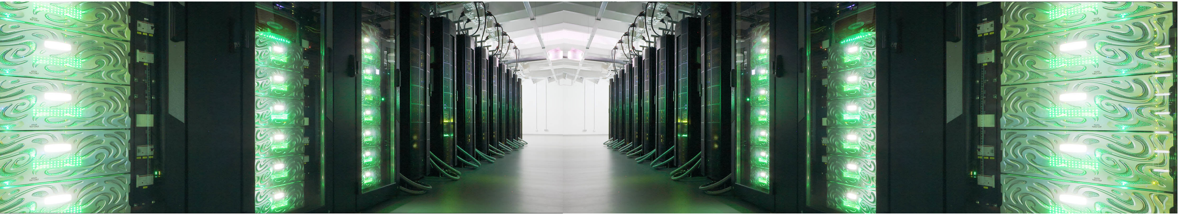
Aasish Kumar Sharma

Introduction to Performance Engineering



2025-04-08

Learning Objectives

- Describe basic system and workload characteristics.
- Understand performance engineering and its fundamentals.
- Utilize a basic system model to assess performance measures.
- Grasp Compute (strong/weak), Memory, and Storage/Network (I/O) performance metrics.
- Recognize the challenges of performance analysis and optimization.

Our Journey Today

- **Part 1:** Unraveling the basics—understanding systems and workloads.
- **Part 2:** Diving into profiling, modeling, and scaling strategies.
- **Part 3:** Real-world case studies and practical insights.

Outline

Interesting Facts!!

According to the American National Institute of Standards and Technology,[1]

- Nearly, four out of every five dollars spent on the total cost of ownership of an application
  - is directly attributable to finding and fixing issues post-deployment.
- A full one-third of this cost could be avoided with better software testing.

[1] https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf

Do You Know What Are The Goals for HPC Hosts/Owners?

- To empower users with complete compute/storage resources usages.
- What are the requirements to fulfill this goal?

Do You Know What Are The Goals for HPC Hosts/Owners?

- To empower users with complete compute/storage resources usages.
- What are the requirements to fulfill this goal?
  - **Ease of Use**: Empower users to easily acquire the compute/storage resources they need.
  - **Usability**: Enable users to fully utilize the provided resources.

Do You Know What Are The Goals for HPC Hosts/Owners?

- To empower users with complete compute/storage resources usages.
- What are the requirements to fulfill this goal?
  - **Ease of Use**: Empower users to easily acquire the compute/storage resources they need.
  - **Usability**: Enable users to fully utilize the provided resources.
  - **Efficiency**: Critical for High-Performance Computing—for example,
    - If only 1% of the node resources are used efficiently, scaling becomes meaningless.
  - **Cost-efficiency**: Proper resource utilization reduces overall costs.

Do You Know What Are The Goals for HPC Hosts/Owners?

- To empower users with complete compute/storage resources usages.
- What are the requirements to fulfill this goal?
  - **Ease of Use**: Empower users to easily acquire the compute/storage resources they need.
  - **Usability**: Enable users to fully utilize the provided resources.
  - **Efficiency**: Critical for High-Performance Computing—for example,
    - If only 1% of the node resources are used efficiently, scaling becomes meaningless.
  - **Cost-efficiency**: Proper resource utilization reduces overall costs.
  - **Programmability**: Simplify application development for programmers.
  - **Performance-portability**: Ensure applications perform consistently across different architectures.

What is Performance Engineering?

Performance Engineering

- **Definition:** A systematic approach to optimizing both software and hardware performance.
- Requires a deep understanding of:
  - System and Application behaviors,
  - Tools and Models that help in performance measurement.

What is Performance Engineering?

- Performance Engineering
  - **Definition:** A systematic approach to optimizing both software and hardware performance.
  - Requires a deep understanding of:
    - System and Application behaviors,
    - Tools and Models that help in performance measurement.
- Crucial for HPC centers **ref1**; **ref2**
  - Scaling applications,
  - Energy efficiency, and
  - Maximizing resource utilization.

What is Performance Engineering?

Performance Engineering

- **Definition:** A systematic approach to optimizing both software and hardware performance.
- Requires a deep understanding of:
  - System and Application behaviors,
  - Tools and Models that help in performance measurement.

Crucial for HPC centers ref1; ref2

- Scaling applications,
- Energy efficiency, and
- Maximizing resource utilization.

Goals of Performance Engineering:

- Identify performance bottlenecks.
- Improve application scalability.
- Maximize efficiency on modern hardware.

Efficiency

- Let us focus on efficiency.
- What is efficiency from a System/Application perspective?

Efficiency

- Let us focus on efficiency.
- What is efficiency from a System/Application perspective?
  - It means using the full capabilities of the available hardware.

$$Efficiency = \frac{\text{Total Resources Utilized}}{\text{Total Resources Available}} \times 100\% \tag{1}$$

  - Examples:
    - CPU/GPU utilization at 100%
    - Network/storage bandwidth: Using 9 out of 10 GBit/s
    - Memory/storage capacity usage at 90%
  - Note that applications may use different resources in varying proportions.

Efficiency

System/Data center perspective

- Efficiency means optimum utilization of the allocated resources and runtime.
- We invest in high resources, so they must be fully utilized.

Efficiency

System/Data center perspective

- Efficiency means optimum utilization of the allocated resources and runtime.
- We invest in high resources, so they must be fully utilized.

User perspective

- Efficiency means maximum utilization of available resources and runtime.
- Users might expect that using 10x nodes/cores results in 1/10th of the runtime.

Motivation in HPC Context

- Expensive compute time and energy costs.
- Heterogeneous and complex system architectures.
- Critical for both scientific and industrial applications.

How Can We Understand System or Workload Behavior?

To understand System and Workload (Application) Performance, we can follow:

- **Methodology:** Performance Modeling
  - Modeling – Determining performance characteristics.
  - Behavioral models – Building models based on observed behavior.
  - Characteristics – Fundamental parameters for system and workload models.

How Can We Understand System or Workload Behavior?

To understand System and Workload (Application) Performance, we can follow:

- **Methodology:** Performance Modeling
  - Modeling – Determining performance characteristics.
  - Behavioral models – Building models based on observed behavior.
  - Characteristics – Fundamental parameters for system and workload models.
- **Observation:** Analyze the Records
  - Measurements – Recording system/application behavior.
  - Benchmarking – Using specialized tests to reveal system behavior.
  - Tracing/Profiling – Logging operations to analyze timing.

How Can We Understand System or Workload Behavior?

To understand System and Workload (Application) Performance, we can follow:

- **Methodology:** Performance Modeling
  - Modeling – Determining performance characteristics.
  - Behavioral models – Building models based on observed behavior.
  - Characteristics – Fundamental parameters for system and workload models.
- **Observation:** Analyze the Records
  - Measurements – Recording system/application behavior.
  - Benchmarking – Using specialized tests to reveal system behavior.
  - Tracing/Profiling – Logging operations to analyze timing.
- **Monitoring:** Applying tools and techniques to collect data.
- Key! Simulate the system/application based on models.

Outline

System And Workload Characteristics

System Characteristics

- CPU/GPU architecture
- Memory hierarchy
- Interconnects (e.g., InfiniBand, Ethernet)
- Node topology

System And Workload Characteristics

System Characteristics

- CPU/GPU architecture
- Memory hierarchy
- Interconnects (e.g., InfiniBand, Ethernet)
- Node topology

Workload Characteristics

- Compute-bound vs. memory-bound vs. I/O-bound
- Static vs. dynamic workloads
- Parallelism: task, data, and pipeline

System And Workload Characteristics

System Characteristics

- CPU/GPU architecture
- Memory hierarchy
- Interconnects (e.g., InfiniBand, Ethernet)
- Node topology

Workload Characteristics

- Compute-bound vs. memory-bound vs. I/O-bound
- Static vs. dynamic workloads
- Parallelism: task, data, and pipeline

In summary, there are four main components:

- *Compute characteristics*: Execution time, CPU/GPU utilization.
- *Memory characteristics*: Peak memory usage, swap behavior.
- *Storage characteristics*: Read/write speeds, file system latency.
- *Network characteristics*: Data transfer speed, latency.

HPC Compute Cluster With Storage and Network Topology

- This is an abstract view of a compute cluster showing shared storage and network topology.

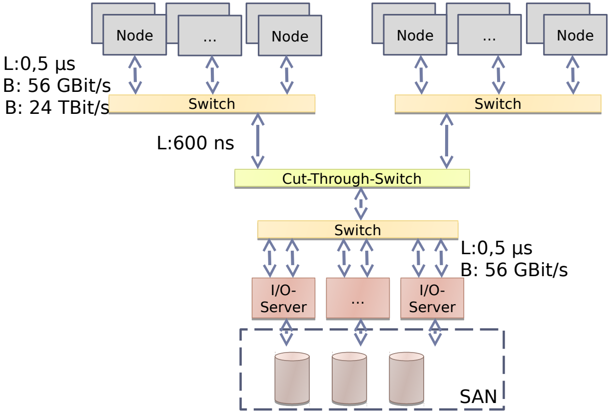
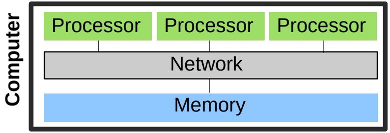


Figure: Architecture of a typical HPC cluster (using a fat-tree network topology)

HPC: Parallel & Distributed Memory Architectures

In practice, HPC memory systems blend two paradigms

Shared memory

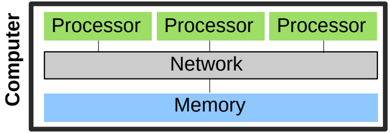


- Suited for varying memory workloads.
  - Requires Communication coordination!
- Limited scope for scaling.
- Expensive to build a large one.

HPC: Parallel & Distributed Memory Architectures

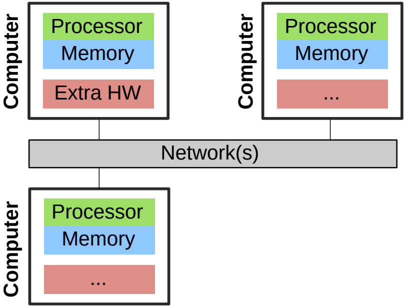
In practice, HPC memory systems blend two paradigms

Shared memory



- Suited for varying memory workloads.
  - Requires Communication coordination!
- Limited scope for scaling.
- Expensive to build a large one.

Distributed memory systems



- Highly scalable
  - Network performance is the key!
- Each Processor accesses its own

Outline

Application Performance Profiling

- Performance modeling provides theoretical insights into application behavior **ref3**; **ref4**.
- The performance of any parallel application is ultimately limited by one resource.
  - For example: Compute, Memory, Storage, or Network.

Application Performance Profiling

- Performance modeling provides theoretical insights into application behavior **ref3**; **ref4**.
- The performance of any parallel application is ultimately limited by one resource.
  - For example: Compute, Memory, Storage, or Network.
- Application profiles indicate whether the app is compute, memory, network, or I/O bound.
  - Even within a single core, applications can stress different parts of the processor.

Application Performance Profiling

- Performance modeling provides theoretical insights into application behavior **ref3**; **ref4**.
- The performance of any parallel application is ultimately limited by one resource.
  - For example: Compute, Memory, Storage, or Network.

- Application profiles indicate whether the app is compute, memory, network, or I/O bound.
  - Even within a single core, applications can stress different parts of the processor.

- **Operational intensity** for a process can be expressed as:
  - $I = \frac{W}{Q}$ (operations per byte of memory traffic).
    - Here, W is work (often measured in FLOPs) and Q is the memory traffic.

  - Analyzing whether an application is memory or compute-bound helps in deciding where to optimize.

Profiling Tools and Benchmarking Approaches and Trends

Profiling Tools

- gprof, perf, Intel VTune
- nvprof, Nsight (for GPU)
- HPCToolkit, Vampir, Score-P, TAU

Profiling Tools and Benchmarking Approaches and Trends

Profiling Tools

- `gprof`, `perf`, `Intel VTune`
- `nvprof`, Nsight (for GPU)
- `HPCToolkit`, `Vampir`, `Score-P`, `TAU`

Benchmarking Approaches

- Micro-benchmarks: LINPACK, STREAM
- Application benchmarks: IO500, HPCG, NAS
- Synthetic workloads and real traces

Profiling Tools and Benchmarking Approaches and Trends

Profiling Tools

- `gprof`, `perf`, `Intel VTune`
- `nvprof`, Nsight (for GPU)
- `HPCToolkit`, `Vampir`, `Score-P`, `TAU`

Benchmarking Approaches

- Micro-benchmarks: LINPACK, STREAM
- Application benchmarks: IO500, HPCG, NAS
- Synthetic workloads and real traces

Changing Trends

- Increasing complexity of HPC systems.
- Adoption of adaptive and intelligent optimization solutions.
- Integration of machine learning for performance tuning **ref5**.

Outline

System Performance Modeling

Compute and Memory

- CPU performance: Frequency $\times$ cores $\times$ sockets.
  - Example: 2.5 GHz $\times$ 12 cores $\times$ 2 sockets = 60 Gcycles/s.
  - Note: The cycles per operation depend on the instruction mix.
- Memory: (Throughput $\times$ channels) plus latency per access.
  - Example: 25.6 GB/s per DDR4 DIMM, with cache hierarchies (L1/L2/L3) playing a role.

System Performance Modeling

Compute and Memory

- CPU performance: Frequency $\times$ cores $\times$ sockets.
  - Example: 2.5 GHz $\times$ 12 cores $\times$ 2 sockets = 60 Gcycles/s.
  - Note: The cycles per operation depend on the instruction mix.
- Memory: (Throughput $\times$ channels) plus latency per access.
  - Example: 25.6 GB/s per DDR4 DIMM, with cache hierarchies (L1/L2/L3) playing a role.

Communication via the Network

- Throughput: Data transferred per unit time (e.g., 125 MiB/s with Gigabit Ethernet).
- Latency: The delay or overhead time (e.g., 0.1 ms with Gigabit Ethernet).

System Performance Modeling

Compute and Memory

- CPU performance: Frequency $\times$ cores $\times$ sockets.
  - Example: 2.5 GHz $\times$ 12 cores $\times$ 2 sockets = 60 Gcycles/s.
  - Note: The cycles per operation depend on the instruction mix.
- Memory: (Throughput $\times$ channels) plus latency per access.
  - Example: 25.6 GB/s per DDR4 DIMM, with cache hierarchies (L1/L2/L3) playing a role.

Communication via the Network

- Throughput: Data transferred per unit time (e.g., 125 MiB/s with Gigabit Ethernet).
- Latency: The delay or overhead time (e.g., 0.1 ms with Gigabit Ethernet).

Input/Output Devices: Storage

- HDDs have mechanical limitations (seek time due to head movement and rotation).
  - Performance is influenced by I/O granularity and access patterns.
    - Example: 150 MiB/s with 10 MiB blocks.

Amdahl Law, Speedup and Strong Scaling

- **Amdahl's Law** (1967):
  - Speedup:

$$S_{\text{Amdahl}} = \frac{1}{s + \frac{p}{N}} \quad (2)$$

    - N: Number of processors.
    - s: Serial fraction; $p = 1 - s$: Parallel fraction.
    - This sets an upper bound on strong scaling.
  - Constraints:
    - Constant problem size.
    - Increasing number of processors (N).
  - Example:
    - Weather forecasting using more processors—practical limits exist.

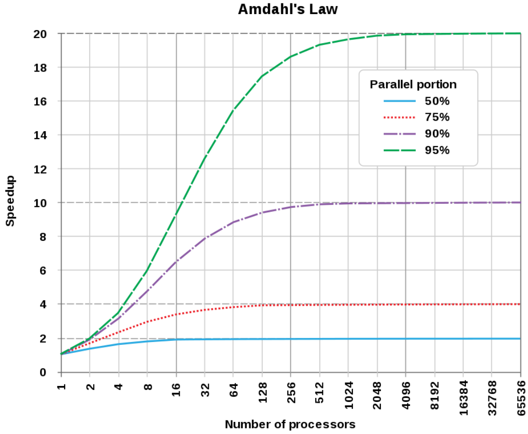


Figure: Source: Daniels220, Wikipedia

Amdahl Law, Speedup and Strong Scaling

- **Amdahl's Law** (1967):
  - Speedup:

$$S_{\text{Amdahl}} = \frac{1}{s + \frac{p}{N}} \tag{3}$$

    - N: Number of processors.
    - s: Serial fraction; $p = 1 - s$: Parallel fraction.
    - This sets an upper bound on strong scaling.
  - Constraints:
    - Constant problem size.
    - Increasing number of processors (N).
  - Example:
    - Weather forecasting using more processors—practical limits exist.

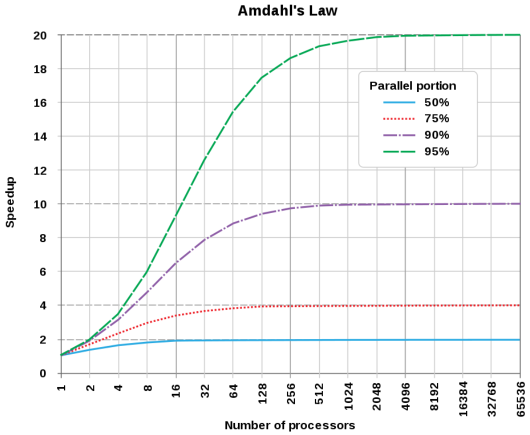


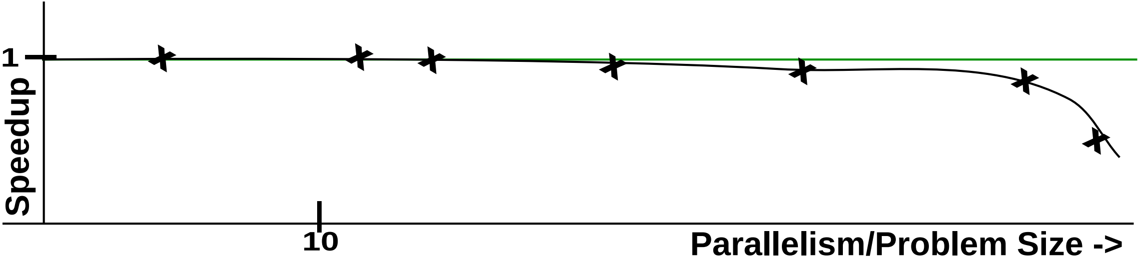
Figure: Source: Daniels220, Wikipedia

Efficiency

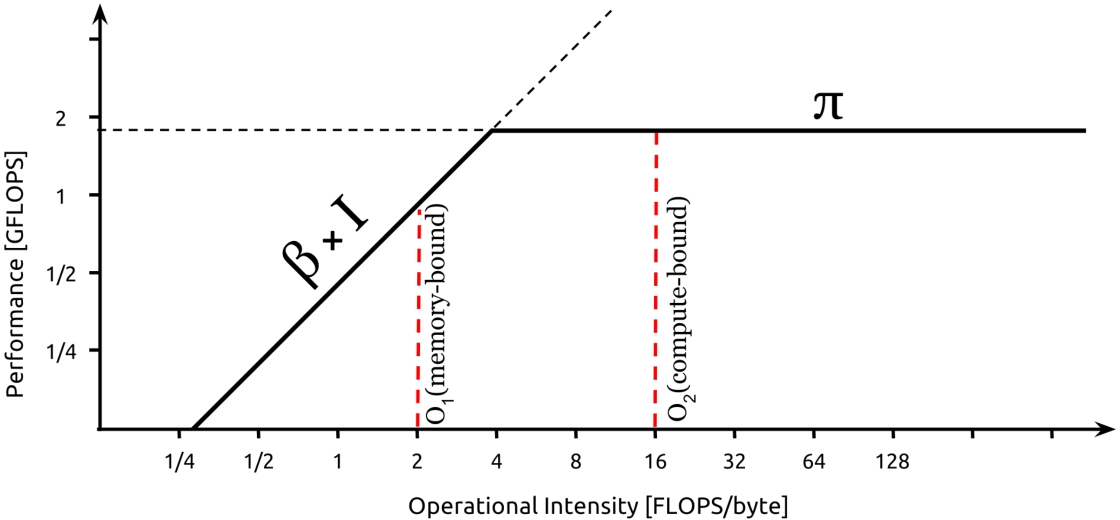
- **Definition**: $\frac{\text{Speedup}}{\text{Parallelism}} \times 100\%$.

Weak Scaling

- Scenario: Increasing both problem size and resources.
  - The work per processor remains constant, ideally keeping runtime the same.
- Example: Using 10× nodes for a 10× larger problem.
- Optimal outcome: Runtime remains constant.



Roofline Model: Naive Model



Roofline Model: Illustrates Memory and Compute Limitations

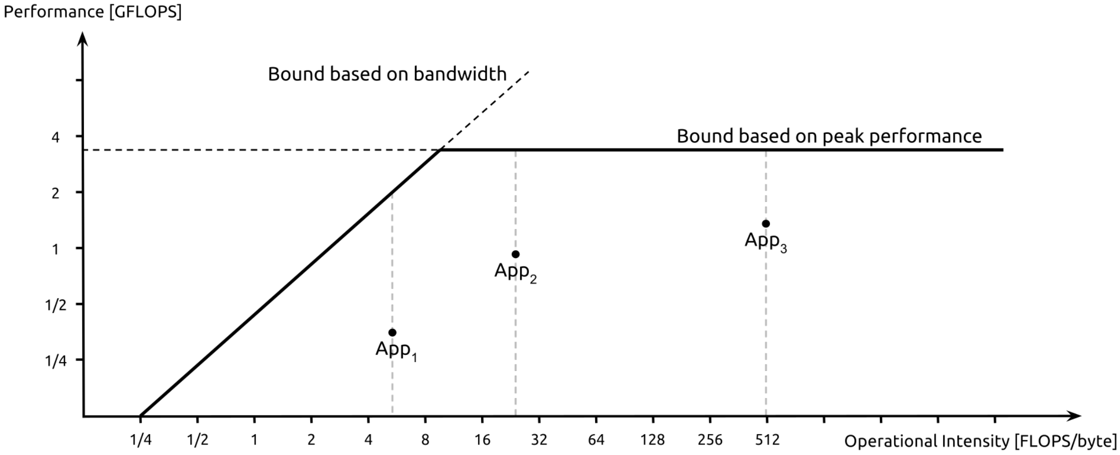
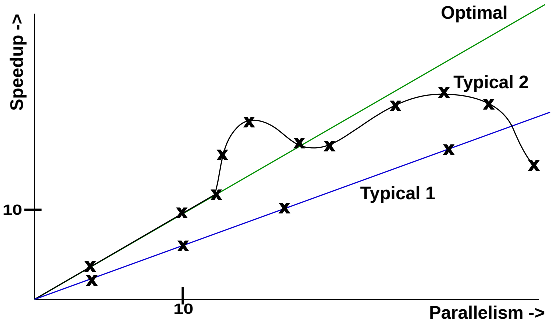


Figure: Giu.natale / Wikipedia

Group-work: Assessing Speedup Diagrams

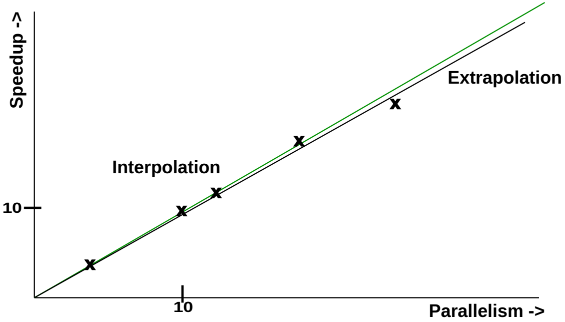
Task: Assess the two strong scaling curves, Typical 1 and Typical 2

- Evaluate whether the measurements of Typical 1 or Typical 2 are realistic.
- Discuss potential causes for performance variations in Typical 2.
- Identify any relationships between the shapes of Typical 1 and Typical 2.
- **Time:** 5 minutes.



(Self-) Cheating

- Fewer measured points can lead to deceptive curves.
  - Typical 1 might be misinterpreted as Typical 2.
- Interpolating missing points may mask true performance.
- Be cautious: Reported *Speedups* $>$ *Parallelism* imply efficiency $>$ 100%, which is suspicious.



Execution Time Breakdown

- Separates CPU time from I/O time.
- Considers communication overhead.
- Includes synchronization costs in parallel programs.

Example: Benchmark for Memory Throughput

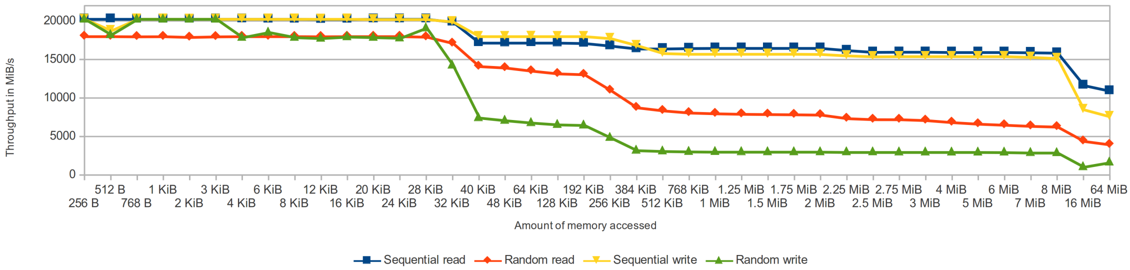


Figure: Memory performance using the fbui benchmark (on an older system)

Outline

Code-Level Optimization

- Techniques include loop unrolling, inlining, and vectorization.
- Minimize unnecessary memory accesses.
- Adopt cache-aware programming strategies.

Code Optimization

Alternatives/Options

1. Run code on a more appropriate system. Example:
   - Faster hardware, more memory, different CPUs.
2. Tune execution without altering the code.
3. Increase efficiency by directly optimizing the code—though this can be complex.

Code Optimization

Alternatives/Options

1. Run code on a more appropriate system. Example:
   - Faster hardware, more memory, different CPUs.
2. Tune execution without altering the code.
3. Increase efficiency by directly optimizing the code—though this can be complex.

Tuning

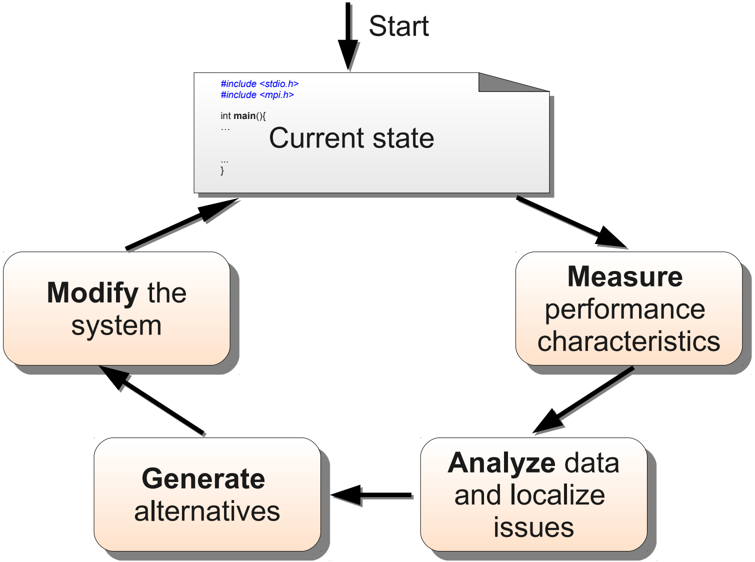
- Analyze and adjust system parameters without changing the application code.
- Examples:
  - Compiler optimizations, system setting tweaks, and modifying tunable parameters.

- Thus, a user must have a basic understanding of both systems and workloads.

Parallelization Strategies

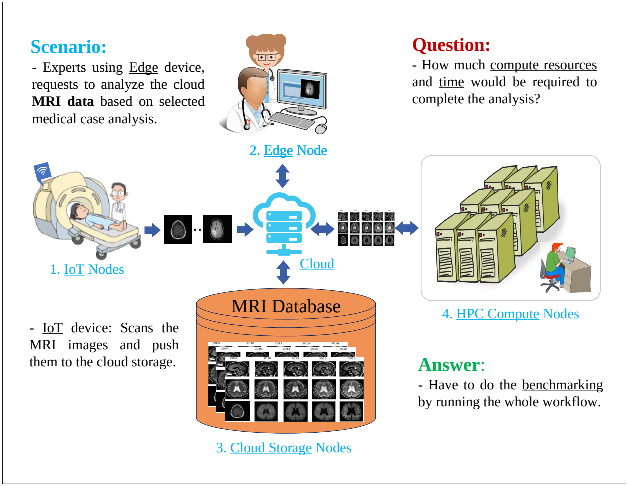
- MPI, OpenMP, CUDA, HIP.
- Consider task granularity and load balancing.
- Focus on data partitioning and affinity.

Optimization Cycle (for Any System)



Outline

Scenario: MRI Use Case



Case Studies

Taking the MRI Use Case, we learn:

- **Case Study 1:** Theoretical Study – How to model the workflow.
- **Case Study 2:** Practical Study – How to perform performance profiling.

Why MRI?

- MRI (Magnetic Resonance Imaging) is crucial for brain diagnosis.
- MRI processing is both compute- and data-intensive—ideal for performance engineering.
- A typical MRI pipeline includes preprocessing, registration, segmentation, and analysis.

Case Study 1: Workflow Modeling

Objective: Map MRI data processing stages to computing resources.

- **Stage 1: Preprocessing** – Noise reduction and bias correction.
- **Stage 2: Registration** – Align scans with a brain atlas.
- **Stage 3: Segmentation** – Label anatomical regions.
- **Stage 4: Analysis** – Quantify structures and detect abnormalities.

Case Study 1: Workflow Modeling

Objective: Map MRI data processing stages to computing resources.

- **Stage 1: Preprocessing** – Noise reduction and bias correction.
- **Stage 2: Registration** – Align scans with a brain atlas.
- **Stage 3: Segmentation** – Label anatomical regions.
- **Stage 4: Analysis** – Quantify structures and detect abnormalities.

Tools: FreeSurfer, FSL, ANTs, Nipype. **Modeling Inputs:**

- Task durations and dependencies (represented as a DAG).
- Required memory, CPU/GPU usage.
- Data transfer times.

Case Study 2: Performance Profiling

Objective: Identify bottlenecks and resource inefficiencies in the MRI workflow.

Profiling Techniques:

- Use `perf` or `LIKWID` for CPU and memory usage.
- Monitor disk and I/O overhead with tools like `iotop` or HPC monitors.
- Examine task allocation across nodes via logs or trace tools (e.g., `Snoopy`, `Paraver`).

Results Example:

- The segmentation task is GPU-intensive—consider appropriate node mapping.
- Preprocessing shows high I/O latency—SSD-backed nodes might help.

Case Study 1: How to do Modeling? System Characteristics

Suppose the following are the System Characteristics (Available Resources):

Nodes	N_1	N_2	N_3...
Node Name	Node 1	Node 2	Node 3
Core (R^1)	8	48	2572
Storage (R^3 in TB)	0.5	20	210
Features	F^1	F^1, F^2	F^1, F^2
Data Transfer Rate (P^3 in GB/s)	100	100	100
Processing Speed (P^1 FLOPS)	1	1	1

Case Study 1: MRI Workflow on HPC

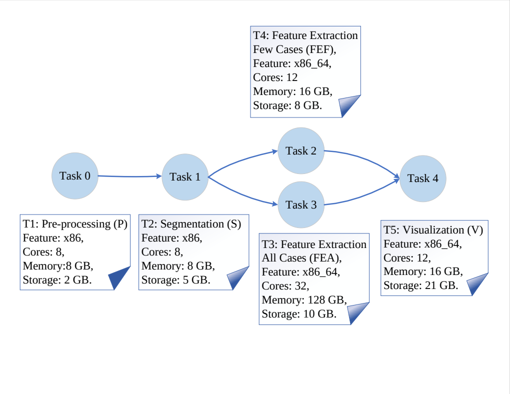


Figure: Two workflows of MRI workload.

- Pre-processing occurs at the edge.
- Analysis spans both cloud and HPC environments.

How to do Modeling: Workload Characteristics (Contd..)

Workload Modeling Characteristics:

W Id	T Id	(R^1)	(F^f)	(R^3)	(d^r_{ij}) $N1, N2, N3$	$(d^r_{t:ii'j'})$ (same)	$\delta(j, j')$
	T_1	8	F_1	2	(3, 3, 3)	0.02	-
W_1	T_2	12	F_1, F_2	5	(5, 5, 5)	0.05	$\delta(T_1, T_2)$
	T_3	12	F_1, F_2	8	(2, 2, 2)	0.08	$\delta(T_2, T_3)$
	T_1	8	F_1	2	(3, 3, 3)	0.02	-
W_2	T_2	12	F_1, F_2	5	(5, 5, 5)	0.05	$\delta(T_1, T_2)$
	T_3	32	F_1, F_2	5	(2, 2, 2)	0.05	$\delta(T_1, T_3)$
	T_4	12	F_1, F_2	10	(2, 2, 2)	0.10	$\delta(T_2, T_4)$, $\delta(T_3, T_4)$

Case Study 1: Theoretical Findings

Estimate:
→ What should be the makespan, theoretically?

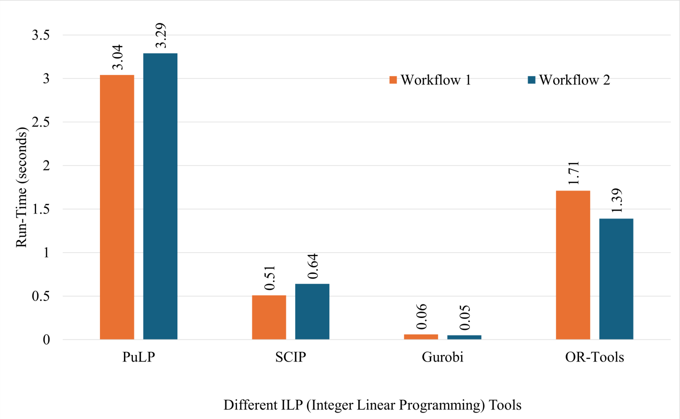
Theoretical Findings

Manual Estimation: Best Theoretical Solution

Status	Workflow Id	Task Id	Optimal Node	Start (sec.)	End (sec.)	Resource Usage	Makespan (sec.)
		T_1	N_2	0.0	3.0		
Optimal	W_1	T_2	N_2	3.0	8.0		
		T_3	N_2	8.0	10.0		
	Total					32.0	10.0
		T_1	N_1	0.0	3.0		
Optimal	W_2	T_2	N_1	3.0	8.0		
		T_3	N_2	3.02	5.02		
		T_4	N_1	8.0	10.0		
	Total					64.0	10.0

Case Study 2: Experimental Findings

Experimentally: When programmed this model using different ILP tools and did Profiling:



- The performance of the modeling logic varied with different ILP tools implementation.

Outline

Conclusion

- **Performance:**
  - Goal (from a user perspective): Optimize time-to-solution.
  - Understand hardware to assess performance accurately.
  - Linear scalability of the architecture is crucial.
  - Essential steps:
    1. Estimate the workload.
    2. Compute workload throughput per node.
    3. Compare with hardware capabilities.

Conclusion

- **Performance:**
  - Goal (from a user perspective): Optimize time-to-solution.
  - Understand hardware to assess performance accurately.
  - Linear scalability of the architecture is crucial.
  - Essential steps:
    1. Estimate the workload.
    2. Compute workload throughput per node.
    3. Compare with hardware capabilities.
- Achieving peak performance is challenging due to:
  - Complex systems, deep software stacks, variability, and optimization challenges.

Conclusion

- **Performance:**
  - Goal (from a user perspective): Optimize time-to-solution.
  - Understand hardware to assess performance accurately.
  - Linear scalability of the architecture is crucial.
  - Essential steps:
    1. Estimate the workload.
    2. Compute workload throughput per node.
    3. Compare with hardware capabilities.
- Achieving peak performance is challenging due to:
  - Complex systems, deep software stacks, variability, and optimization challenges.
- Monitoring, performance analysis, and benchmarking are indispensable.
- Future sessions will apply these techniques to real HPC applications.

Key Takeaways

- Performance engineering is essential for modern HPC applications.
- Comprehensive system and workload modeling leads to better automation.
- Future trend is getting complex, demanding adaptive & intelligent optimization solutions.

Learning Materials

Books

1. **High Performance Computing: Modern Systems and Practices**
   Authors: Thomas Sterling, Matthew Anderson, Maciej Brodowicz
   A comprehensive guide on HPC systems, programming models, and performance engineering.
2. **Performance Tuning of Scientific Applications**
   Editor: David H. Bailey
   A collection of case studies and techniques on profiling and optimizing scientific applications.
3. **Efficient R Programming** (Chapter on HPC concepts and profiling)
   Authors: Colin Gillespie, Robin Lovelace
   `Open Access: https://csgillespie.github.io/efficientR/`
4. **High Performance Computing (HPC) for Dummies**
   Author: Douglas Eadline
   Introductory text covering key HPC principles and tuning concepts.

Learning Materials

Tools and Frameworks

- **LIKWID:** `https://github.com/RRZE-HPC/likwid`
- **TAU:** `https://www.cs.uoregon.edu/research/tau/`
- **Score-P:** `https://score-p.readthedocs.io`
- **Paraver:** `https://tools.bsc.es/paraver`
- **Intel VTune Profiler:** `https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html`

References

Dongarra, J., et al. (2020). "High-Performance Computing: State of the Art." *Journal of Parallel and Distributed Computing.*

Hoefler, T., et al. (2015). "Performance Modeling and Prediction for Modern HPC Systems." *ACM Computing Surveys.*

Gropp, W., Lusk, E., & Skjellum, A. (1999). *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press.

Choi, J., et al. (2012). "Scalable Performance Analysis Tools for HPC." *IEEE Transactions on Parallel and Distributed Systems.*

Agakov, F., et al. (2006). "Using Machine Learning for Automated Performance Tuning." *IEEE Transactions on Parallel and Distributed Systems.*

Questions and Feedback?

Ask here or email:

- aasish-kumar.sharma@gwdg.edu