

# Profiling Basics

Understanding Metrics and Tools for HPC

Chris Stylianou  
Research Engineer

CaSToRC  
[eurocc.cyi.ac.cy](http://eurocc.cyi.ac.cy)



## A Bit About Myself



- Research Engineer at CaSToRC, PhD
- *Training and Skills Development* Task Leader for EuroCC2
- Area of expertise in High Performance Computing (HPC)
- HPC x AI
- Contact & Info:
  - Email: [c.stylianou@cyi.ac.cy](mailto:c.stylianou@cyi.ac.cy)
  - Website: [cstyl.github.io](http://cstyl.github.io)





## Objectives



Introduce profiling in HPC.



Discuss metrics for performance evaluation.



Explore workflows and tools for performance optimization.



**Definition:** Profiling is the process of analysing an **application's performance** to identify **bottlenecks**.

### **Importance:**

- Optimize computational workloads.
- Enhance scalability and efficiency.
- Reduce time-to-solution.

### **Types:**

- **Performance Profiling:** Tracks computational and communication performance.
- **Memory Profiling:** Monitors memory usage and leaks.
- **I/O Profiling:** Measures efficiency in data read/write operations.



## 80/20 Rule

- Programs typically spend **80% of their time in 20% of the code**
- Programmers typically spend **20% of their effort to get 80% of the total speedup**
  - stop before over optimization
- **Tip:** *Don't optimize parts of the code which does not matter and **start optimization after you understand your performance bottlenecks.***

Two independent parts    **A**   **B**

Original process



Make   **B**   5x faster



Make   **A**   2x faster





## Metrics and Tools for Serial Code Profiling



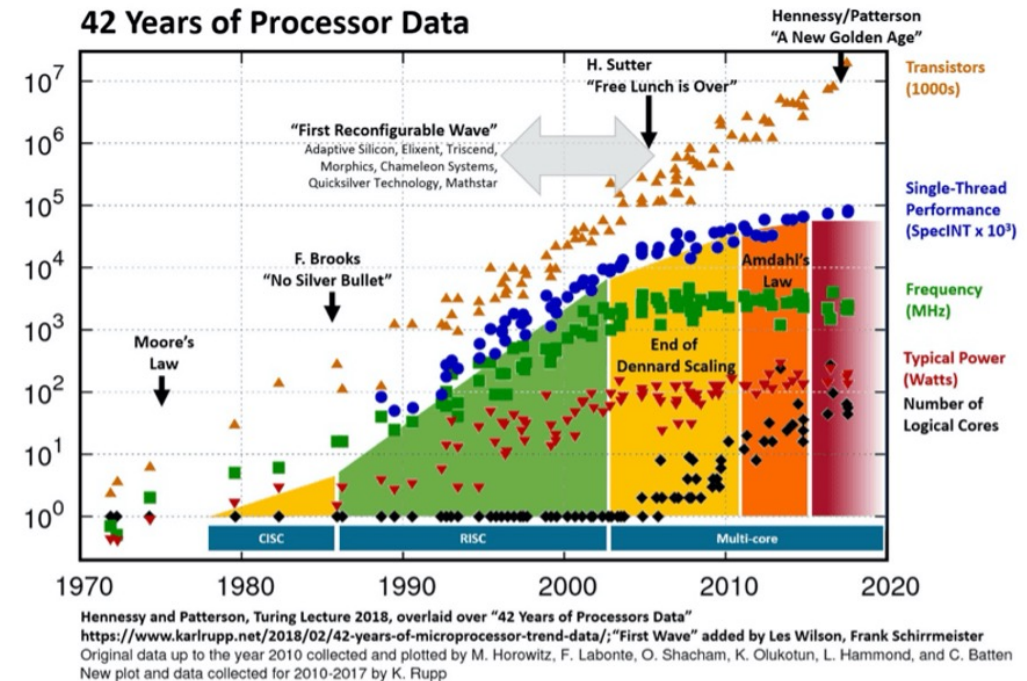
Metric	Description
<b>Execution Time</b>	Measure and rank time spent in each function.
<b>Instructions Per Cycle (IPC)</b>	Indicates efficiency of CPU usage.
<b>Cache Hit/Miss Ratios</b>	Impacts memory-bound workloads.
<b>Branch Prediction Accuracy</b>	Affects conditional-heavy code.
<b>I/O Wait Time</b>	Measures inefficiency in disk operations.

Tool	Description
<b>gprof</b>	GNU Profiler to determine which parts of a program take most of the execution time.
<b>Perf</b>	Linux profiling with performance counters and can instrument CPU performance counters, tracepoints, kprobes, and uprobes (dynamic tracing).
<b>Intel VTune</b>	x86 profiler by Intel.



## From Serial to Parallel

- Clock rates no longer increase – *free lunch is over!*
- **Performance** gain mainly **through parallelism**
- Optimization of applications becomes challenging
  - Increasing **application complexity**
    - multi-physics and multi-scale
  - Increasing **machine complexity**
    - hierarchical networks/ memory
    - more CPUs/multi-cores/accelerators
- Parallelization is limited by overheads
  - non-parallelized regions, I/O, communications, dependencies.
- Theoretical Speedup is **limited by the serial part** of the application (*Amdahl's Law*)





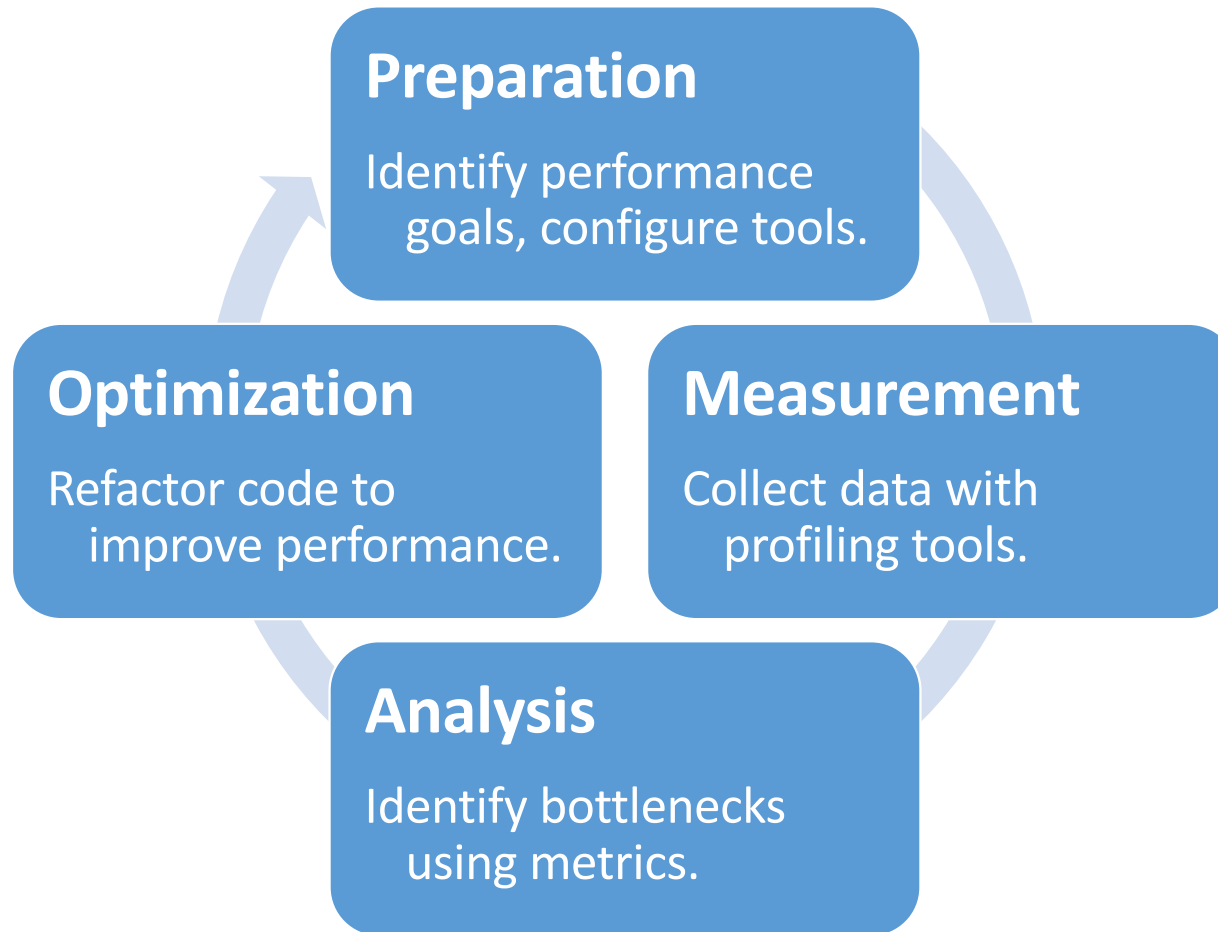
### **Sequential performance factors**

- Computation
  - choose right algorithm, use optimizing compiler
- Cache and memory
  - difficult, there is only limited tool support (compiler level)
- Input/Output

### **Parallel performance factors**

- Partitioning/decomposition
- Communication
- Multi-threading
- Synchronization







- **Global Efficiency (GE):** Measures overall performance.
  1. **Parallel Efficiency (PE):**
    - **Load Balance Efficiency (LB):** Work distribution across processes.
    - **Communication Efficiency (CommE):** Time lost in communication.
  2. **Computation Efficiency (CompE):**
    - Instruction scaling.
    - IPC scaling.



- The Global Efficiency describes **how well the parallelization of your application is working.**
- The Global Efficiency can be split into **Parallel Efficiency (PE)** and **Computation Efficiency (CompE).**

$$GE = PE * CompE$$



- The Parallel Efficiency describes **how well the execution of the code in parallel is working.**
- The Parallel Efficiency can be split into **Load Balance Efficiency (LB)** and **Communication Efficiency (CommE).**

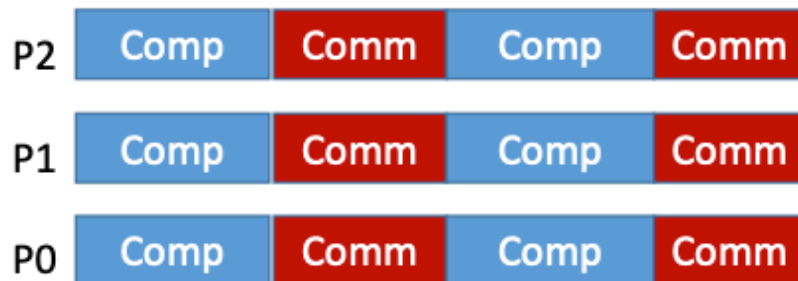
$$PE = LB * CommE$$



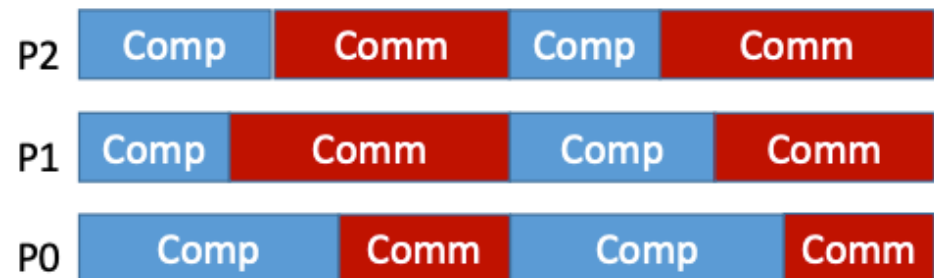
- The **Load Balance Efficiency (LB)** reflects **how well the distribution of work to processes of threads is done** in the application.
- The Load Balance Efficiency is the ratio between the **average time of a process** spend in computation and the **maximum time a process** spends in computation.

$$LB = \frac{avg(t_{comp})}{max(t_{comp})}$$

Example 1: good load balance (LB = 100%)



Example 2: bad load balance (LB = 77%)





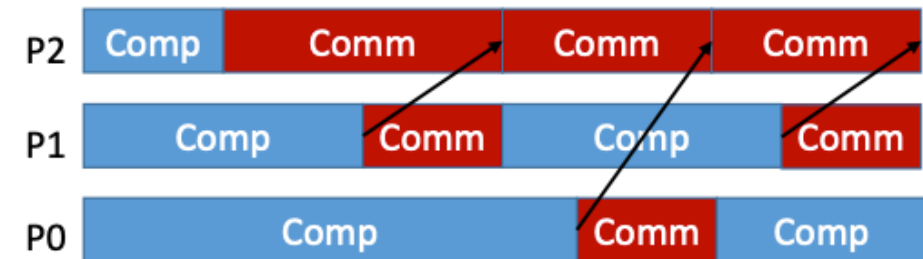
- The **Communication Efficiency (CommE)** reflects the **loss of efficiency by communication**.
- The Communication Efficiency can be computed as:

$$CommE = \max_{procs} \left( \frac{T_{computation}}{T_{total}} \right)$$

- The Communication Efficiency can be split further into **Serialization Efficiency (SerE)** and **Transfer Efficiency (TE)**.

$$CommE = SerE * TE$$

Example:



Compute	Communication	Efficiency
1 sec.	5 sec.	1/6
4 sec.	2 sec.	4/6
5 sec.	1 sec.	5/6

$$CommE = \frac{5}{6} = 83\%$$




- The **Serialization Efficiency (SerE)** describes **loss of efficiency due to dependencies between processes**.
- Dependencies can be observed as **waiting time** in MPI calls where no data is transferred, because one required process did not arrive at the communication call yet.
- On an ideal network with **instantaneous data transfer** these inefficiencies are still present, as no real data transfer happens.
- Serialization Efficiency is computed as

$$SerE = \max_{procs} \left( \frac{T_{computation}(ideal\ network)}{T_{total}(ideal\ network)} \right)$$

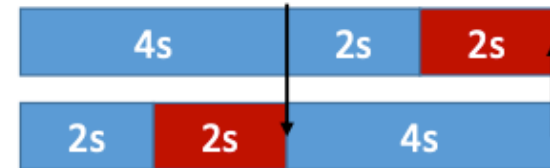
Execution on a real network



 = Computation



Simulation on an ideal network



 = Communication

$$SerE = \frac{6}{8} = 75\%$$



## Transfer Efficiency (TE)




- The **Transfer Efficiency (TE)** describes **loss of efficiency due to actual data transfer**.
- The Transfer Efficiency can be computed as:

$$TE = \frac{T_{total}(ideal\ network)}{T_{measured}}$$

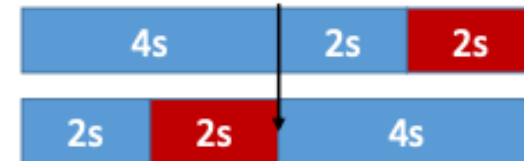
Execution on a real network



 = Computation



Simulation on an ideal network



 = Communication

$$TE = \frac{8}{12} = 66.6\%$$





- The **Computation Efficiency (CompE)** describes **how well the computational load of an application scales with the number of processes.**
- The Computation Efficiency is computed by **comparing the total time spend in computation for a different number of threads/processes.**
- For a **linearly-scaling application** the total time spend in computation is **constant** and thus, the **Computation efficiency is one.**
- The Computation Efficiency can be characterised by **IPC (Instructions Per Cycle)** and **Instruction Scaling.**











A low computation efficiency can have two reasons:

- 1. With more processes more instructions are executed**, e.g. some extra computation for the domain decomposition is needed.
  - Instruction Scaling compares the **total number of instructions executed for a different number of threads/processes**.
- 2. The same number of instructions is computed but the computation takes more time**, this can happen e.g. due to shared resources like memory channels.
  - IPC Scaling compares **how many instructions per cycle are executed for a different number of threads/processes**.



## Performance Analysis – A Recipe



	Background	Documentation for reproducibility of data and simulations, which will be important for comparison
	Application Structure	Give an overview of the application structure
	Region of Interest	Identify and isolate region of interest, e.g. with heavy workload or I/O communications
	Scalability Information	First performance property to investigate
	Application Efficiency	How much time is spent for essential computation and check typical issues like communicational overheads or imbalances
	Load Balance	Investigate how the work is distributed among workers
	Serial Performance	Investigation of regions without communication, use metrics like IPC to get hints if the hardware is used efficiently
	Communication	Investigate the communication overheads



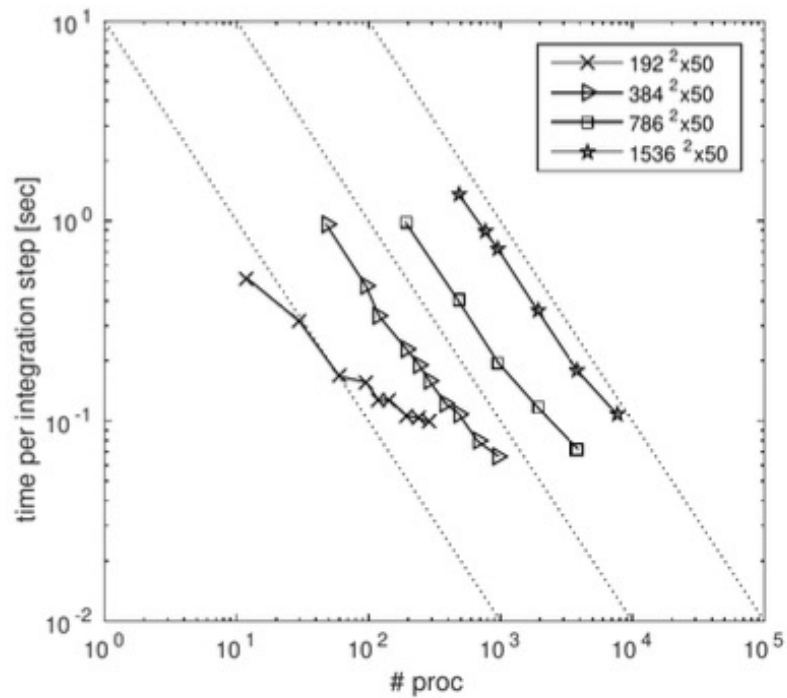
- Weather Research and Forecast Model [<https://www2.mmm.ucar.edu/wrf/users/>]
  - WRF **4.0** is parallelized using **domain decomposition** techniques, where communication between domains are done via **MPI**, each domain can be **parallelized using OpenMP**
  - Grid-sizes which include volumes from **V = 192x192x50** to **V = 1536x1536x50**, which have output, **files from 300MB to 2.2GB**
  - **Test-case:** weather forecasting
    - Morrison double-moment scheme with the RRTMG (Rapid Radiative Transfer Model) scheme for long- and shortwave radiation
    - Integration is done by a Runge Kutta scheme of 3rd order and resolution are chosen with respect to the resolution of the physical box
- Region of Interest:
  - **computational kernel:** due to domain decomposition, the computational load per domain becomes small
  - **I/O kernels:** after each hour, a configuration is written to the disc



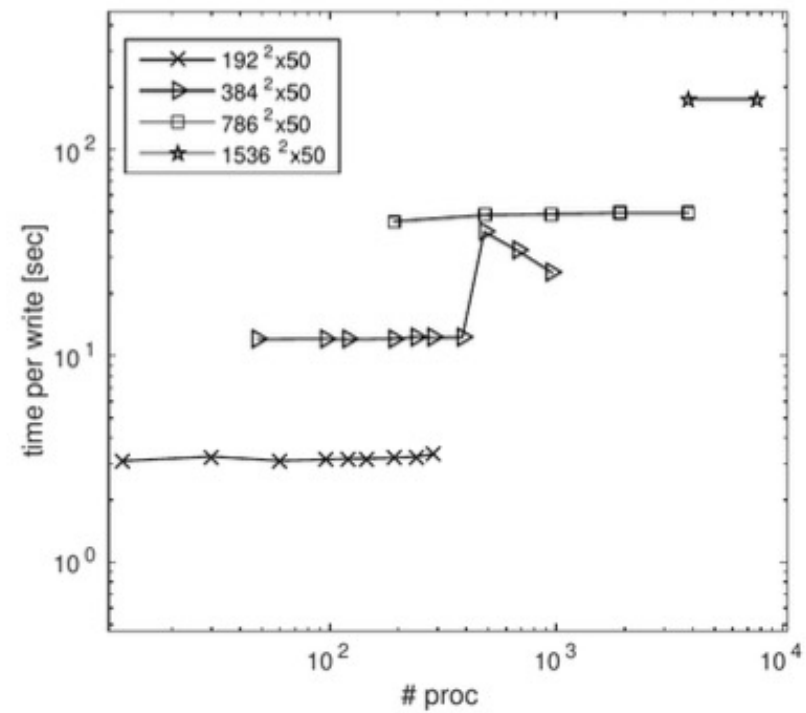
- After Region of Interests are identified
  - Investigate the **overall scalability** of the application
  - Describe **how the runtime of the application changes** if the **number of cores is changing**
  - Running with different number of processes the scalability analysis will give us a **first hint on parallel overhead**
  - Analyse the scalability for the whole application and possible regions of interest using timers or profilers
- Take notes of all running parameter so that it is possible to update later
  - how many cores, how many nodes are used, after optimization etc. and you don't want to re-do all that stuff



### Strong scaling of Computational Kernel

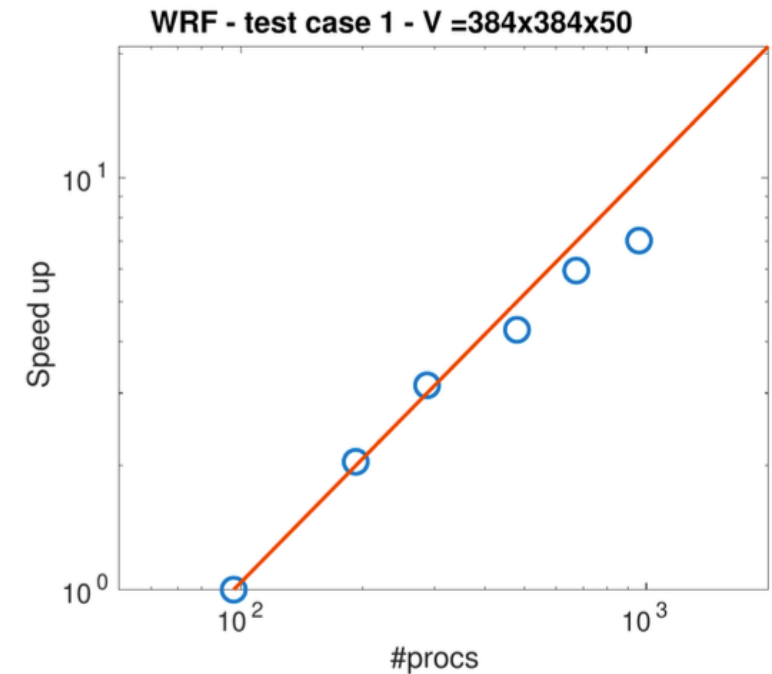


### Strong Scaling of I/O





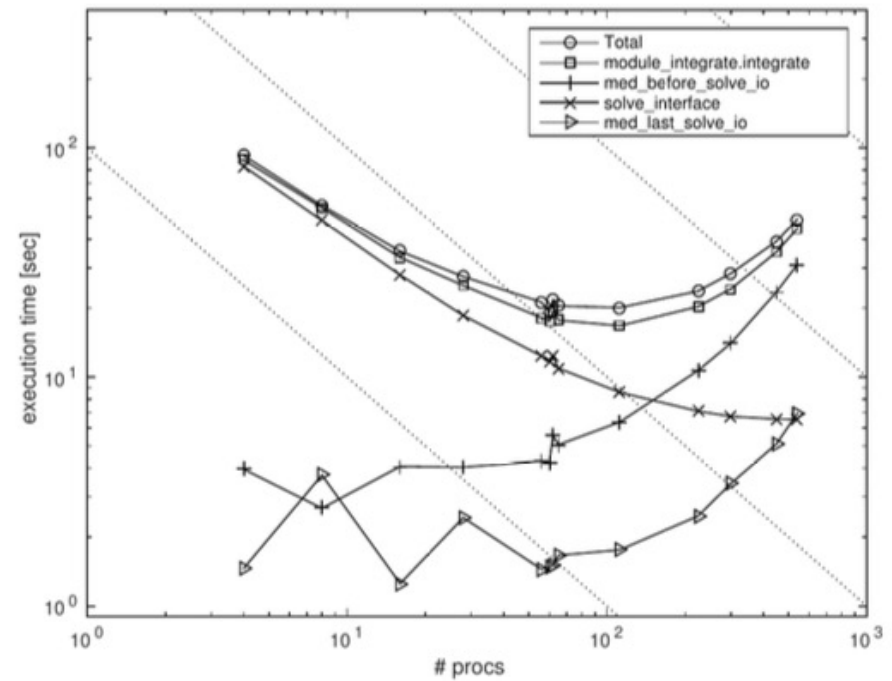
- Look at the **performance gain** through parallelization
- **Speed up** plot: *reference time / actual time*
- Very easy to differentiate the scaling behaviour of different regions.
- Keep the **reference time** (it is lost within the speed up plot) or do an efficiency plot
- Speed up shown with respect to linear speed up
- For example, **95% efficiency it follows 5% overhead due to parallelization**
  - Discuss the performance changes with increasing core counts





## Example – WRF: Limiting Factors

- Using profilers: V=64x45x30
- I/O is one limit in scalability
- Domains becomes too small







### From BSC:

- Extrae: generate tracing files  
[<https://tools.bsc.es/extrae>]
- Paraver: a flexible performance analysis tool  
[<https://tools.bsc.es/paraver/>]
- Dimemas: predict parallel performance using a single CPU machine  
[<https://tools.bsc.es/dimemas>]
- Darshan: a scalable HPC I/O characterization tool  
[<https://www.mcs.anl.gov/research/projects/darshan/>]

### From JSC:

- Score-P: scalable performance measure infrastructure for parallel codes  
[<https://www.vi-hps.org/projects/score-p/>]
- Cube: presentation component suitable for displaying a wide variety of performance data  
[<https://www.scalasca.org/software/cube-4.x/documentation.html>]
- Scalasca: collection of trace-based performance analysis tool  
[<https://www.scalasca.org/>]

Tutorials, Videos and  
Material available at:  
<https://pop-coe.eu>



### **Before Optimization:**

- Identify major computation kernels
- Understand scalability and bottlenecks
  - different bottlenecks results in different solutions
- To get a better overview beyond scaling and timers, we can use profilers
- What are your performance bottlenecks ?
  - Parallelization ? Scalability ? I/O ?
- Do you have plans for Optimization ?
  - of parallelization ?
  - computational kernels ? using libraries ?
  - porting them to GPUs ?



Thank you for your attention!



More information:



<https://eurocc.cyi.ac.cy/>  
<https://www.linkedin.com/company/eurocc2>



Contact us at:  
[eurocc-contact@cyi.ac.cy](mailto:eurocc-contact@cyi.ac.cy)



Co-funded by the  
European Union



Republic of Cyprus



RESEARCH  
& INNOVATION  
FOUNDATION



**EuroHPC**  
Joint Undertaking

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus (co-funded by the EU within the framework of the Cohesion Policy Programme “THALIA 2021-2027”), Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.