Performance Modelling 101

Simon Schwitanski

IT Center, RWTH Aachen University

December 5th, 2022 | Aachen
Performance Modelling – Goals

What is performance modelling in HPC about?

- Understand the interaction between hardware (processors, network, ...) and software (your code)
  - What is the scaling behavior of my program on the given hardware?
  - What is the maximum achievable performance of my program on the given hardware?
  - Which part of the hardware limits the achievable performance of my program?

```c
#pragma omp parallel for
for (int i=0; i < n; i++) {
    d[i] = a*x[i]+y[i];
}
```
Performance Modelling – Overview

- **Performance Metrics: How to quantify performance?**
- **Hardware Model: Processors, Nodes, Networks**
- **Model 1: Amdahl’s Law**
- **Model 2: Roofline Model**
- **Further Modelling Approaches**
Performance Metrics

- **Performance metric:** (Numerical) value used to compare (1) hardware capabilities or (2) program executions

- Typical metrics for hardware (processors)
  - Clock frequency $f$ [Hz] of a CPU (number of cycles per second)
  - Number of cores of a CPU
  - Maximum number of floating-point operations per second (“peak performance”) [$Flop/s$]
    - Dominant metric in scientific computing / HPC
  - Maximum achievable memory bandwidth $b_s$ [Byte/s]
Performance Metrics

- **Performance metric**: (Numerical) value used to compare (1) hardware capabilities or (2) program executions

- Typical metrics **for program executions**
  - Wall-clock time [s]: Run time / Time-to-solution measured by wall clock
  - CPU time [s]: Sum of execution times of all cores used for computation (“core hours”)
  - Number of floating-point operations performed per second [Flop/s]
  - Amount of data accesses / memory accesses performed per second [Word/s] or [Byte/s]
  - Instructions per cycle (IPC): Number of instructions executed on a processor per cycle

- Characteristics of **good** performance metrics for program executions: reliable, easy to measure, repeatable
Performance Metrics

How to retrieve performance metrics?

• Depends on the metric

• Static hardware metrics (e.g., number of cores, maximum number of Flop/s): Read product sheet

• Time measurements
  – Insert time measurement code manually
  – Use a performance analysis tool (presented later today)

• Number of floating-point operations / memory accesses
  – Look at the source code (count the number of Flop / memory accesses manually)
  – Use hardware performance counters (processor itself can provide metrics, presented later today)
  – Use a performance analysis tool (that uses hardware performance counters, presented later today)
Performance Modelling – Overview

• Performance Metrics: How to quantify performance?
• **Hardware Model: Processors, Nodes, Networks**
  • Model 1: Amdahl’s Law
  • Model 2: Roofline Model
• Further Modelling Approaches
Basic Terminology: Processor

- **Processor**: Hardware unit that runs programs, consists of multiple cores
- **Core**: Processing unit on a processor that can execute instructions independently
- **Main memory**: Location where runtime application data is stored
- **Cache (L1/L2/L3)**: Fast memory to speed up memory accesses
  - Main memory accesses are significantly (order of magnitudes) slower than floating-point computations
  - Cache: Small chunk of memory with high bandwidth / small latency
  - Speeds up repeating memory accesses to the same address (temporal locality) and nearby memory accesses (spatial locality)
Basic Terminology: Shared-Memory Computers

- **Multiple processors** connected to a **shared** main memory (shared-memory computer)
  - Also called “Symmetric multiprocessing” (SMP)
  - NUMA: Non-Uniform Memory Access (multiple memory interfaces, connected via interconnect)
- Setup in CLAIX-2018: 2-processor systems (= “dual-socket” systems)

Shared-Memory Computer with NUMA architecture (2 processors with 4 cores each)
Basic Terminology: Distributed-Memory Computers

- **Node**: Single system consisting of processor(s), memory, network interface
- **Distributed-memory computer** ("cluster"): Combine **nodes**, each of them having private memory
  - Typical setup in a data center / cluster: Connect dual-socket systems via network together
- **Use network interconnect to exchange data** (e.g., via MPI)
Performance Modelling – Overview

• Performance Metrics: How to quantify performance?
• Hardware Model: Processors, Nodes, Networks
• Model 1: Amdahl’s Law
• Model 2: Roofline Model
• Further Modelling Approaches
**Speedup**

- Indicator for relative performance improvement: Speedup
- $T(1)$: serial runtime with a single processor
- $T(N)$: runtime of a (parallel) program with $N$ processors / cores
- Definition of Speedup (according to Amdahl)

\[
\text{Speedup } S_p(N) = \frac{T(1)}{T(N)}
\]

(Ratio between serial and parallel execution of a program)
Linear Speedup

- Ideal situation: All work is perfectly parallelizable: linear Speedup
  - In general: upper bound for parallel execution of programs

\[ S_p(N) = \frac{T(1)}{T(N)} = N \]
Amdahl’s Law

- A more realistic model: There are serial parts which limit the maximum speedup

- Amdahl’s law assumes the program is dividable into an ideal parallelizable fraction $p$ and a serial fraction $s$ (non-parallelizable)

Timeline

\[
\begin{align*}
W_1 & \quad W_2 & \quad W_3 & \quad W_4 \\
\text{serial} & \quad \text{serial} & \quad \text{serial} & \quad \text{serial}
\end{align*}
\]

\[
\frac{s}{2} \cdot T(1) \quad p \cdot T(1) \quad \frac{s}{2} \cdot T(1)
\]

In the given example:
Program start and end are serial. ($s=2 \times s/2$)
Amdahl’s Law

- $s + p = 1 \rightarrow p = 1 - s$

- The parallelized program’s execution time is then assumed to be (with $N$ processors):
  - $T(N) = (s + \frac{p}{N}) \cdot T(1)$

- The speedup thus resembles:

$$S_p(N) = \frac{T(1)}{T(N)} = \frac{T(1)}{\left(s + \frac{p}{N}\right) \cdot T(1)} = \frac{1}{s + \frac{1 - s}{N}}$$


or “strong scaling”
Amdahl’s Law

- Example: Program with 5% serial (s = 0.05) and 95% (p = 0.95) parallel fraction
  - Speedup according to Amdahl:

\[
S_{0.95}(N) = \frac{T(1)}{T(N)} = \frac{1}{0.05 + \frac{1-0.05}{N}}
\]

\[
\lim_{N \to \infty} S_p(N) = \lim_{N \to \infty} \frac{1}{s + \frac{1-s}{N}} = \frac{1}{s}
\]
Limitation of Amdahl’s Law

- Assumption for Amdahl’s law
  - Tasks exist that are perfectly parallelizable

- In reality, no task is perfectly parallelizable
  - Resources have to be shared, mostly have to be used serially
  - Dependencies between tasks exist and have to be communicated
  - Tasks have to exchange information
  - Work imbalances
Weak Scaling vs. Strong Scaling

• Gustafson’s law: Counterpart to Amdahl’s law
  – Addresses the assumption of a fixed data set, which Amdahl’s law is based on
  – Problem size changes for weak scaling, fixed runtime (sketched below)

<table>
<thead>
<tr>
<th>Workload</th>
<th>Time Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=1</td>
<td>𝑊₁</td>
</tr>
<tr>
<td>N=2</td>
<td>𝑊₁, 𝑊₂</td>
</tr>
<tr>
<td>N=4</td>
<td>𝑊₁, 𝑊₂, 𝑊₃, 𝑊₄</td>
</tr>
</tbody>
</table>

Strong scaling (𝑇(1)=const, workload stays the same)

<table>
<thead>
<tr>
<th>Workload</th>
<th>Time Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=1</td>
<td>𝑊₁</td>
</tr>
<tr>
<td>N=2</td>
<td>𝑊₁, 𝑊₂</td>
</tr>
<tr>
<td>N=4</td>
<td>𝑊₁, 𝑊₂, 𝑊₃, 𝑊₄</td>
</tr>
</tbody>
</table>

Weak scaling (𝑇(𝑁)=const, workload is increased with 𝑁)
Real-World Example: Weak Scaling

- In real world, you usually measure runtimes
  - Increase data set with increasing number of processors
- Perfect weak scaling: constant runtime among varying #processors
Performance Modelling – Overview

- Performance Metrics: How to quantify performance?
- Hardware Model: Processors, Nodes, Networks
- Model 1: Amdahl’s Law
- **Model 2: Roofline Model**
- Further Modelling Approaches
Roofline Model: Motivation

• Typical question in performance modelling: **What** is the performance bottleneck?
  – **Arithmetic operations** / Computing capability of processor
  – **Memory accesses**: bandwidth or latency (main memory, caches)
  – **Network accesses**: bandwidth or latency

• Example: STREAM Triad: $\tilde{a} = \tilde{b} + s \times \tilde{c}$
  – What could limit the performance of this code?

• How “expensive” is a single iteration of the loop?
  – 2 floating point operations (ADD and MULT)
  – 3 memory operations (2 LOADs and 1 STORE)
Roofline Model: Motivation

- How long does a FP operation take?
  - Typically just a few nanoseconds (< 5ns) for a modern processor
  - Even better: Different mechanisms on a CPU can achieve multiple FP ops. per cycle

- How long does a memory access take?
  - Latency: about 90ns for main memory (DRAM)

- How much memory can be read per cycle?
  - Bandwidth: typically a few bytes per cycle (DRAM)

- Code is memory-bound.

- How to systematically reason on a code’s bottleneck?
  → Roofline model
Peak Performance of a Processor

• Flop/s (also: FLOPS) – Floating point operations per second
  – Measures the number of floating-point (FP) operations which can be carried out per second for a given arithmetic operation on a given architecture
  – Dominant metric in HPC

• Theoretical peak performance $P_{\text{peak}} [\text{Flop/s}]$ of a processor:

$$P_{\text{peak}} = \# \text{cores} \cdot \text{frequency} \cdot \frac{\text{FP operations}}{\text{cycle}}$$

• Note: In the following “peak performance” always refers to FP operations with double precision.
  – “Double precision” means calculations with datatype double (typically 64 bits)
Peak Performance of a Processor – Examples

- Theoretical peak performance $P_{\text{peak}}$ [Flop/s] of a processor:

  $$P_{\text{peak}} = \text{#cores} \cdot \text{frequency} \cdot \frac{\text{FP operations}}{\text{cycle}}$$

- Examples:
  - Intel Xeon Platinum 8160, 1 core: $P_{\text{peak}} = 1 \cdot 2.1 \text{GHz} \cdot 32 \text{ Flop} = 67.2 \text{ GFlop/s}$
  - Intel Xeon Platinum 8160, 24 cores: $P_{\text{peak}} = 24 \cdot 2.1 \text{GHz} \cdot 32 \text{ Flop} = 1612.8 \text{ GFlop/s}$
  - CLAIX-2018 Node (2 x Intel Xeon Platinum 8160): $P_{\text{peak}} = 48 \cdot 2.1 \text{GHz} \cdot 32 \text{ Flop} = 3225.6 \text{ GFlop/s}$
Understanding Code Properties: Operational Intensity

• STREAM Triad: \( \vec{a} = \vec{b} + s \cdot \vec{c} \)

\[
\text{for}(i=0; \ i < N; \ ++i) \ { }
\begin{align*}
\text{a}[i] &= \text{b}[i] + s \cdot \text{c}[i]; \\
\end{align*}
\]

• Operations and accesses per loop iteration:
  – 2 arithmetic operations (floating-point, ADD and MULT)
  – 3 data transfers (2 Load, 1 Store)

• Operational intensity \( I \): Describes requirements of the code

\[
I = \frac{\text{arithmetic operations}}{\text{data transfers (LOAD,STORE)}} = \frac{2\text{ Flops}}{3\text{ Words}} = 0.66 \frac{\text{Flop}}{\text{Word}} = 0.083 \frac{\text{Flop}}{\text{Byte}}
\]

Assuming word size 64-bit

Scalar \( s \) kept in register

2x Load

2x FP Operations

Store

Scalar \( s \) kept in register
Classification of Typical Kernels

• Classification of \( \frac{\text{arithmetic operations}}{\text{data transfers}} \)
• Assume: \( N \) is problem size or (outer / inner) loop length

\[ O(N) \] arithmetic operations \[ O(N) \] data transfers
- scalar product, vector addition, sparse matrix-vector multiplication
  - Typical “memory-bound” algorithm

\[ O(N^2) \] arithmetic operations \[ O(N^2) \] data transfers
- dense matrix-vector multiply, matrix addition
  - Also “memory-bound”, can profit from caching (depending on the problem)

\[ O(N^x) \] arithmetic operations \( x > 2 \) \[ O(N^2) \] data transfers
- dense matrix-matrix multiplication, dense matrix diagonalization
  - Most favorable case, computation outweighs traffic for large \( N \)
Roofline Model

• Roofline: Relate peak performance $P_{peak}$ and max. memory bandwidth $b_S$ to operational intensity $I$ of a code → Result is the **achievable performance** $P$

  $$P = \min(I \cdot b_S, P_{peak})$$

$P_{peak}$: Peak performance [Flop/s]
$I$: Operational intensity [Flop/word] or [Flop/byte]
$b_S$: Achievable bandwidth over slowest data path [words/s] or [byte/s]

• The higher $I$, the higher the number of arithmetic operations per data transfer.
Roofline Plot Example: Different Slowest Data Paths

STREAM Triad Kernel (assuming DRAM used)
Roofline Plot Example: Different Peak Performances

Roofline Model
Single Core, Intel Xeon Skylake Platinum (8160)

If the available arithmetic units cannot be optimally used (no SIMD, no FMA), $P_{\text{peak}}$ is much smaller.
Roofline Model – Limitations

- Roofline only models the slowest data path
  - Caching effects not considered
- Memory access latency is ignored

- Further reading (extensions of Roofline)
  - Execution-Cache-Memory (ECM) model (https://doi.org/10.1002/cpe.3180)
  - “Cache-aware roofline model: Upgrading the loft” (https://doi.org/10.1109/L-CA.2013.6)
Performance Modelling – Overview

- Performance Metrics: How to quantify performance?
- Hardware Model: Processors, Nodes, Networks
- Model 1: Amdahl’s Law
- Model 2: Roofline Model
- Further Modelling Approaches
Further Modelling Approaches

• Only covered two basic modelling approaches here
• Modelling network performance: LogP model ([https://doi.org/10.1145/155332.155333](https://doi.org/10.1145/155332.155333))
  • Four basic parameters (Latency L, overhead o, gap g, P number of processors) to predict runtime of network communication
• Modelling cache behavior: See Roofline extensions (e.g., ECM)
• Tool support (*presented later today*)
  • Intel Advisor: Roofline Automation  
  • Extra-P (TUDa): Understanding scaling behavior based on extrapolation  
    [https://github.com/extra-p/extrap](https://github.com/extra-p/extrap)
Summary

• Performance Modelling: Understand the interaction between hardware and software
  • Hardware model: Processors, nodes, networks
  • Software model: Understand code properties (flops, memory accesses, …)

• Amdahl’s Law: Model the scaling behavior of applications

• Roofline model: Determine bottleneck of code execution (compute-bound or memory-bound) and estimate achievable performance