



# Introduction to ARM Performance Reports

HPC.NRW Competence Network



THE COMPETENCE NETWORK FOR HIGH PERFORMANCE COMPUTING IN NRW.

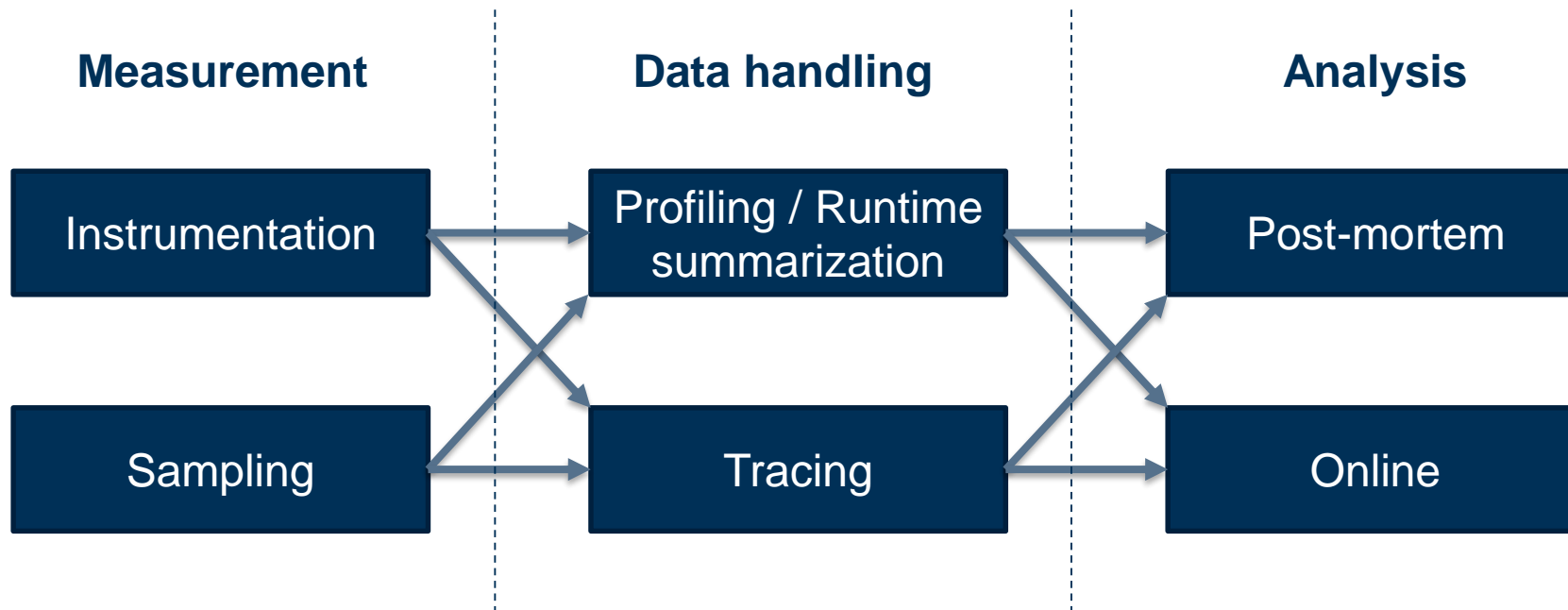
# Performance Analysis 101

HPC.NRW Competence Network

## Introduction to ARM Performance Reports

- Gain an understanding of the application behavior
- Identify potential performance problems of the application and their sources
  - You observe the symptoms, but need to fix the problem
- Present measurement and analysis data in a human understandable way
- Performance analysis tools do not automatically tune your application
  - Auto-tuning tools exist, but are not covered here

- Vast number of different tools available
- Which tool to use?
  - Purpose may dictate the type of tool
  - Tools often specialized for a specific purpose (Threading, Cache, Communication, ...)
- Coarse-grained (technical) classification along three dimensions
  - Measurement
    - How is measurement data obtained? (Instrumentation vs. Sampling)
  - Data handling
    - How is measurement data handled at runtime? (Runtime summarization vs. Tracing)
  - Analysis
    - When is analysis data obtained / presented? (Online vs. Post-mortem)



- Insert measurement code into the application at specific locations in the application
  - State transitions (events)
    - Entering/Leaving a function call (provides context)
    - Communication and synchronization points (provides interaction and dependency information)
- Can be done ...
  - Before compilation (using source-to-source translation)
  - During compilation (using compiler hooks)
  - During linking (using pre-instrumented libraries)
  - After compilation (using binary instrumentation / binary rewriting)

- Measurement system is not part of the application
  - Application is interrupted by the measurement system to obtain data (a.k.a. “a sample”)
- Application is unmodified (debugging symbols may provide additional context)
  - All compiler optimizations stay in the code
- Behavior between samples remains a blackbox
  - Individual important state transitions may be missed / not be sampled
  - With sufficient measurement time data becomes more reliable
    - Statistically even rare events will be sampled at some point

- Measurement data is processed and aggregated at runtime
  - Original measurement data instance is discarded after aggregation
- Memory footprint independent of execution time
  - Data is aggregated
  - Metric values accumulate over time
- Individual information of measurement data instances is lost
  - Some statistical metrics can be obtained at runtime
  - Some information is lost during processing



- Record measurement data instances
  - No aggregation at runtime
  - No computation of metrics at runtime
- Memory footprint dependent on execution time
  - Can become prohibitive for long running applications
- Complex metrics can be computed from trace
  - Profile can be created from trace, but not vice versa
- Computation of metrics at temporally or spatially different location
  - Online but different computer
  - Post-mortem on the same (or different) computer

- Performance metrics can be investigated at runtime
  - Diagnostics screen
  - Visualization
- Useful in computational steering
  - Use performance information to guide simulation configuration
- Useful in automatic tuning & load balancing
  - Base tuning and data distribution decisions on performance metrics

- Complex performance metrics are computed after measurement
  - Measurement is perturbed the least
- More complex metrics can be computed
  - Measurement will no longer be impaired by heavy-weight computation/communication
  - Multiple sources of data can be combined

- Data set
  - Representative of normal production runs
    - Different memory footprint may impact performance
  - Restrict runtime (1-5 minutes)
    - Should expose behavior
    - Must be repeated multiple times
  - Integrate data verification
    - Performance tools trigger hidden bugs
- Integrate a coarse-grained time measurement to your application
  - Helps to estimate the runtime dilation during a performance measurement run
  - High-level OpenMP, MPI, or UNIX time measurement suffices

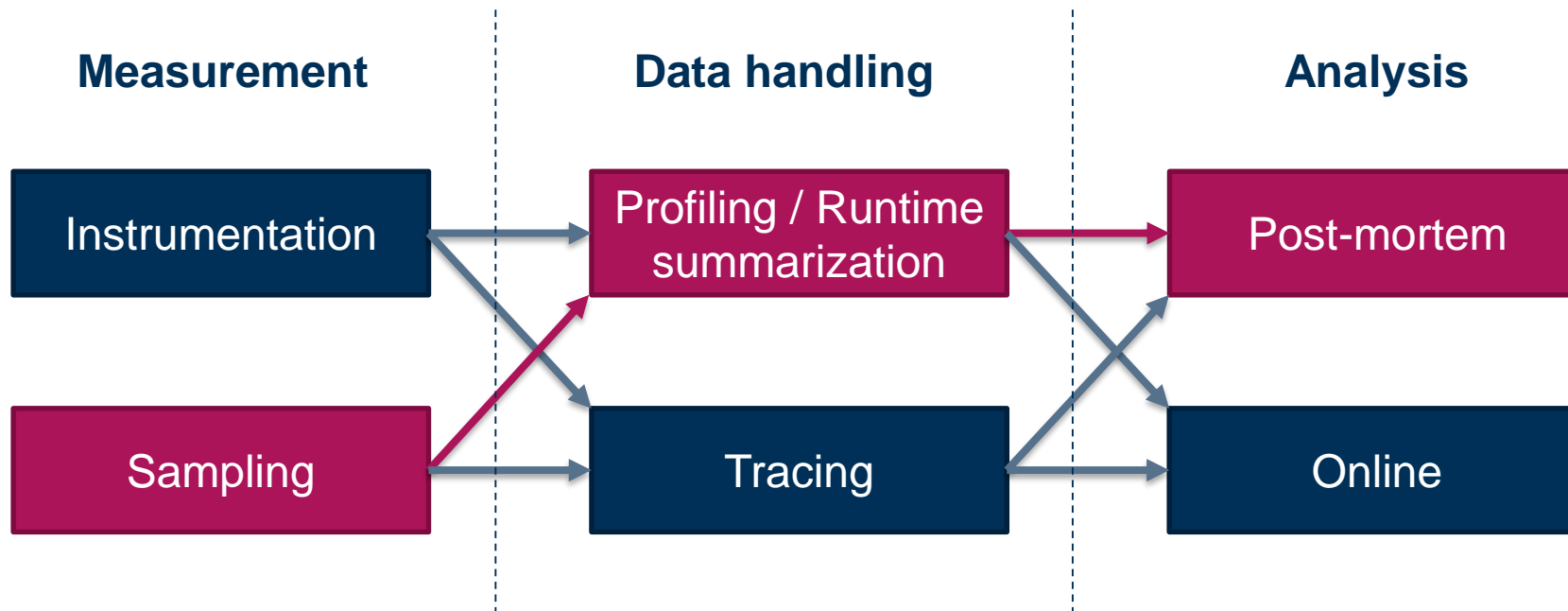
- Establish a “stable” testing environment to get repeatable performance results
  - Use exclusive nodes (minimize outside influence)
  - To test noise resiliency use deterministic noise sources/generators
  - Use thread binding
- Repeat application runs to eliminate outlier behavior (if possible)
  - Use appropriate statistical data analysis of performance results
    - E.g., mean, standard deviation, significance
- Automate measurement and analysis steps as much as possible
  - Use a workflow framework such as JUBE (<https://www.fz-juelich.de/jsc/jube/>)
    - Setup costs of integrating your application in a JUBE workflow will amortize over time

# Using ARM Performance Reports

HPC.NRW Competence Network

## Introduction to ARM Performance Reports

- ARM Performance Reports is a **sampling profiler** that writes a text/HTML report **after measurement**

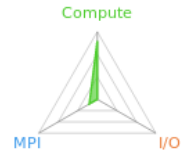


# ARM Performance Reports – Overview

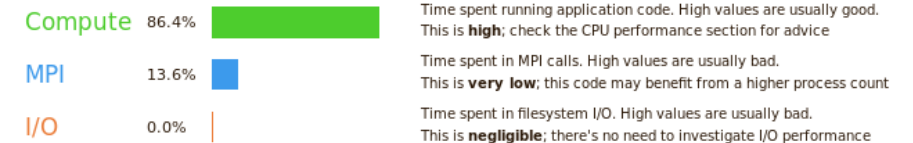
- General overview of application behavior
  - Suitable for initial performance screening
- Breakdown of behavior in different categories
- General advice on how to improve performance
  - Provides human readable, high-level information
  - May need experience to derive actions



Command: /opt/intel/impi/2018.4.274/compilers\_and\_libraries/linux/mpi/bin64/mpirun -np 4 jacobi.exe  
Resources: 1 node (48 physical, 48 logical cores per node)  
Memory: 376 GiB per node  
Tasks: 4 processes  
Machine: login18-1.hpc.itc.rwth-aachen.de  
Start time: Mi. Dez. 2 22:21:43 2020  
Total time: 147 seconds (about 2 minutes)  
Full path: /rwthfs/rz/cluster/home/mh269604/PPCES2020/MPI/lab/C/6\_jacobi



Summary: jacobi.exe is **Compute-bound** in this configuration

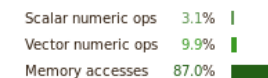


This application run was **Compute-bound**. A breakdown of this time and advice for investigating further is in the **CPU** section below.

As very little time is spent in **MPI** calls, this code may also benefit from running at larger scales.

## CPU

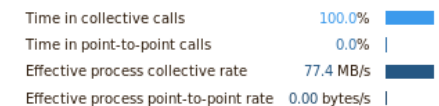
A breakdown of the **86.4%** CPU time:



The per-core performance is **memory-bound**. Use a profiler to identify time-consuming loops and check their cache performance.

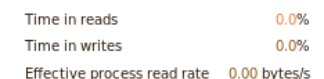
## MPI

A breakdown of the **13.6%** MPI time:



## I/O

A breakdown of the **0.0%** I/O time:



## Threads

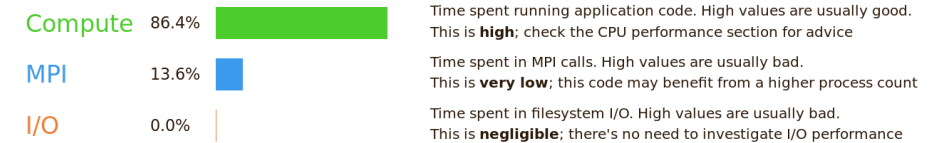
A breakdown of how multiple threads were used:





- Identify where most wallclock time was spent
  - Computation
  - Communication (MPI)
  - I/O
  
- Explains high/low values and guides conclusions
  
- **Note:** **Compute-bound** here just means most time is spent with CPU related tasks
  - Does **NOT** refer to “*compute-bound vs. memory-bound*”

Summary: jacobi.exe is **Compute-bound** in this configuration



This application run was **Compute-bound**. A breakdown of this time and advice for investigating further is in the **CPU** section below.

As very little time is spent in **MPI** calls, this code may also benefit from running at larger scales.

- How well is the code vectorized?
  - Scalar vs. Vector ops
- Is it computing or waiting for data?
  - Compute-bound vs. Memory-bound
- Information on vectorization of your code
  - High vectorization needed for peak performance
  - Not all codes can be vectorized!

## CPU

A breakdown of the 86.4% CPU time:



Scalar numeric ops	3.1%	
Vector numeric ops	9.9%	█
Memory accesses	87.0%	█

The per-core performance is **memory-bound**. Use a profiler to identify time-consuming loops and check their cache performance.

- Check for excessive communication time
  - Depends on application
    - Below 20% considered OK here
- Low transfer rates may indicate ...
  - Waiting time
  - Communication overhead on small messages

## MPI




A breakdown of the 13.6% MPI time:

Time in collective calls	100.0%	
Time in point-to-point calls	0.0%	
Effective process collective rate	77.4 MB/s	
Effective process point-to-point rate	0.00 bytes/s	

- Verify memory consumption
  - Overall consumption
- Mean vs. Peak
  - Are some processes using more than the rest?
- How much headroom is available?
  - Am I using the right scale for my dataset?
  - Estimate feasibility of tracing

## Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	1.55 GiB	
Peak process memory usage	4.54 GiB	
Peak node memory usage	28.0%	


There is **significant variation** between peak and mean memory usage. This may be a sign of workload imbalance or a memory leak.

The **peak node memory usage** is very low. Running with fewer MPI processes and more data on each process may be more efficient.

- Name depending on detected threading model
- Computation
  - Time threads spent actually computing
- Synchronization
  - Time spent in locks and synchronization primitives
- Physical core utilization
  - Values > 100% indicates active hyper-threading
- System load
  - Values < 100% indicate unused resources

## I Threads

A breakdown of how multiple threads were used:

Computation	100.0%	
Synchronization	0.0%	
Physical core utilization	9.5%	
System load	16.5%	

Physical core utilization is low. Try increasing the number of threads or processes to improve performance.

- Check for excessive I/O time
  - Serialized I/O can induce a lot of waiting time
- Low transfer rates may indicate ...
  - High meta data server load
    - HPC filesystems not build for many small ops
  - Filesystem contention
    - Writes to same block from multiple processes

## I/O

A breakdown of the 0.0% I/O time:


Time in reads	0.0%	
Time in writes	0.0%	
Effective process read rate	0.00 bytes/s	
Effective process write rate	0.00 bytes/s	

No time is spent in I/O operations. There's nothing to optimize here!

- Energy consumption by component
  - CPU
  - Accelerators (only shown when available)
  - System (everything else)
- Needs special daemon to collect node-level information

## Energy

A breakdown of how the 7.34 Wh was used:

CPU	100.0%	
System	not supported %	
Mean node power	not supported W	
Peak node power	0.00 W	

The whole system energy has been calculated using the CPU energy usage.

System power metrics: No Arm IPMI Energy Agent config file found in /var/spool/ipmi-energy-agent. Did you start the Arm IPMI Energy Agent?

- Sampling profiler
  - Easy initial performance investigation
  - No special preparation of application needed
- Provides high-level information and advice
  - Key factors covered (Computation, MPI, Threading, IO, Memory)
- May serve for initial performance screening
  - Use report to decide which other tools to employ next