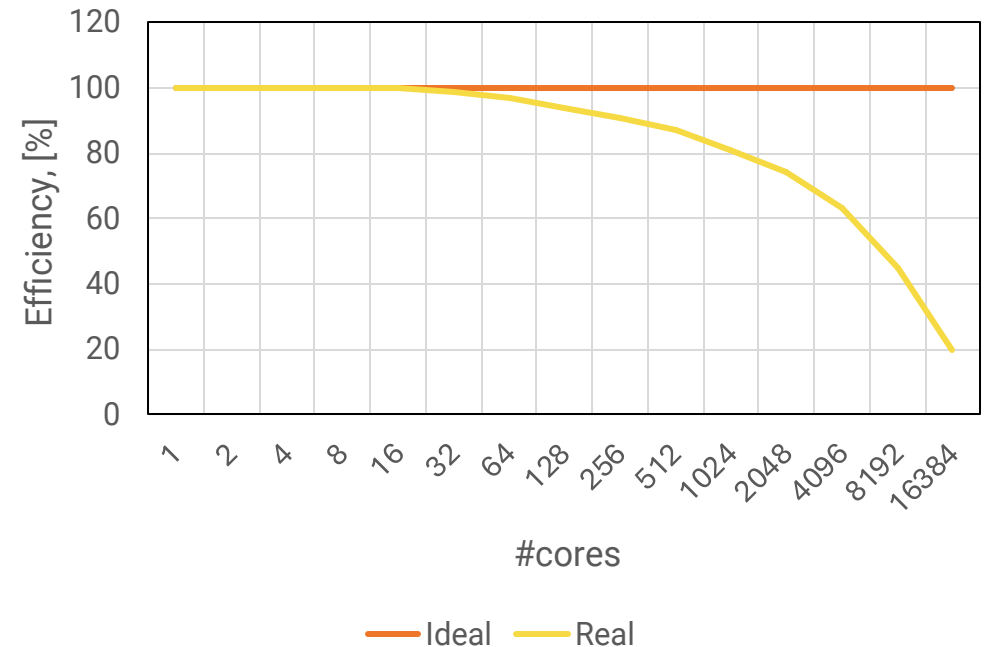
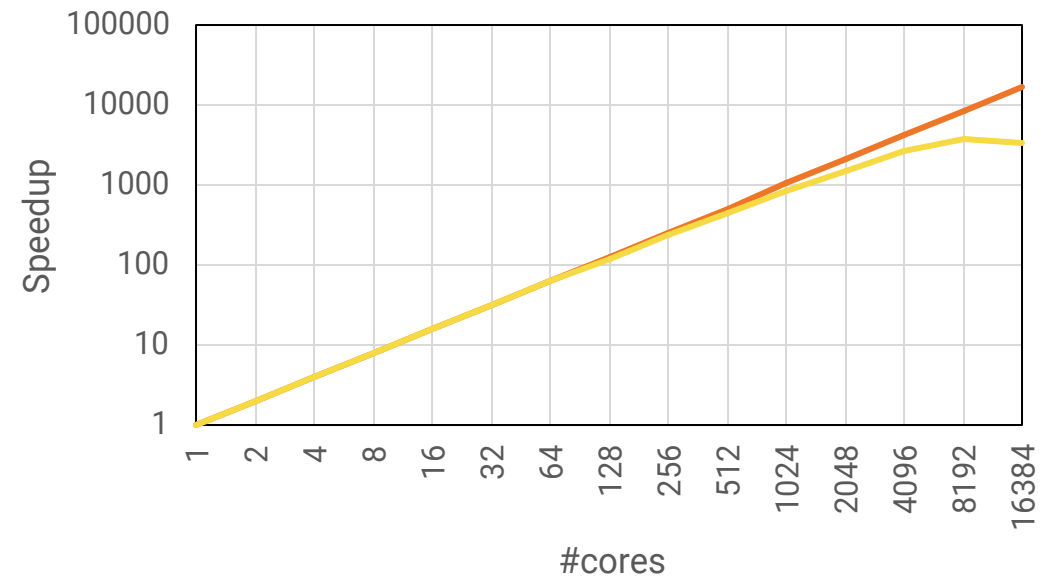
Strong and weak scaling

- Speedup and efficiency:

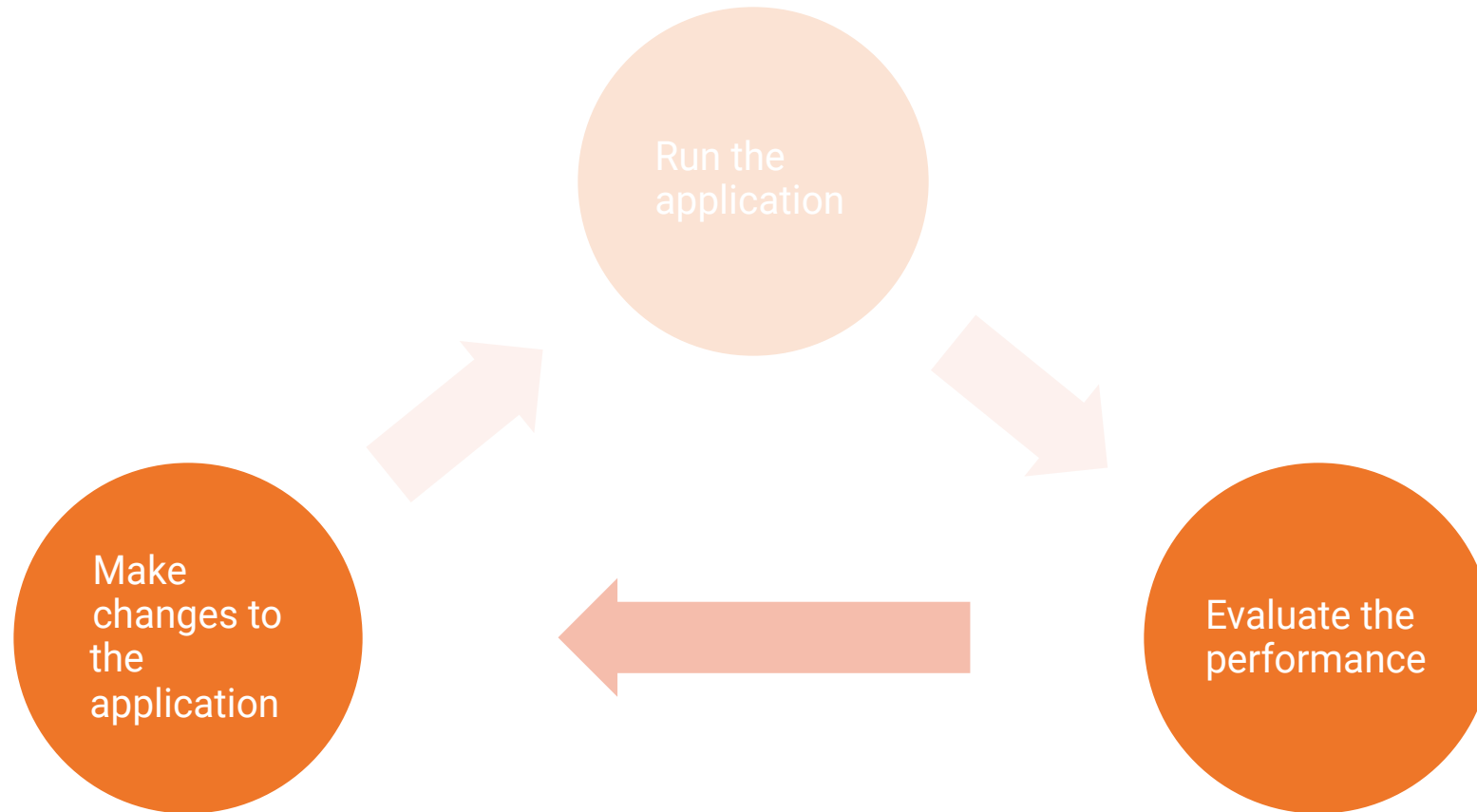
$$speedup = \frac{T_s}{T_N}$$

$$efficiency = \frac{1}{N} \frac{T_s}{T_N}$$

- T_s : the amount of time needed to complete a serial task
- T_N : the time a parallel task takes on N processes
- N : number of processes



| A life cycle of an HPC application

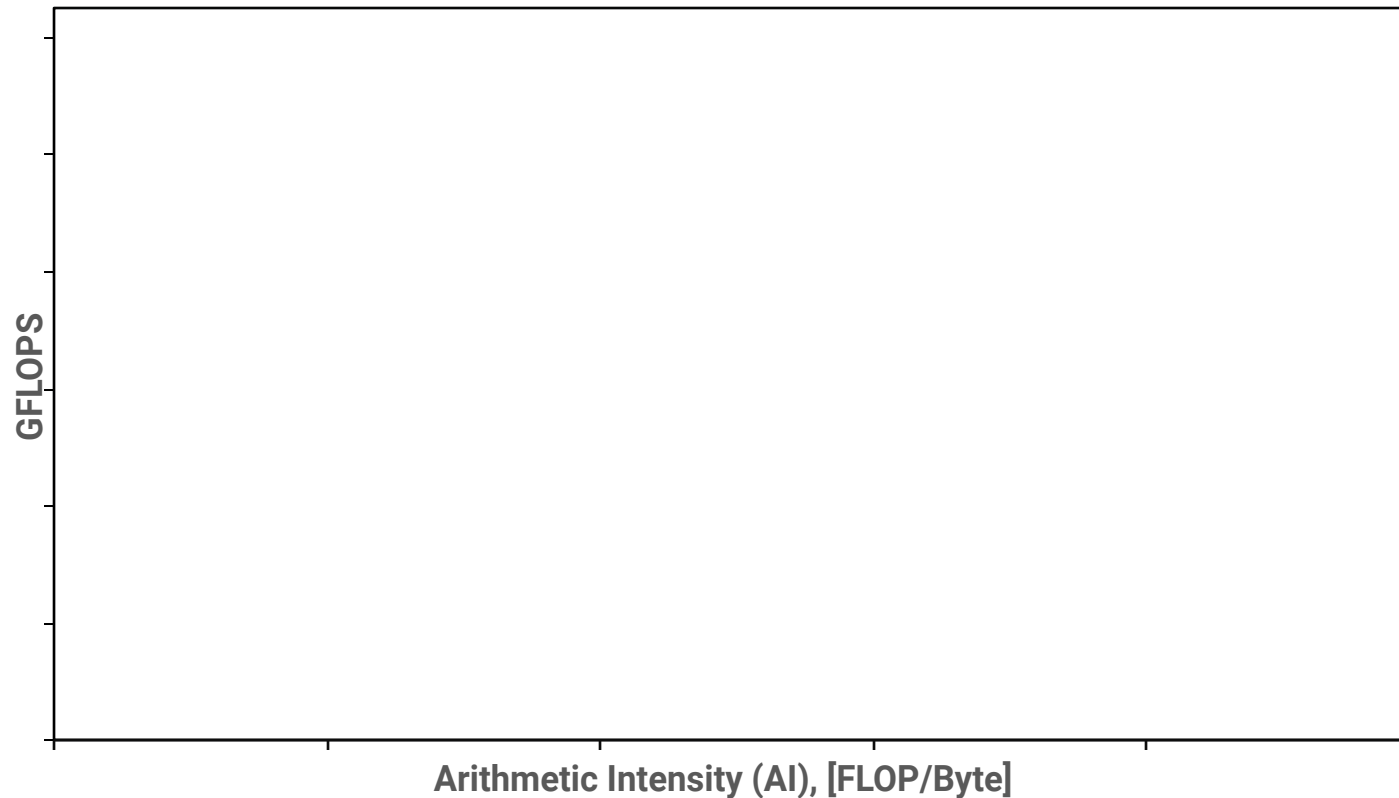


| Standard techniques

- Change the model/method to a faster one
- Change the backend (e.g. linear algebra) to a faster one
- Reduce the precision (e.g. from 64-bit FP to 32-bit FP)
- Optimise data structures (e.g. from AOS to SOA)
- Optimise IO operations
- Optimise data transfers (cache, network)
- Make application hardware-aware
- Use different hardware
- Optimise memory footprint
- Use different programming language

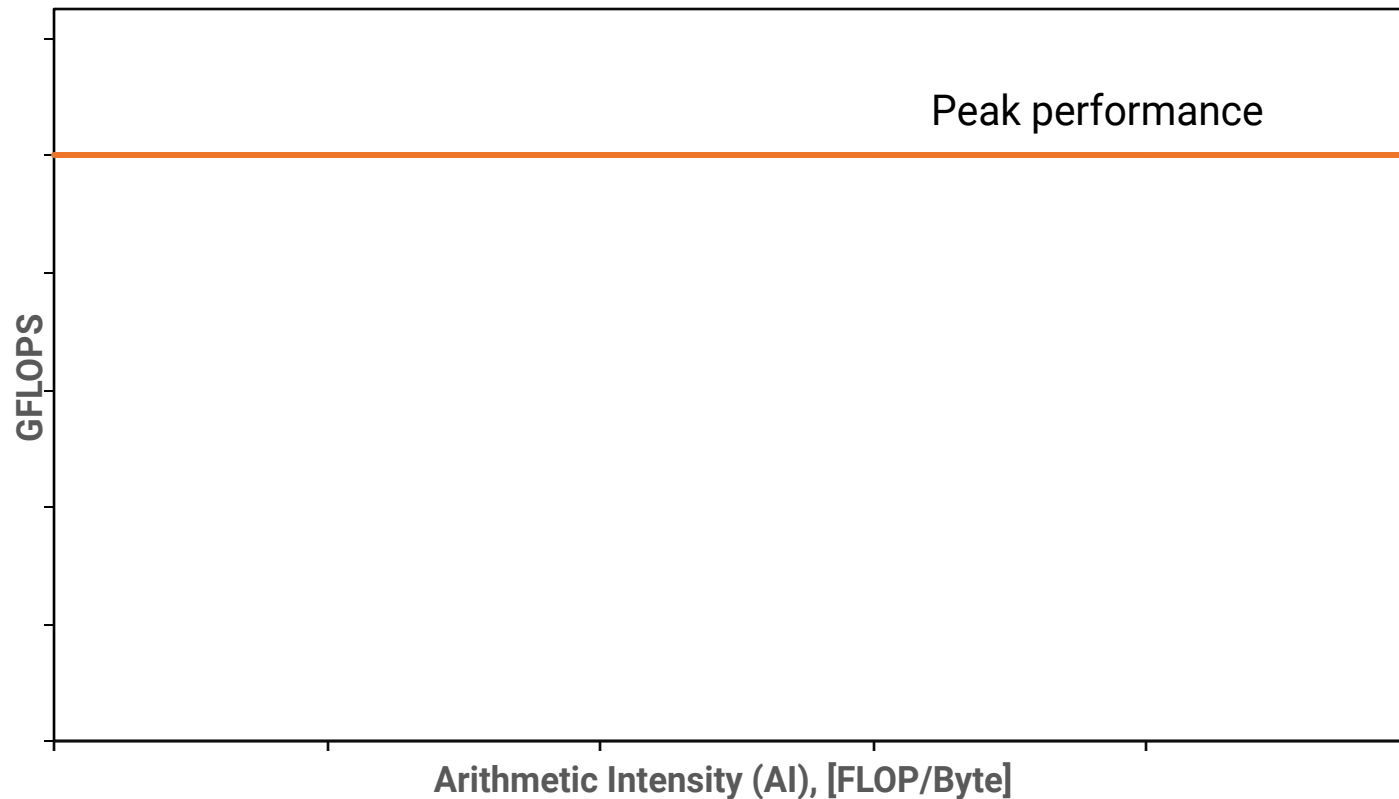
| Roofline model

- A visual model that shows the performance of an application or its kernel with respect to the hardware limitations.
- Shows potential benefits and priority of optimisations.



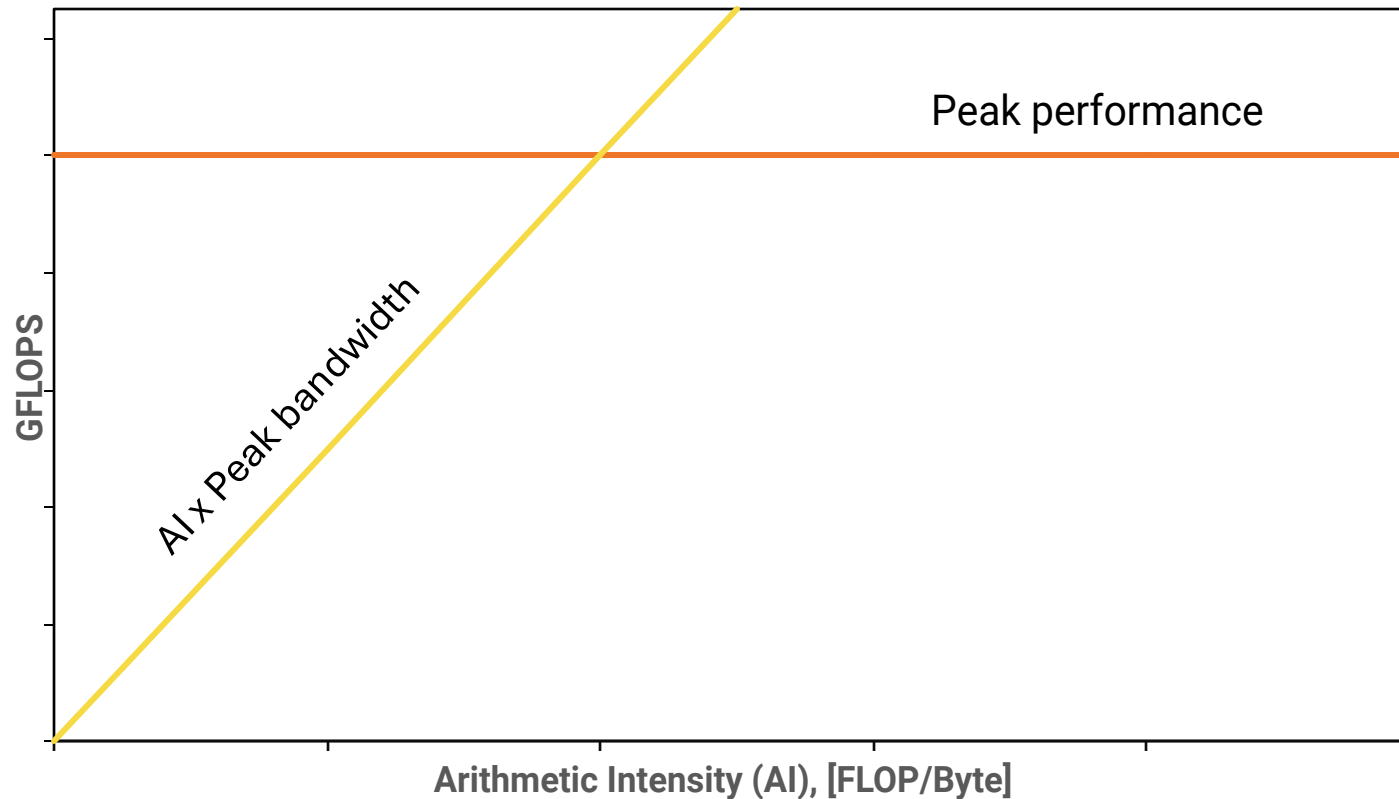
| Roofline model

- A visual model that shows the performance of an application or its kernel with respect to the hardware limitations.
- Shows potential benefits and priority of optimisations.



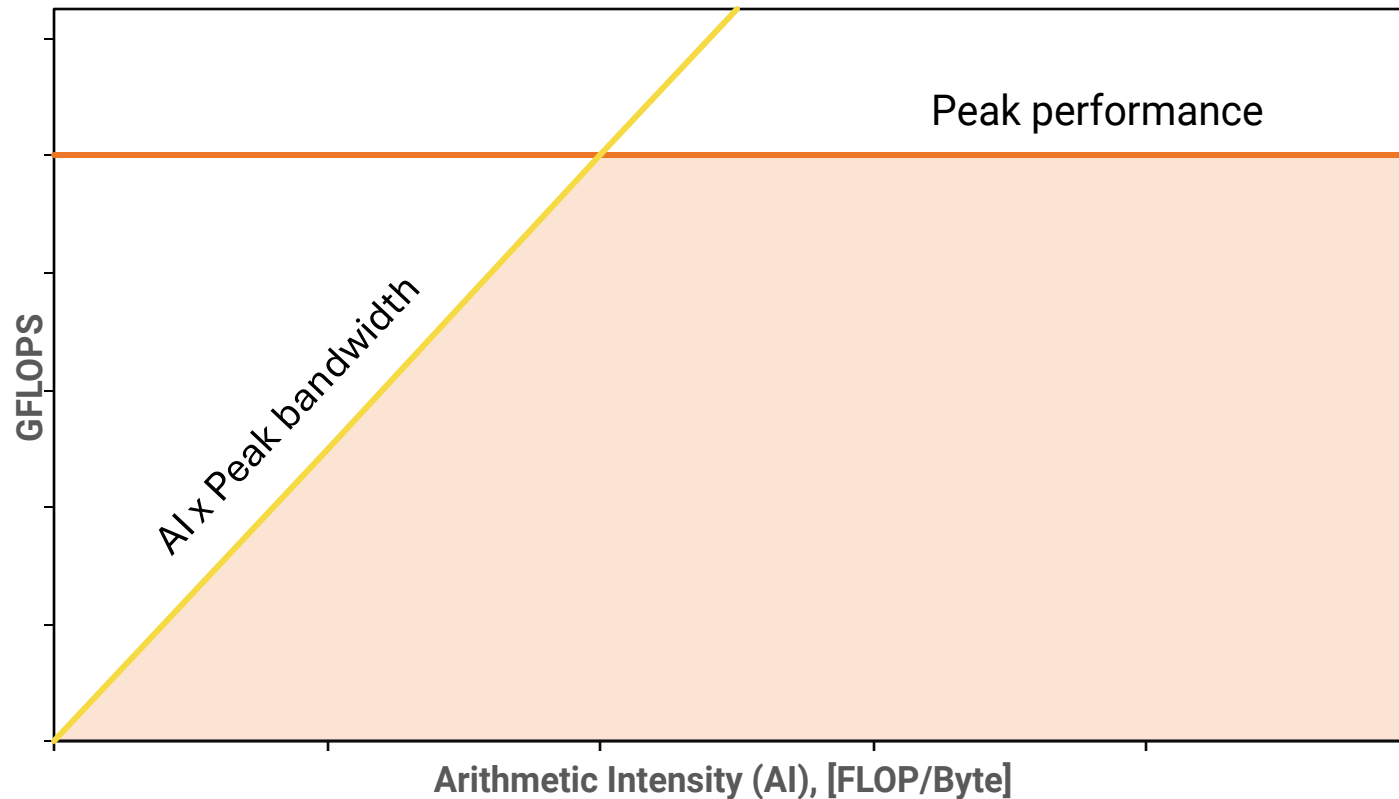
| Roofline model

- A visual model that shows the performance of an application or its kernel with respect to the hardware limitations.
- Shows potential benefits and priority of optimisations.



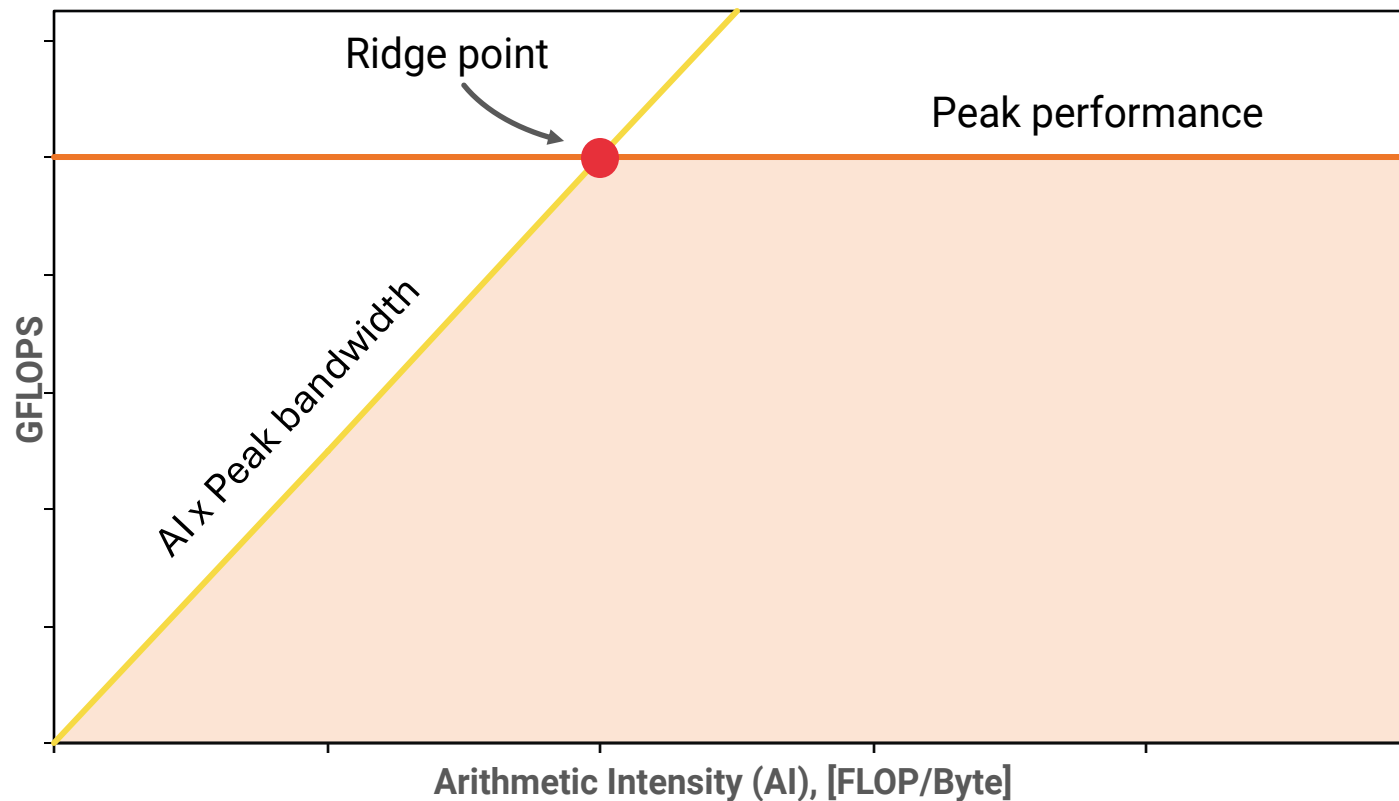
| Roofline model

- A visual model that shows the performance of an application or its kernel with respect to the hardware limitations.
- Shows potential benefits and priority of optimisations.



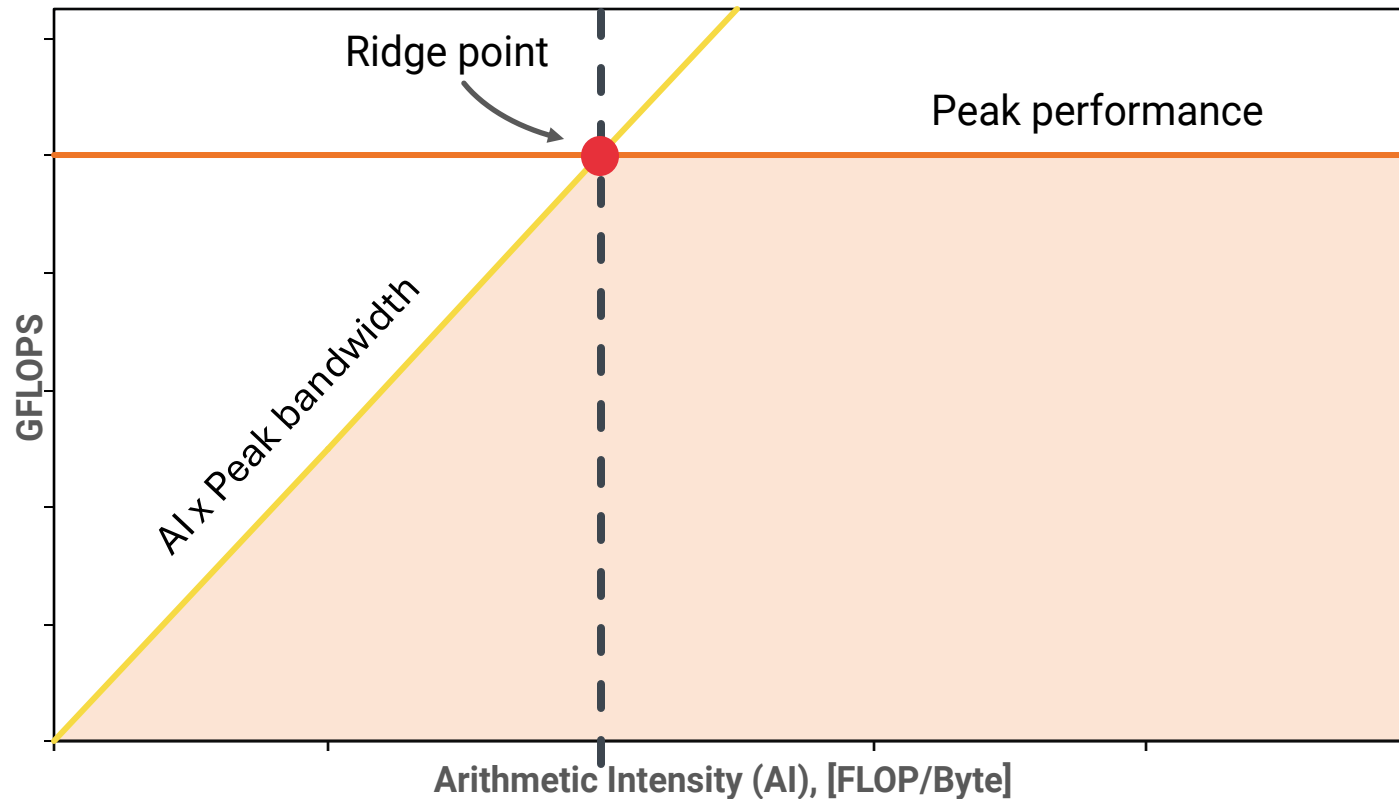
| Roofline model

- A visual model that shows the performance of an application or its kernel with respect to the hardware limitations.
- Shows potential benefits and priority of optimisations.



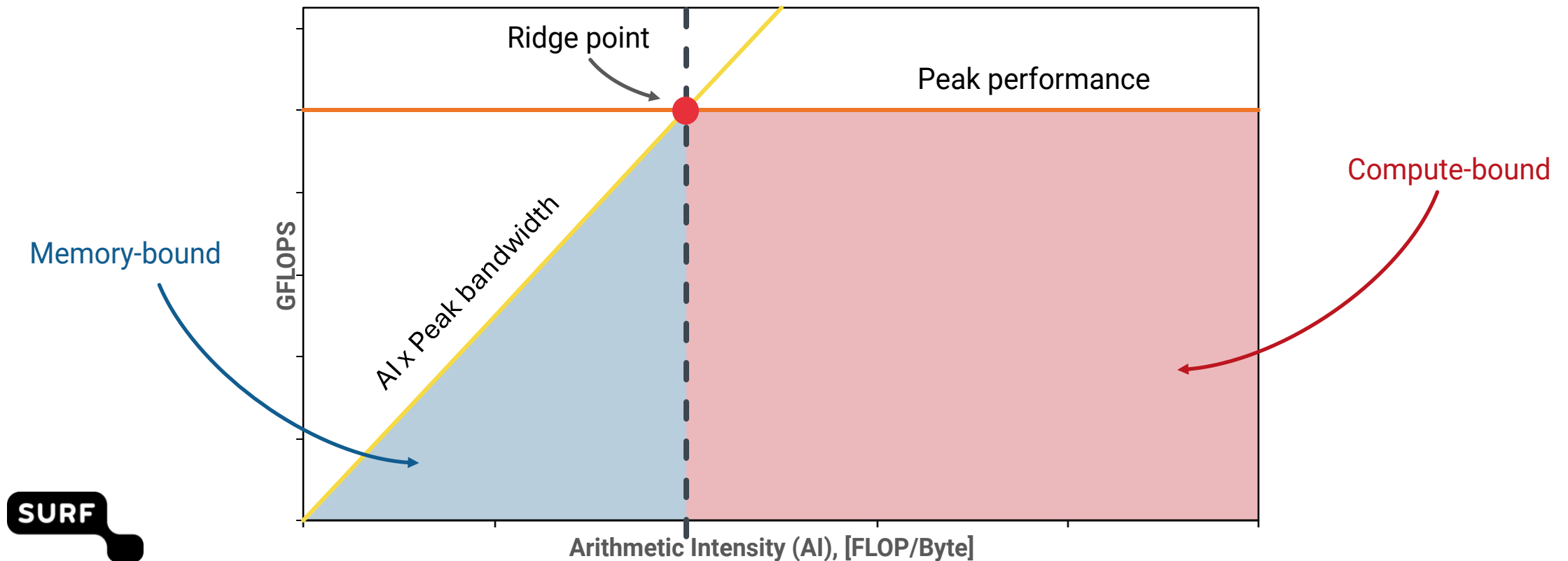
| Roofline model

- A visual model that shows the performance of an application or its kernel with respect to the hardware limitations.
- Shows potential benefits and priority of optimisations.



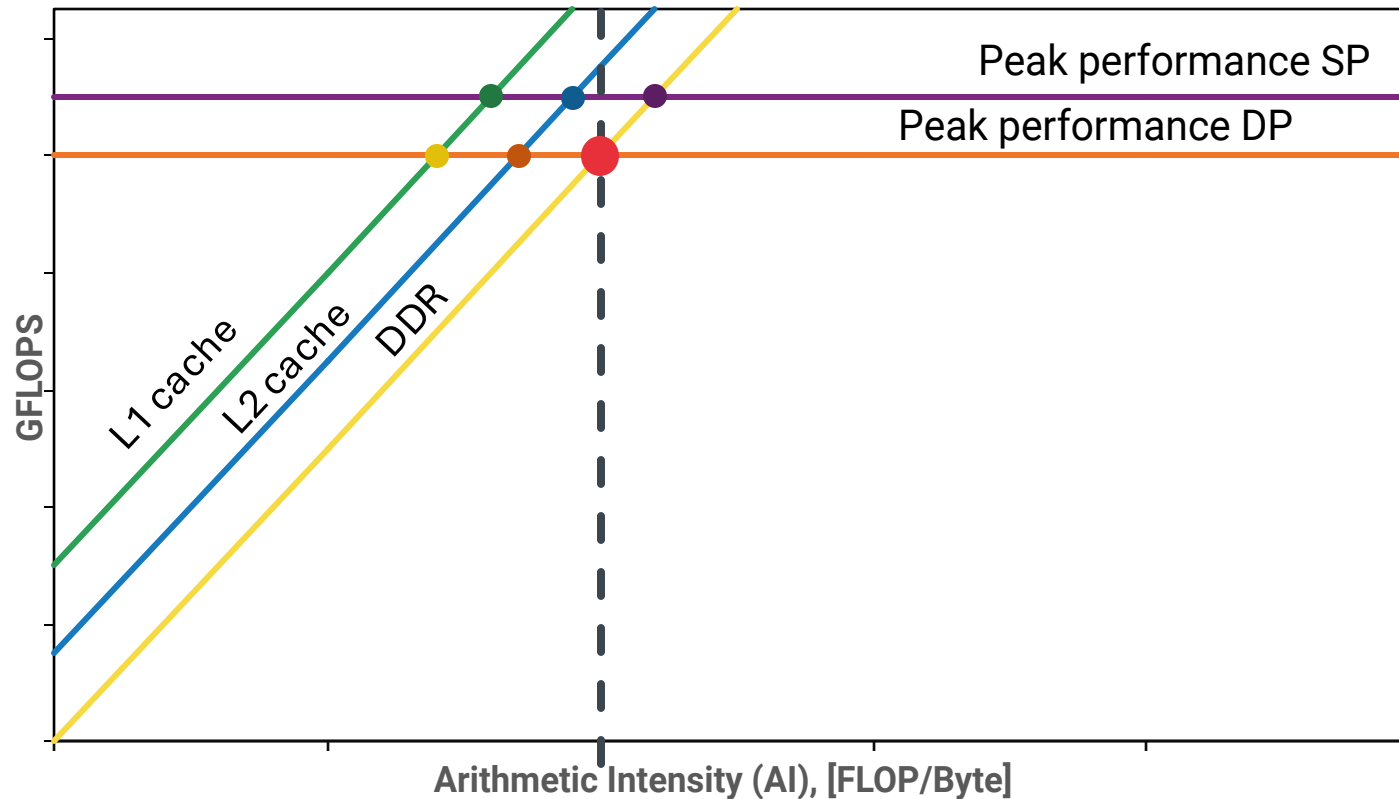
| Roofline model

- A visual model that shows the performance of an application or its kernel with respect to the hardware limitations.
- Shows potential benefits and priority of optimisations.



Roofline model

- A visual model that shows the performance of an application or its kernel with respect to the hardware limitations.
- Shows potential benefits and priority of optimisations.



| How to estimate FLOP counts?

- Manual FLOP estimation

```
for n in range(0, N):  
    z[n] = saxpy(a, x, y)
```

```
def saxpy(a, x, y):  
    return a * x + y
```



Addition: 1 FLOP
Multiplication: 1 FLOP

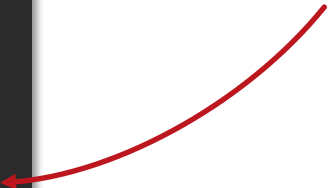
* $N = 2N \text{ FLOP}$

| How to estimate FLOP counts?

- Use knowledge of the algorithm complexity (if possible)
- For instance:
 - Generic dense matrix-matrix multiplication: $\approx O(N^3)$ FLOP
 - FFT algorithm: $\approx O(N \log_2(N))$ FLOP
- This approach often does not consider the total FLOP count of the algorithm

```
def gemm(matA, matB, matC, N):  
    for i in range(0, N):  
        for j in range(0, N):  
            for k in range(0, N):  
                matC[i][j] += matA[i][k] * matB[k][j]
```

$2N^3$ FLOP



| How to estimate FLOP counts?

- Now take a large code base of complex implementation.

```
def check_collisions_v4():
    tree = cKDTree(positions)
    pairs = np.array(list(tree.query_pairs(radii.max() * 2)))

    if len(pairs) > 0:
        i, j = pairs[:, 0], pairs[:, 1]
        r_rel = positions[i] - positions[j]
        v_rel = velocities[i] - velocities[j]
        # dists = np.linalg.norm(r_rel, axis=1)
        mask = np.einsum('ij,ij->i', v_rel, r_rel) < 0
        valid_pairs = pairs[mask]

        for i, j in valid_pairs:
            resolve_collision(i, j)
```

| How to estimate FLOP counts?

- Now take a large code base of complex implementation.
- **Use performance analysis tools!**

Tool name	Costs	Description
Linaro DDT	Non-free	Full featured graphical, parallel debugger
HPCToolkit	Free	Integrated suite of tools for parallel program performance analysis
Intel One API	Free, with conditions	Stack of different performance analysis and debugging tools (MPI/OpenMP/SIMD)
Valgrind	Free	Memory errors debugging tool
TotalView	Non-free	Full featured graphical, parallel debugger
Vampir	Non-free	Full featured trace visualizer for parallel program OTF trace files
memP	Free	Lightweight memory profiling tool
mpiP	Free	Lightweight MPI profiling tool
PAPI	Free	A standardized and portable API for accessing performance counter hardware
likwid	Free	A tool to measure hardware performance counters
Tau	Free	Full featured parallel program performance analysis toolkit
Extrae	Free	MPI/OpenMP profiler
Scalasca	Free	MPI+OpenMP analysis with Score-P (Score-P/9.2-gompi-2025a)
Darshan	Free	IO profiler
Nsight	Free	GPU: nsys (Nsight-Systems/2025.4.1), ncu (CUDA/12.8.0)
Linaro MAP	Non-free	performance analysis tool for MPI+OpenMP
uProf	Free	AMD CPU counters, sampling and roofline analysis
gprof	Free	Standard unix/linux profiling utility

| How to estimate FLOP counts?

- AMD-uProf:

```
$ module purge
$ module load 2025
$ module load AMD-uProf/5.1.701
$ AMDuProfPcm roofline -X -o res.csv -- <prog> <args>
```

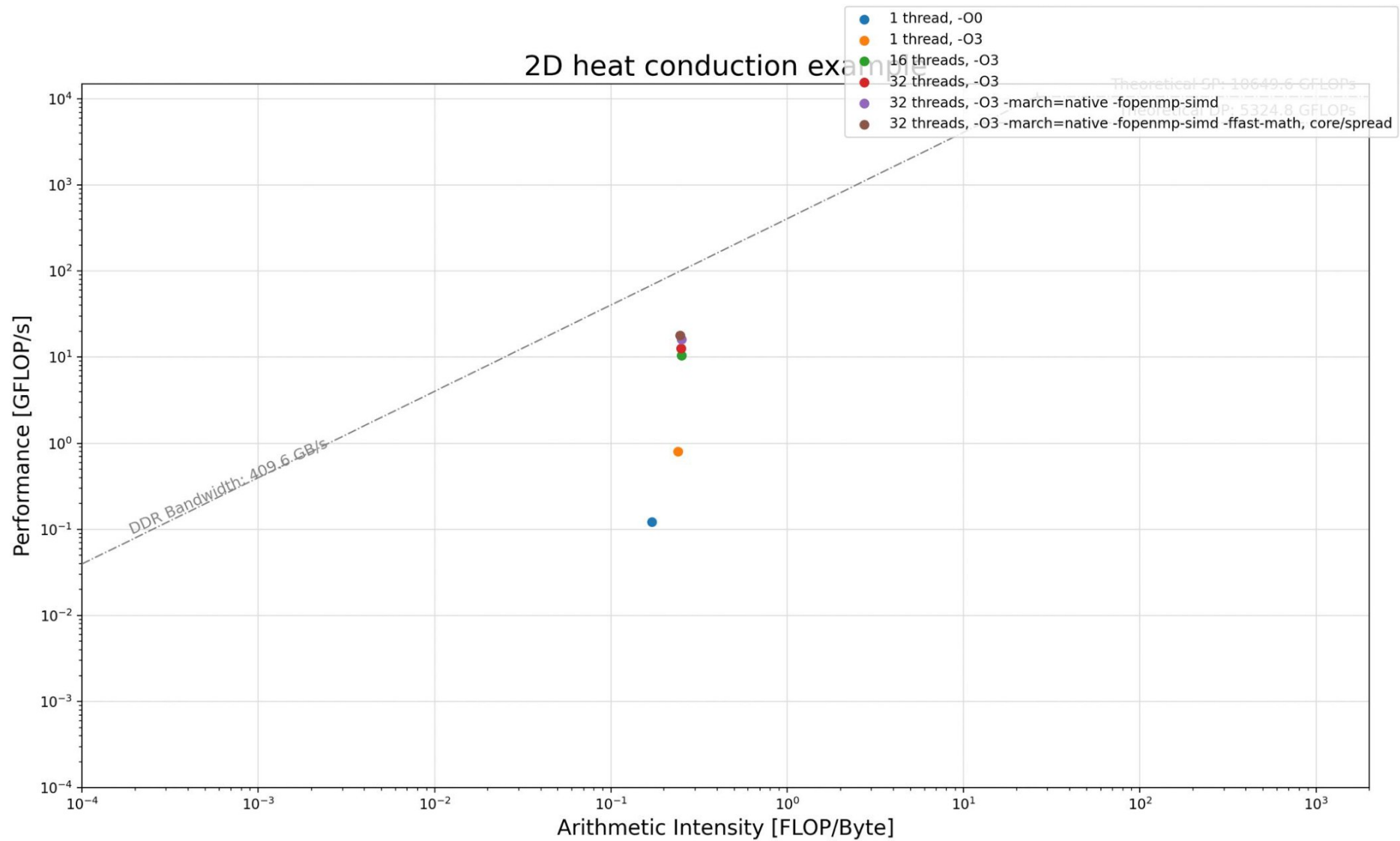
[Profile Data]
Application,./a.out
Type,DRAM
Name,./a.out
Throughput,6.7257
Bandwidth,2.6767

Output example

$$AI = \frac{\text{Throughput}}{\text{Bandwidth}}$$


- uProf reports Throughput in GFLOP/s and Bandwidth in GB/s, so this run has $AI = 6.7257 / 2.6767 = 2.5$ FLOP per byte. Compare it with the roofline ridge.

Roofline model



| How to estimate energy footprint?

- Add the following options to the job script and submit it:

```
#SBATCH --ear=on  
#SBATCH --ear-policy=monitoring  
  
export EARL_REPORT_LOOPS=1
```

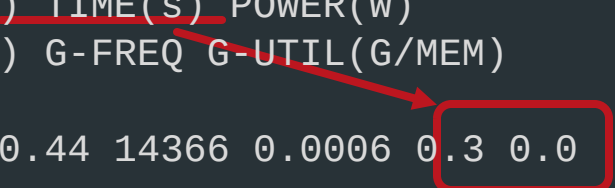
- Read more:
<https://servicedesk.surf.nl/wiki/spaces/WIKI/pages/62226671/Energy+Aware+Runtime+EAR>

| How to estimate energy footprint?

- EAR:

```
$ eacct -j <job-id>
```

```
JOB-STEP AID USER APPLICATION POLICY NODES AVG/DEF/IMC(GHz) TIME(s) POWER(W)
GBS      CPI ENERGY(J) GFLOPS/W IO(MBs) MPI% G-POW(T/U) G-FREQ G-UTIL(G/MEM)
11041163-0 2747012
maximm    job.sh M0 1 2.52/2.60/1.47 56.64 253.62 1.83 0.44 14366 0.0006 0.3 0.0
0.00/---  ---  ---
```



| SLURM



| SLURM

- **Nodes:**
 - **-N, --nodes:** request a number of nodes for a job
- **Tasks:**
 - **-n, --ntasks:** sbatch does not launch tasks, it requests an allocation of resources and submits a batch script. This option advises the Slurm controller that job steps run within the allocation will launch a maximum of *number* tasks and to provide for sufficient resources.
 - **--ntasks-per-node** - request that *ntasks* be invoked on each node.
- **CPUs**
 - **-c, --cpus-per-task** - Advise the Slurm controller that ensuing job steps will require *ncpus* number of processors per task. Without this option, the controller will just try to allocate one processor per task.

| SLURM

- **Sockets**

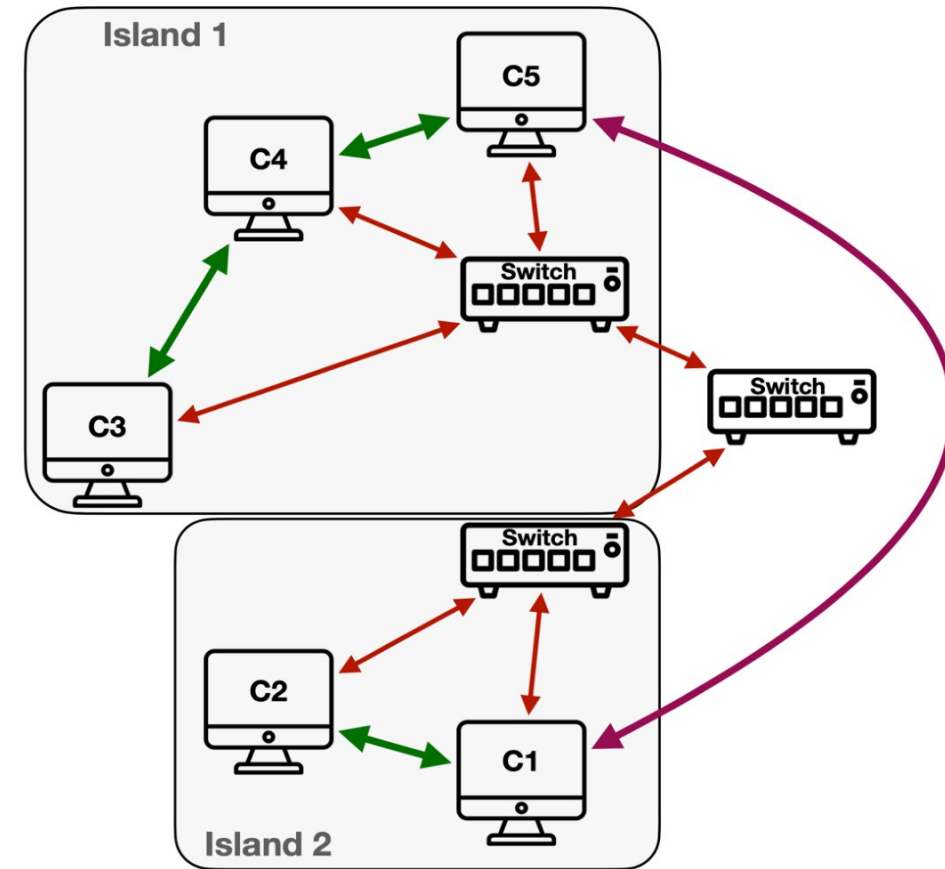
- **--sockets-per-node** - Restrict node selection to nodes with at least the specified number of sockets

- **GPUs**

- **--gpus-per-node** - Specify the number of GPUs required for the job on each node included in the job's resource allocation.

- **Allocations**

- **--contiguous** - If set, then the allocated nodes must form a contiguous set
- **--exclusive** - The job allocation can not share nodes with other running jobs



| SLURM

- **SLURM_JOB_NODELIST** - list of nodes allocated to the job.
- **SLURM_NTASKS** - set to value of the **--ntasks** option, if specified. With **--ntasks-per-node** or **--ntasks-per-gpu** instead, it is the number of tasks in the job.
- **SLURM_CPUS_PER_TASK** - number of cpus requested per task. Set if the **--cpus-per-task** option is specified.
- **SLURM_GPUS** - number of GPUs requested. Only set if the **-G, --gpus** option is specified.

| Abbreviations

- American Standard Code for Information Interchange (ASCII)
- Array of structures (AOS)
- Computational fluid dynamics (CFD)
- Distributed Debugging Tool (DDT)
- Energy Aware Runtime (EAR)
- Fast Fourier transform (FFT)
- Floating-point operation (FLOP)
- Linaro Forge profiler (MAP)
- Message Passing Interface (MPI)
- Non-Volatile Memory Express (NVME)
- Open Trace Format (OTF)
- Performance Application Programming Interface (PAPI)
- Reduced instruction set computer (RISC)
- Single instruction multiple data (SIMD)
- Structure of arrays (SOA)
- Visual Molecular Dynamics (VMD)