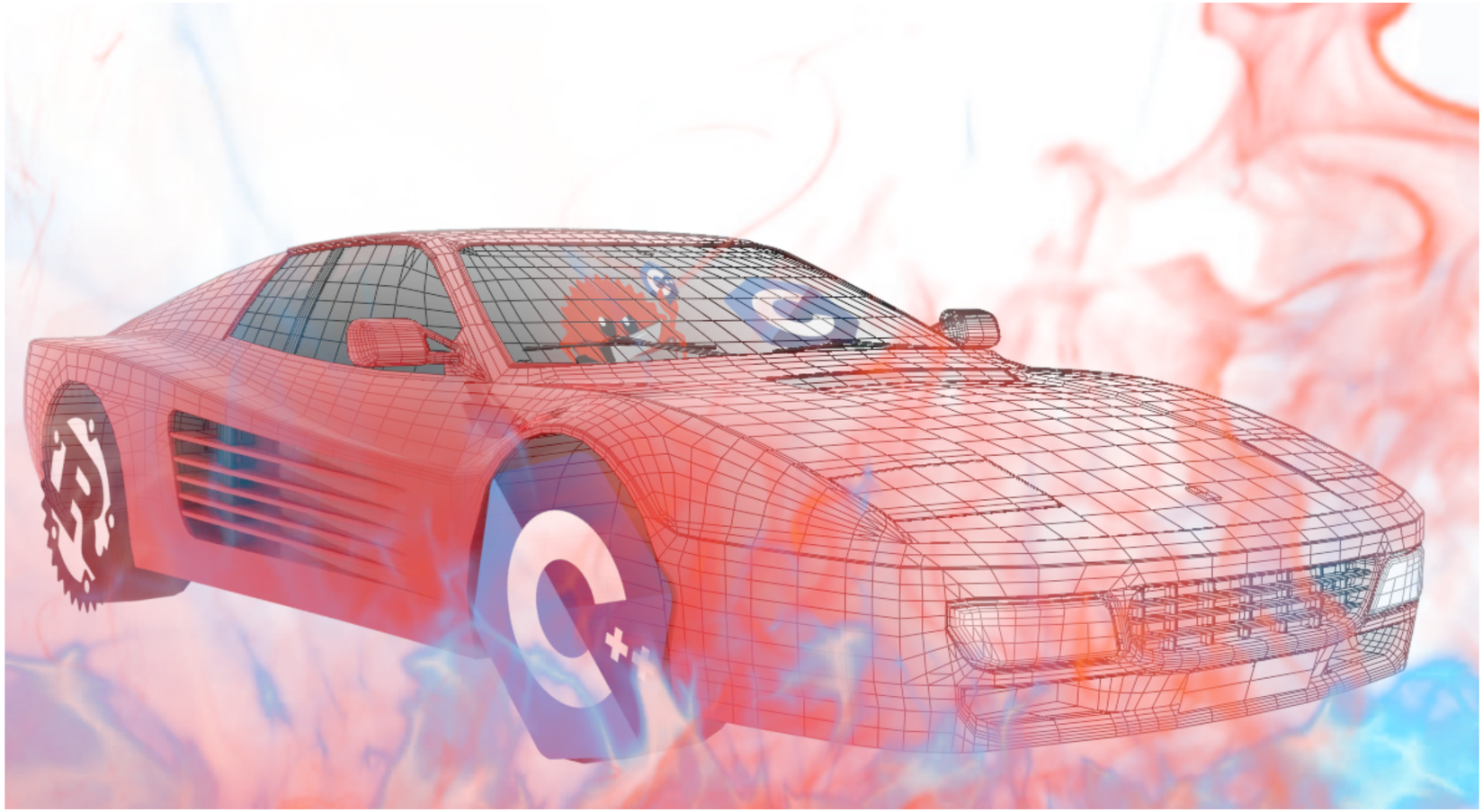
Effizientes Programmieren in C, C++ und Rust

Performance Modelling

Colin Bretl, Nikolai Maas, Peter Maucher | 24.10.2024



Questions of Today

1. What did you like best: C, C++ or Rust?
2. How long does it take to get a carrot?
3. How hard is it to transpose a matrix?
4. What do airports and processors in common?

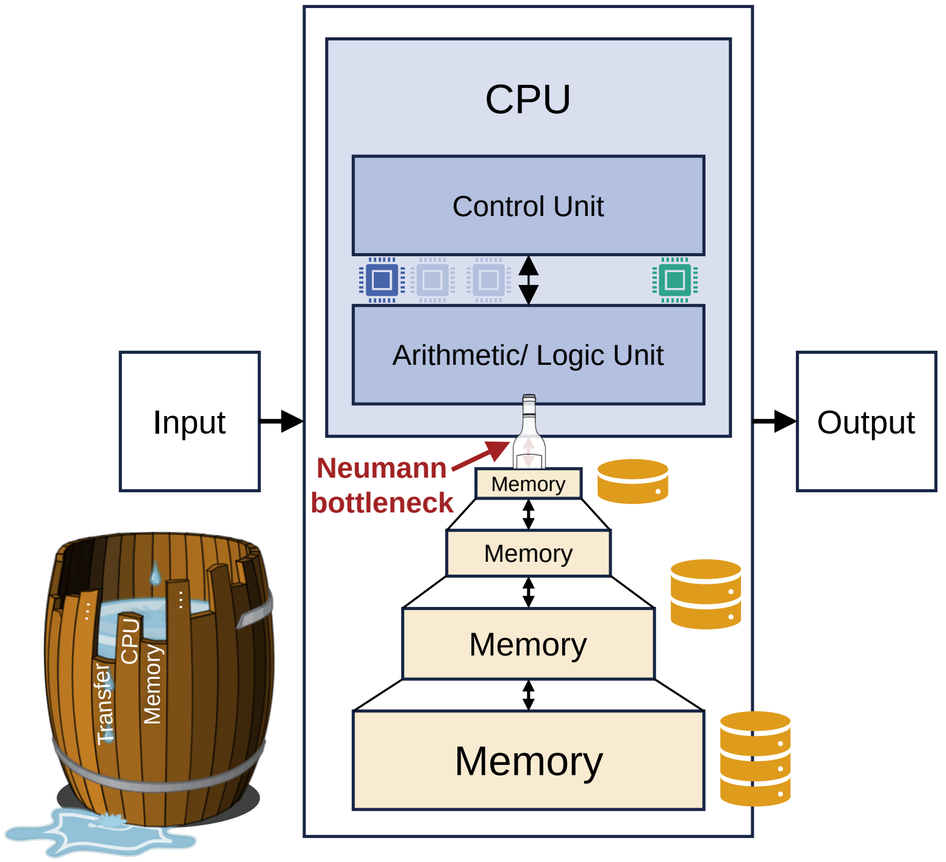
Table of Contents

Boarding

Boarding

What do we deal with?

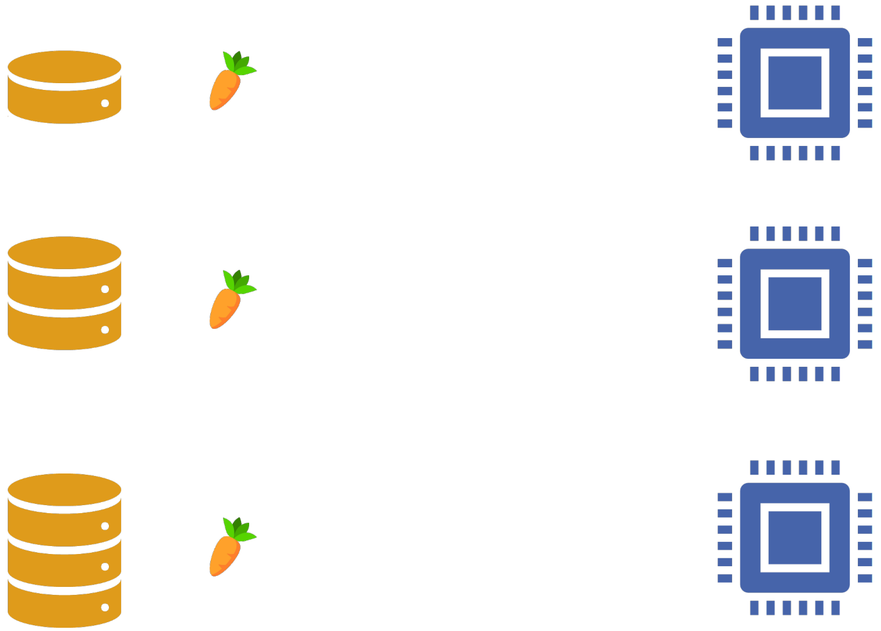
- Von-Neumann architecture
  - CPU and memory connected by a bus
  - Single core
  - Single threaded
- The Neumann bottleneck and how it is addressed
  - Localize (Caching)
  - Parallelize (Vectorization)
  - Pipeline (Out-of-order execution)
- Law of the minimum ("Liebig's Barrel")



Boarding

What do we deal with?

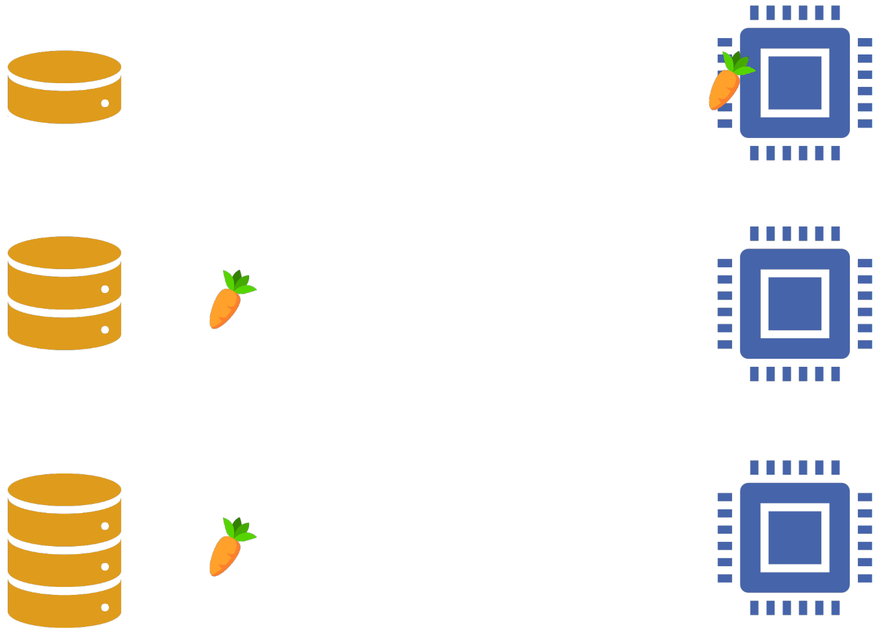
- Von-Neumann architecture
  - CPU and memory connected by a bus
  - Single core
  - Single threaded
- The Neumann bottleneck and how it is addressed
  - Localize (Caching)
  - Parallelize (Vectorization)
  - Pipeline (Out-of-order execution)
- Law of the minimum ("Liebig's Barrel")



Boarding

What do we deal with?

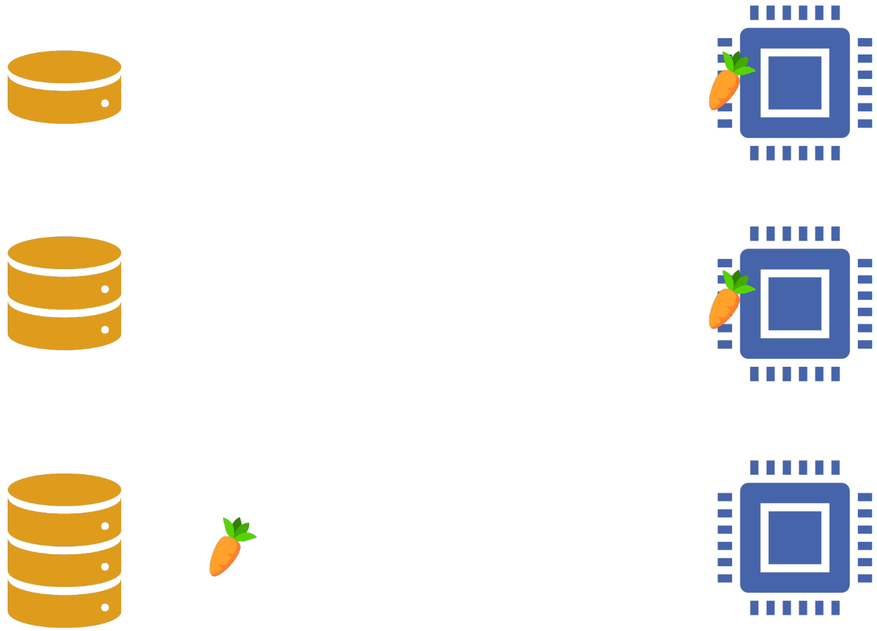
- Von-Neumann architecture
  - CPU and memory connected by a bus
  - Single core
  - Single threaded
- The Neumann bottleneck and how it is addressed
  - Localize (Caching)
  - Parallelize (Vectorization)
  - Pipeline (Out-of-order execution)
- Law of the minimum ("Liebig's Barrel")



Boarding

What do we deal with?

- Von-Neumann architecture
  - CPU and memory connected by a bus
  - Single core
  - Single threaded
- The Neumann bottleneck and how it is addressed
  - Localize (Caching)
  - Parallelize (Vectorization)
  - Pipeline (Out-of-order execution)
- Law of the minimum ("Liebig's Barrel")



Boarding

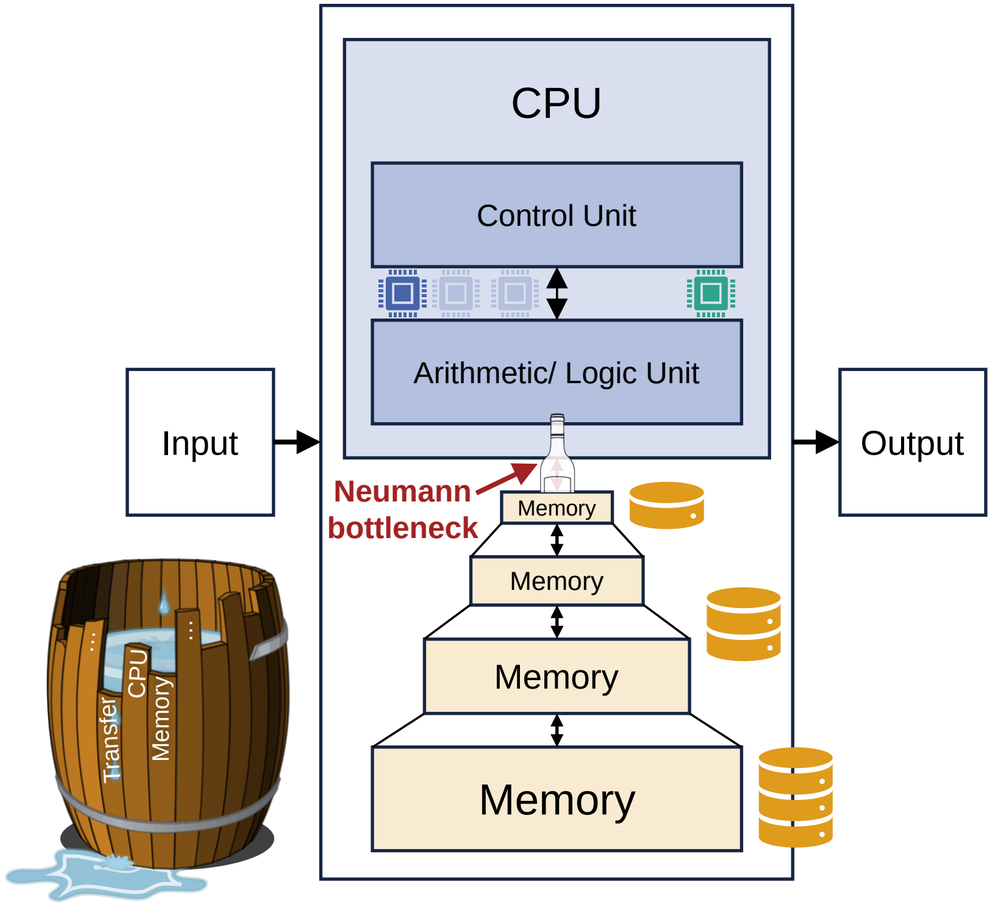
What do we deal with?

- Von-Neumann architecture
  - CPU and memory connected by a bus
  - Single core
  - Single threaded
- The Neumann bottleneck and how it is addressed
  - Localize (Caching)
  - Parallelize (Vectorization)
  - Pipeline (Out-of-order execution)
- Law of the minimum ("Liebig's Barrel")

Boarding

What do we aim for?

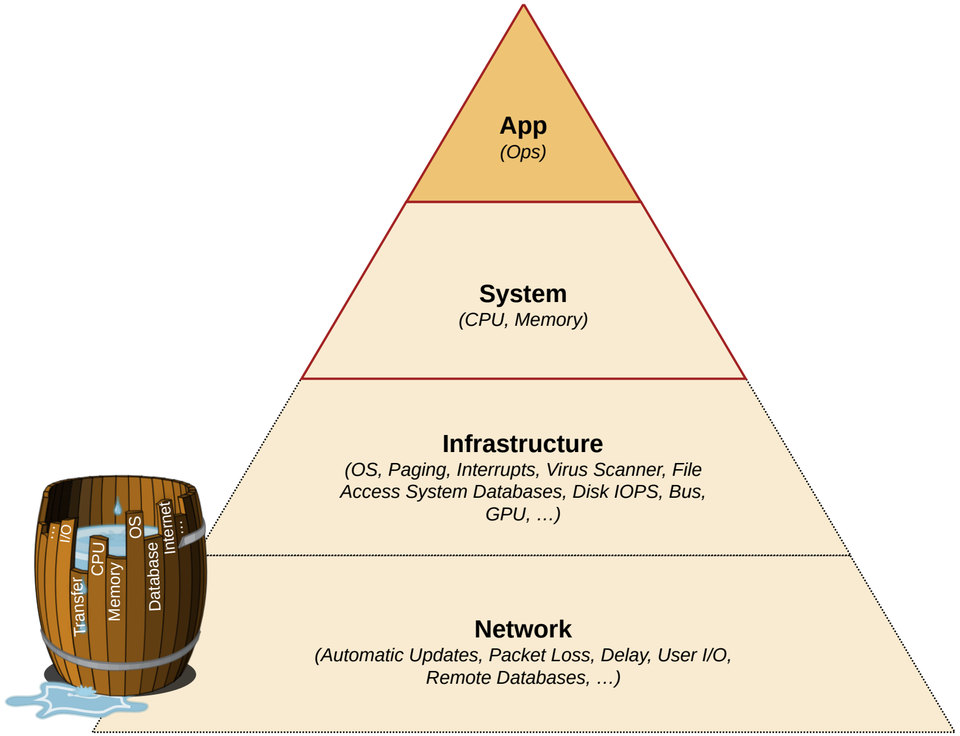
- Performance!
  - *"How successful something is or how well it is doing" (Collins Dictionary)*
  - *"How well a machine does a piece of work or an activity" (Cambridge Dictionary)*
  - *"The amount of useful work accomplished by a computer system" (Wikipedia)*
- Processing speed
- Memory consumption
- Bandwidth
- Throughput
- Overall utilization
- Response time
- Power consumption
- Environmental impact
- ...



Boarding

The Bigger Picture

1. System: CPU and Memory
2. Infrastructure: Other computing resources
3. Network
4. (Business) Process
5. Human-Machine Interface



The Bigger Picture

1. System: CPU and Memory
2. Infrastructure: Other computing resources
3. Network
4. (Business) Process
5. Human-Machine Interface

Motivation

Motivation

Why to Model

- There are unwritten algorithms and unbuilt systems whose performance we want to predict
- Scientific method: Falsification requires a theory or model that can be falsified

ALL MODELS ARE WRONG...

- The real world is too complicated to understand
- The real world is too complex and chaotic to describe
- Most details are not relevant in most situations
- Don't get fooled into thinking that some formula or diagram explains your situation precisely

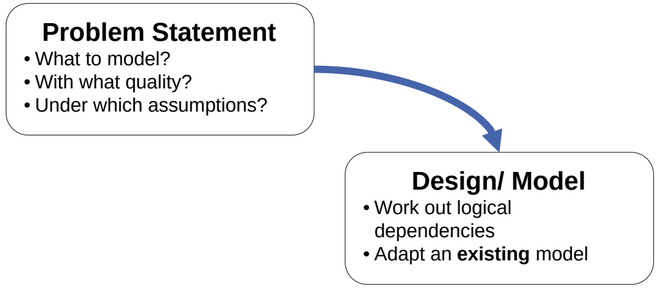
...BUT SOME ARE USEFUL!

- Simplified realities are easier to understand
- Simplified realities are easier to describe
- Ockham's Razor is considered good scientific praxis

Motivation

How to Model

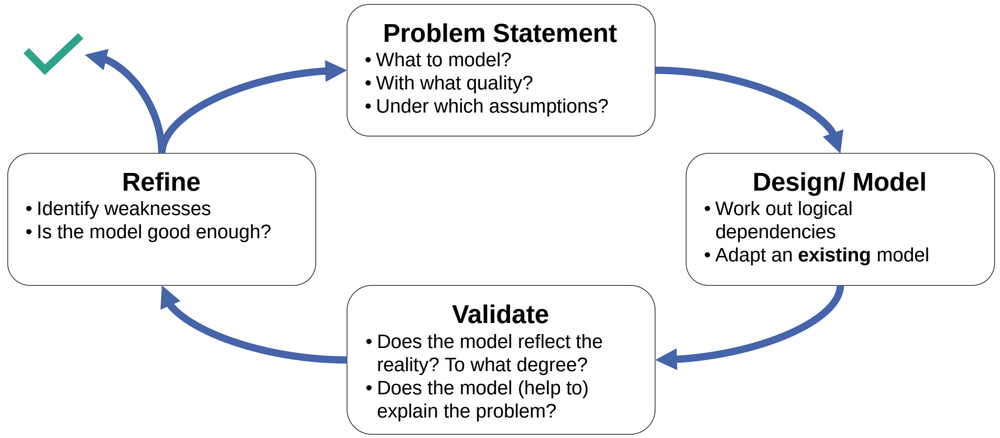
1. Problem Statement: identify and quantify objectives, provide clear assumptions
   - How fast is my code asymptotically?
   - How fast is my code on a given machine?
   - How fast can my code get on a given machine?
2. Design: establish logical dependencies between qualitative and quantitative factors that affect the problem
   - Prefer to adapt an existing model rather than building one from scratch
   - Choose an appropriate existing model out of a few options



Motivation

How to Model

3. Validation: determine to what degree the model reflects the problem
4. Refinement: adapt the model or change it if necessary, reconsider the problem statement
   1. Was your problem statement clear enough?
   2. Was your problem statement what you had in mind?
5. Acceptance: Stop modelling if you found the "easiest model that does the job". Beware of overfitting

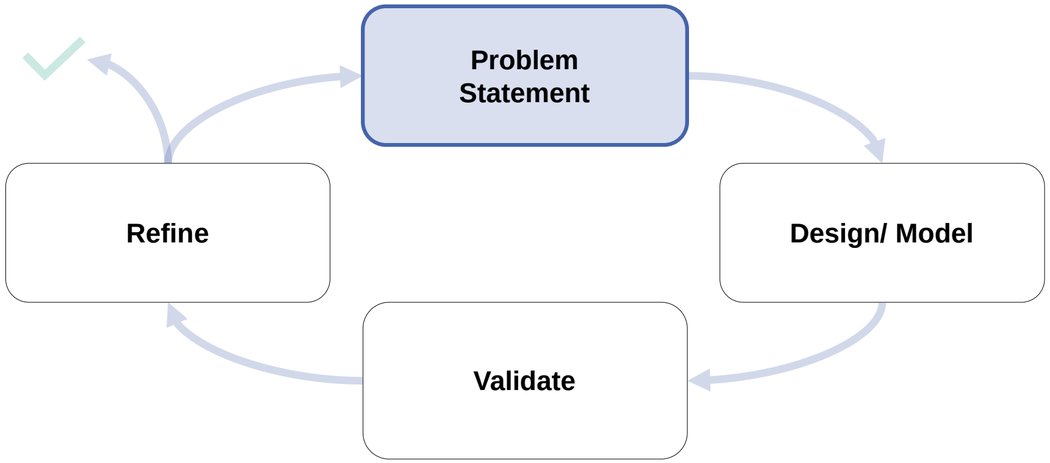


There is no general recipe!

Motivation

Problem Statement

- Question: "What performance can we expect for algorithm $\mathcal{A}$ on machine $\mathcal{M}$?"
  - $\mathcal{M}$: Recent-ish desktop CPU
    - capable of instruction-level parallelism (pipelining, out-of-order execution, branch prediction, ...)
    - E.g., Intel Pentium 4
  - $\mathcal{A}$: Matrix transposition
    - `(double matrix[N][N])`
    - `swap(matrix[r][c], matrix[c][r])`



Motivation

Problem Statement

- Question: "What performance can we expect for algorithm $\mathcal{A}$ on machine $\mathcal{M}$?"
  - $\mathcal{M}$: Recent-ish desktop CPU
    - capable of instruction-level parallelism (pipelining, out-of-order execution, branch prediction, ...)
    - E.g., Intel Pentium 4
  - $\mathcal{A}$: Matrix transposition
    - `(double matrix[N][N])`
    - `swap(matrix[r][c], matrix[c][r])`

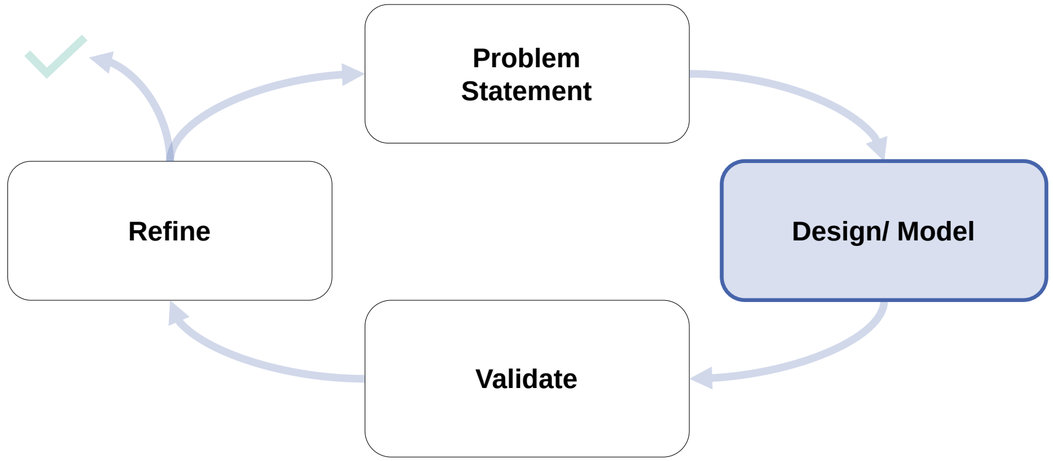
```c
const int N = 64;
void transpose(double m[SIZE][SIZE]) {
    int r, intC; double tmp;
    for (c = 1; r < N; ++r) {
        for (c = 0; c < r; ++c) {
            tmp = m[r][c];
            m[r][c] = m[c][r];
            m[c][r] = tmp;
        }
    }
}

int main() {
    alignas(64) double matrix[N][N];
    transpose(matrix);
}
```

Motivation

Design

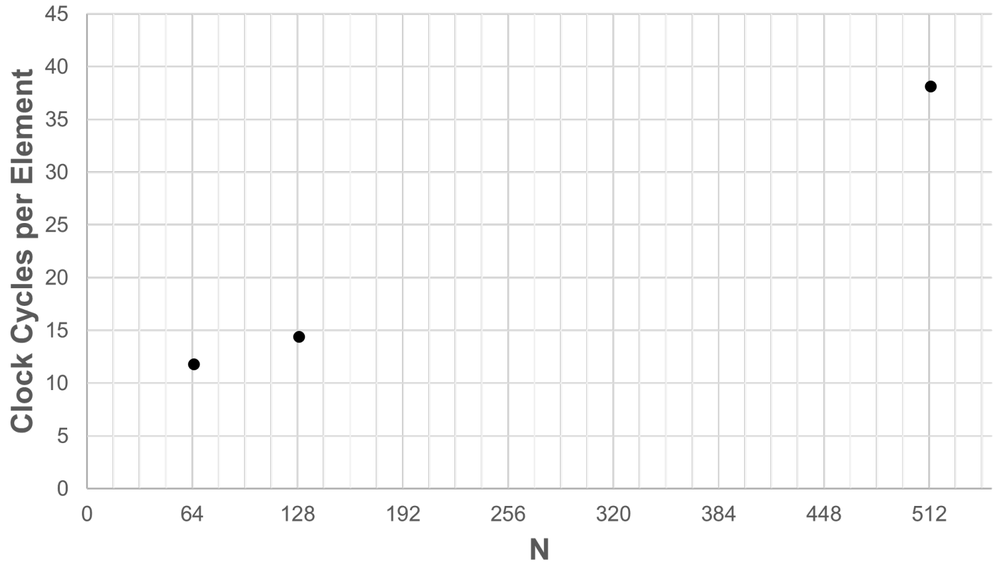
- (Naive) Measure/ Model for performance:
  - Big O-Notation: we expect $\mathcal{O}(\frac{1}{2}n^2)$
    - Helpful for classification/ rough guideline (large n)
  - CPU (Execution) Time
    - `auto start = now();`
    - `transpose(m);`
    - `auto stop = now();`
  - CPU time = #instructions * CPI / clock rate
    - Different instructions need different number of clock cycles (e.g. MOV vs. IMUL [Ag4])
    - CPI [#]: average number of clock cycles per instruction
    - Clock rate [Hz]: frequency of pulse generation

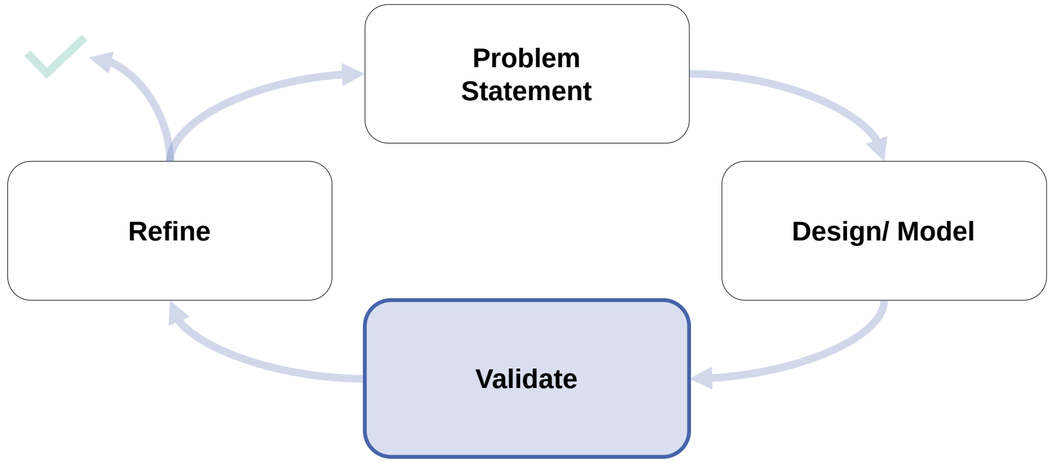


Motivation

Validate

- Few data points but looks non-linear

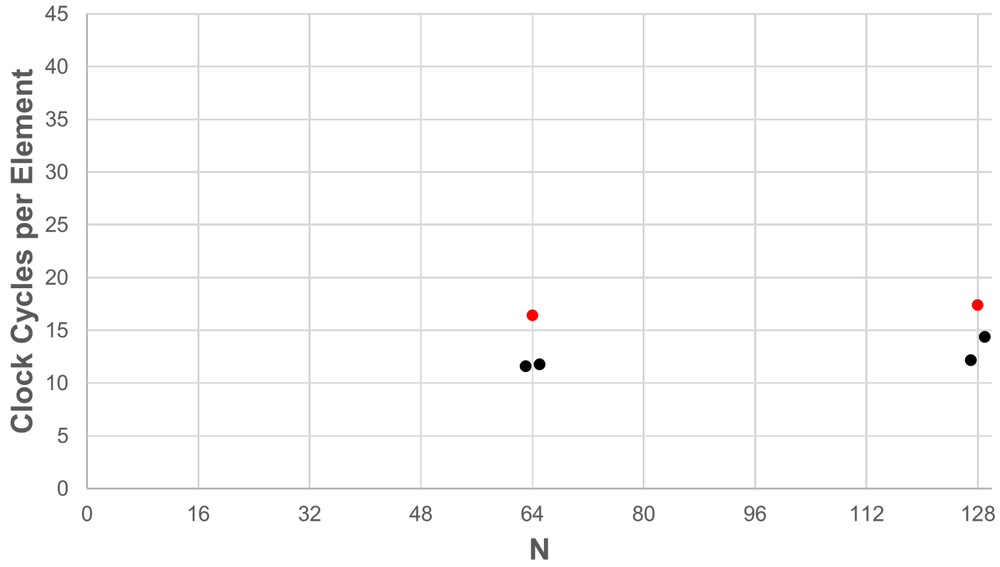


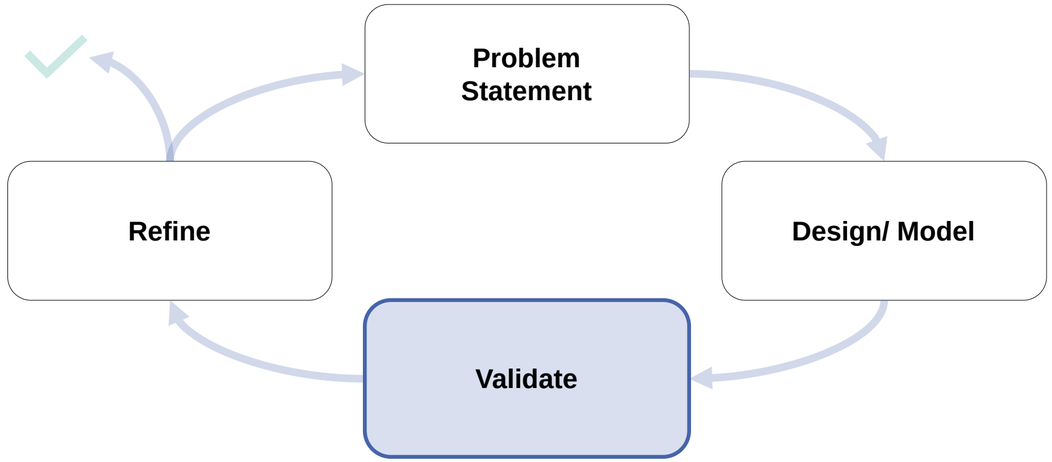


Motivation

Validate

- Few data points but looks non-linear
- Maybe some noise...?

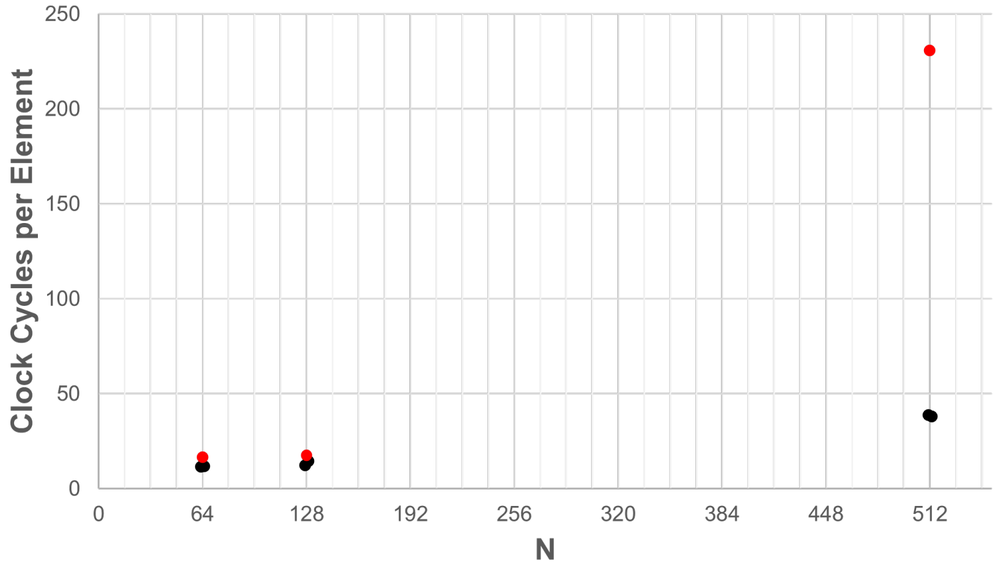


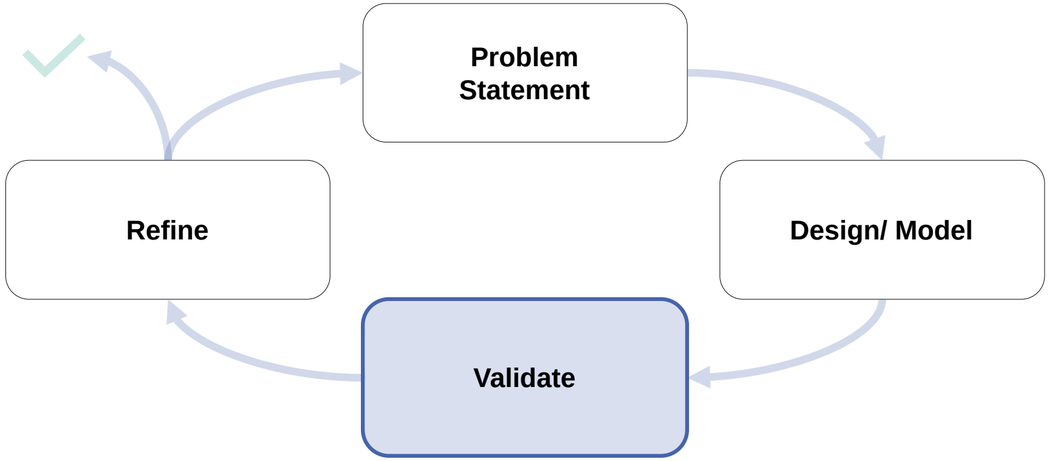


Motivation

Validate

- Few data points but looks non-linear
- Maybe some noise...?

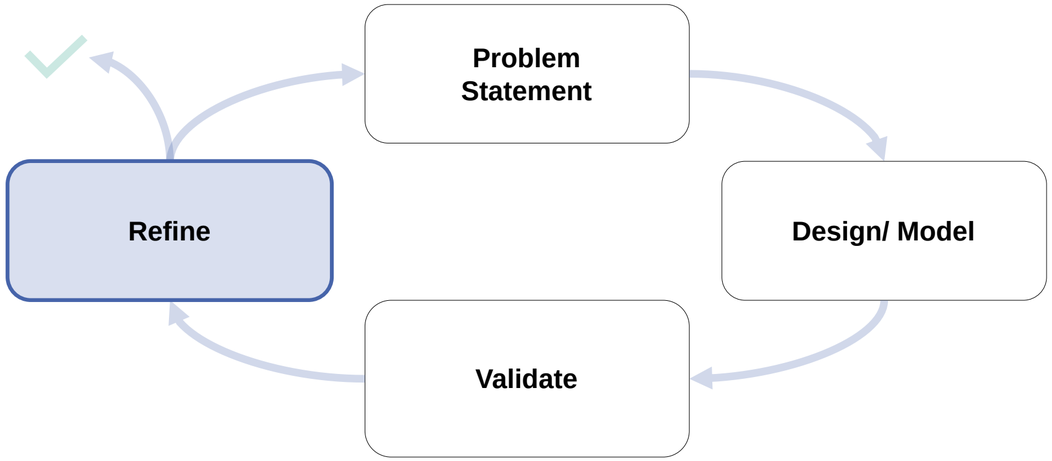




Motivation

Refine

- Reconsider question: "What performance can we expect for algorithm $\mathcal{A}$ on machine $\mathcal{M}$?"
- Model does not explain reality too well, number of instructions is not the only contributing factor
- What happens:
  - Cache lines follow matrix rows
  - Consider `N` = 64
  - L1 Cache: 8192 Bytes, 4 ways, line size 64
  - `sizeof(double)` = 8 Bytes → 8 `doubles` per line
  - `sizeof(row)` = 64 x 8 Bytes x 512 Bytes
  - Critical stride: 8192 / 4 = 2048 Bytes = 4 matrix rows
  - Exc.: Consider r = 28, see what happens
  - Even more dramatic: L2 limit (512 kB, 8 ways, critical stride 64 kB = 16 lines of 512x512 matrix)

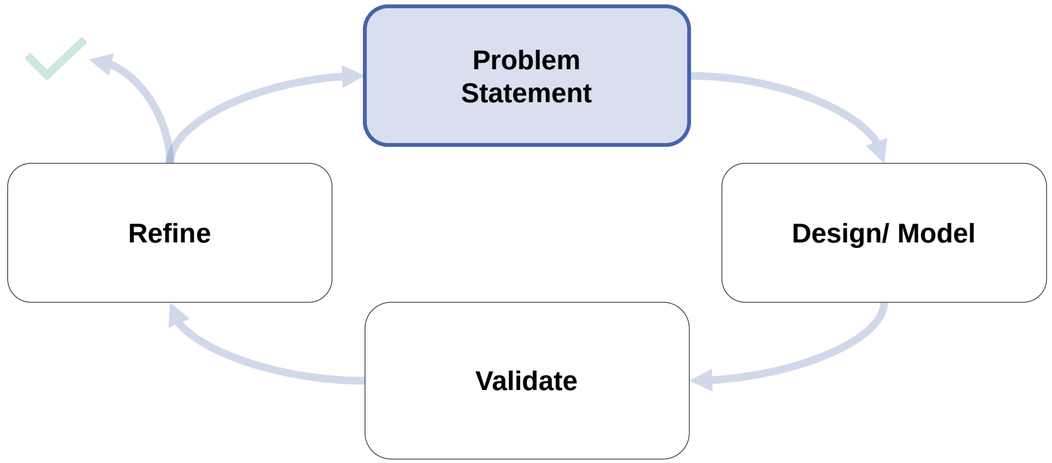


The Roofline Model

The Roofline Model

Problem Statement

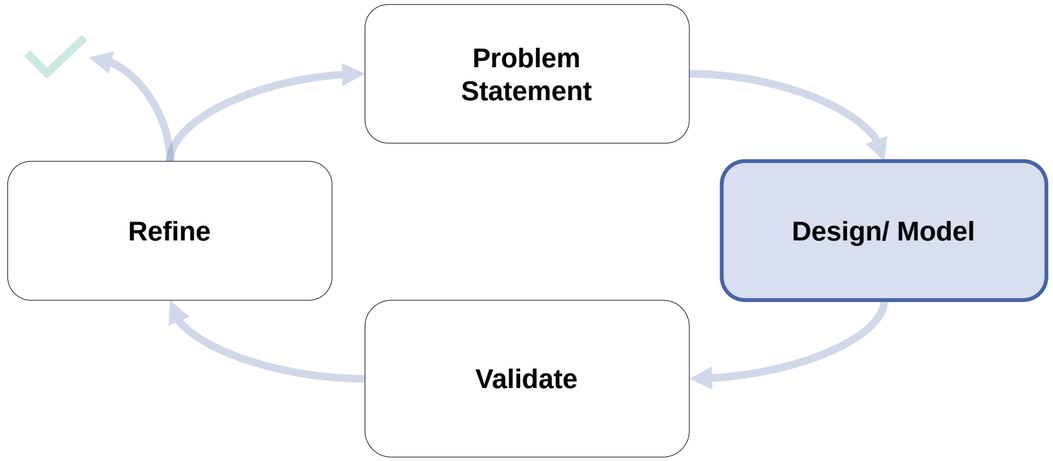
- Question: "What performance can we expect for algorithm $\mathcal{A}$ on machine $\mathcal{M}$ in terms of processor performance and off-chip memory traffic?"



The Roofline Model

Model

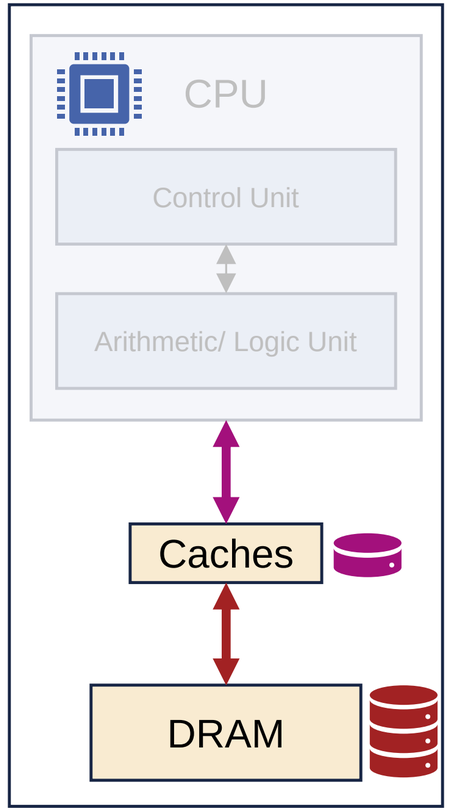
- Question: "What performance can we expect for algorithm $\mathcal{A}$ on machine $\mathcal{M}$ in terms of processor performance and off-chip memory traffic?"
- Processor performance
  - Average instruction time: $\mathcal{A}$-dependent
  - (Million) instructions per second (MIPS): $\mathcal{M}$-dependent
  - (Million) floating point operations per second ((M)FLOPS): good compromise



The Roofline Model

Model

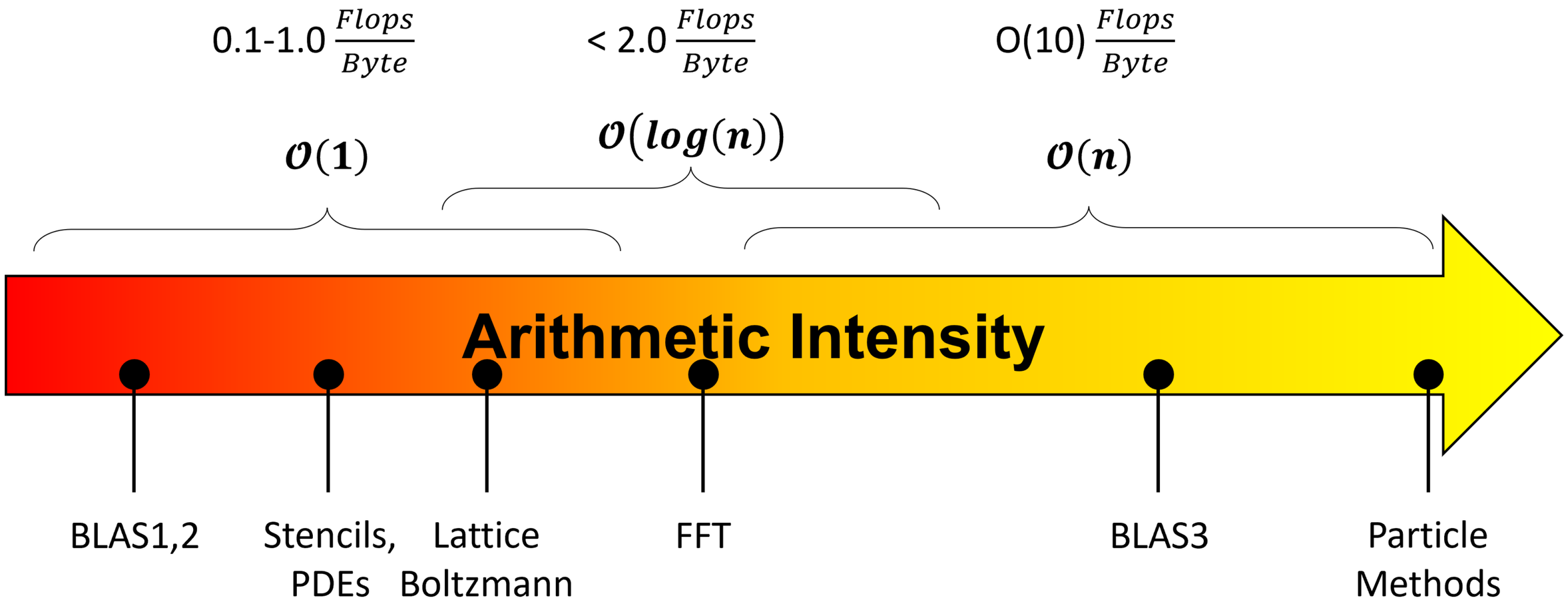
- Question: "What performance can we expect for algorithm $\mathcal{A}$ on machine $\mathcal{M}$ in terms of processor performance and off-chip memory traffic?"
- Processor performance
  - Average instruction time: $\mathcal{A}$-dependent
  - (Million) instructions per second (MIPS): $\mathcal{M}$-dependent
  - (Million) floating point operations per second ((M)FLOPS): good compromise
- Memory traffic:
  - Arithmetic intensity = Ops per byte ($\mathcal{A}$-dependent, "how much" need to be actually computed; e.g. matrix transposition)
  - Operational intensity = Ops per byte of DRAM traffic after filtering by the cache hierarchy
  - Machine balance



The Roofline Model

On (Arithmetic) Intensity

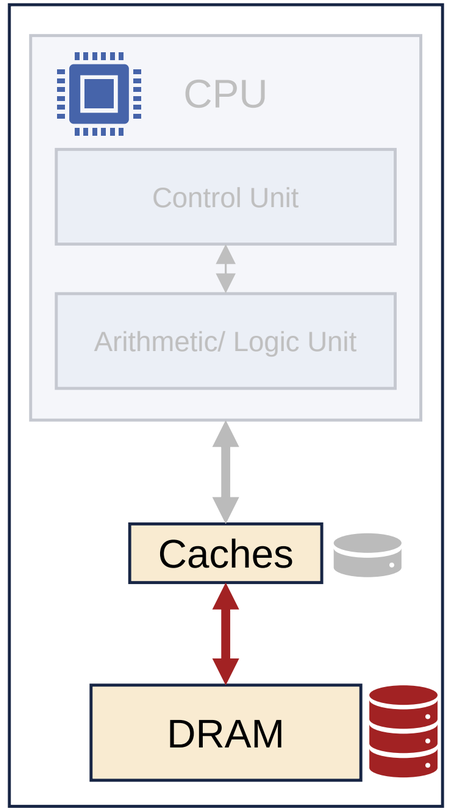
- How common problems rank:



The Roofline Model

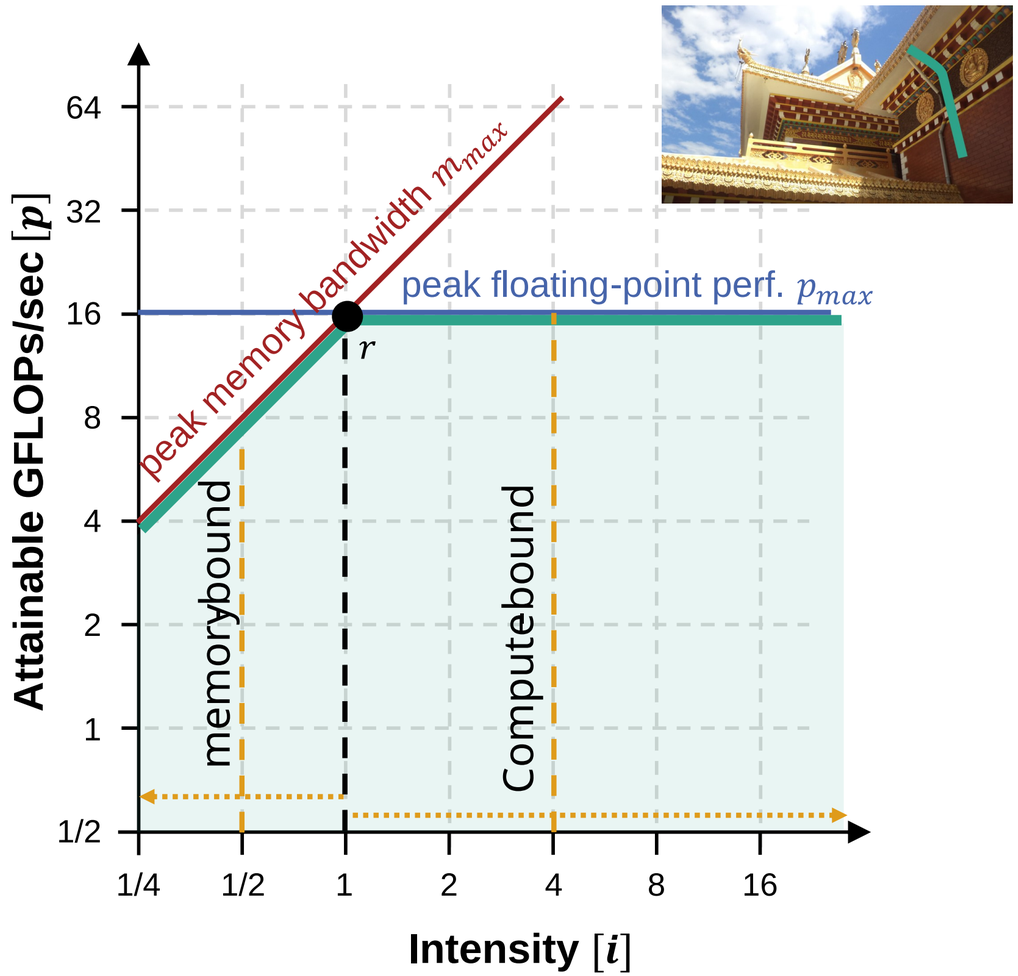
Model

- Processor performance: FLOPS
- Memory traffic: (Operational) Intensity
- Assumptions:
  - Floating point performance dominates (vector addition, rotation, shifts, ... in innermost loop)
  - Branch mispredictions are rare
  - Non pipelined instructions are rare
  - Proper prefetching
  - Maximizing locality minimizes communication
  - ...
- Measuring:
  - Manufacturer's information, performance manual
  - (Micro-)Benchmarking, e.g., STREAM for memory bandwidth



The Roofline Model

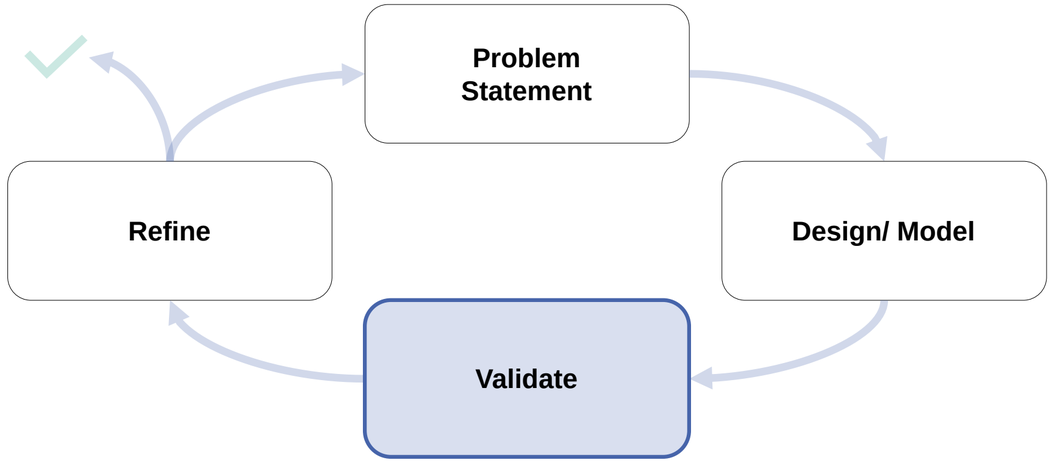
- Processor performance: FLOPS
- Memory traffic: (Operational) Intensity
- Relationships
  - Actual floating point performance cannot exceed peak performance (p_{max})
  - Bytes per second = $\frac{GFLOPs/sec}{GFLOPs/Byte}$ (m_{max})
  - Attainable GFLOPs $p = min(p_{max}, m_{max} * i)$ ( )
- Be wary of logscale!
- **Ridge point** r at i_{max} of p_{max} and m_{max}:
  - $i < i_{max}$: memory-bound
  - $i > i_{max}$: compute-bound



The Roofline Model

Validate

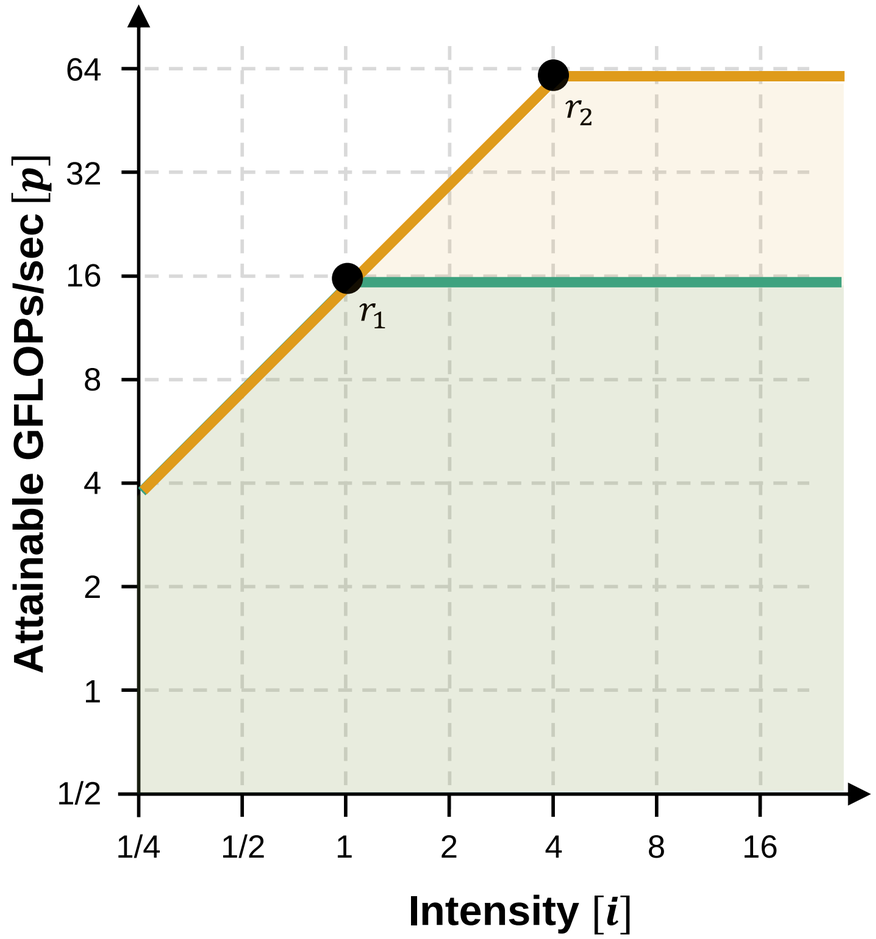
- Machine $\mathcal{M}_1$ (2 cores) 
- Machine $\mathcal{M}_2$ (4 cores) 
- Same socket, same peak memory bandwidth
- Where is the new ridge point?



The Roofline Model

Validate

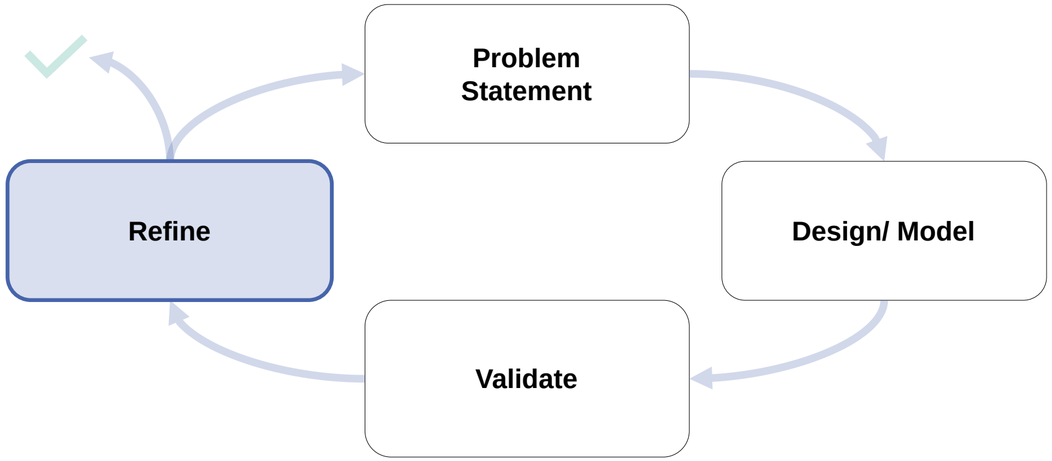
- Machine $\mathcal{M}_1$ (2 cores)
- Machine $\mathcal{M}_2$ (4 cores)
- Same socket, same peak memory bandwidth
- Where is the new ridge point?



The Roofline Model

Refine

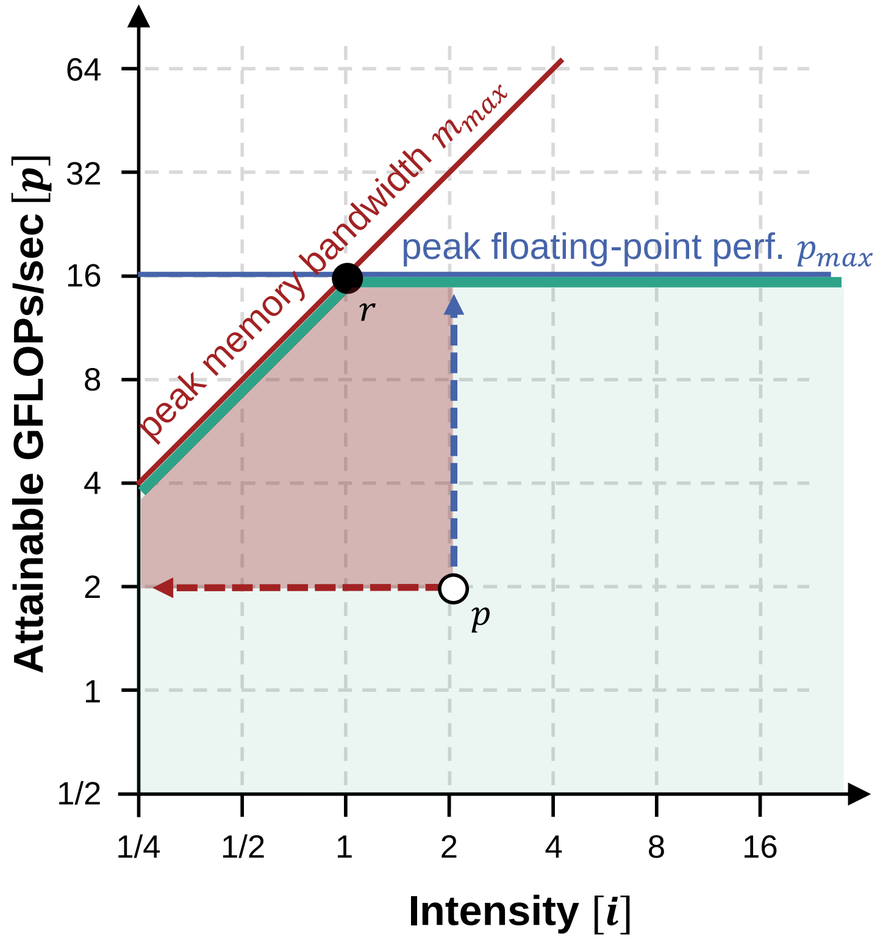
- So far: upper bound to performance



The Roofline Model

Refine

- So far: upper bound to performance
- What optimizations to perform if $p \ll r$?
  - Improve instruction level parallelism
  - Use SIMD
  - Balance floating-point operation mix
  - Restructure loops for unit stride accesses
  - Ensure memory affinity
  - Use software prefetching
  - ...
- Consider these as additional ceilings you cannot break through without that particular optimization
- Proper estimation (micro-benchmarking)



The Roofline Model

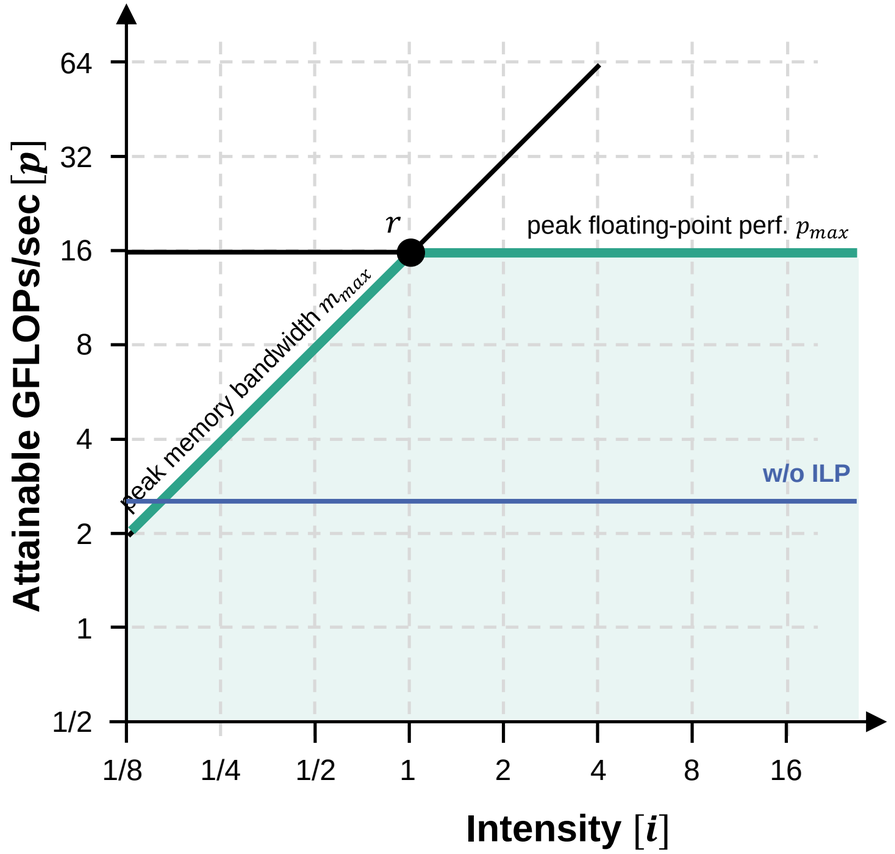
Improve ILP

- Pipelining can increase frequency but it can also increase latency (bubbles)
- Programmer and compiler must ensure independent instructions are issued in sequence
- Break up (long) dependency chains:
  - Unoptimized:

```
float a, b, c, d, y;
y = (((a + b) +  c) + d);
```

  - Optimized:

```
y = ( (a + b) + (c  + d));
```



The Roofline Model

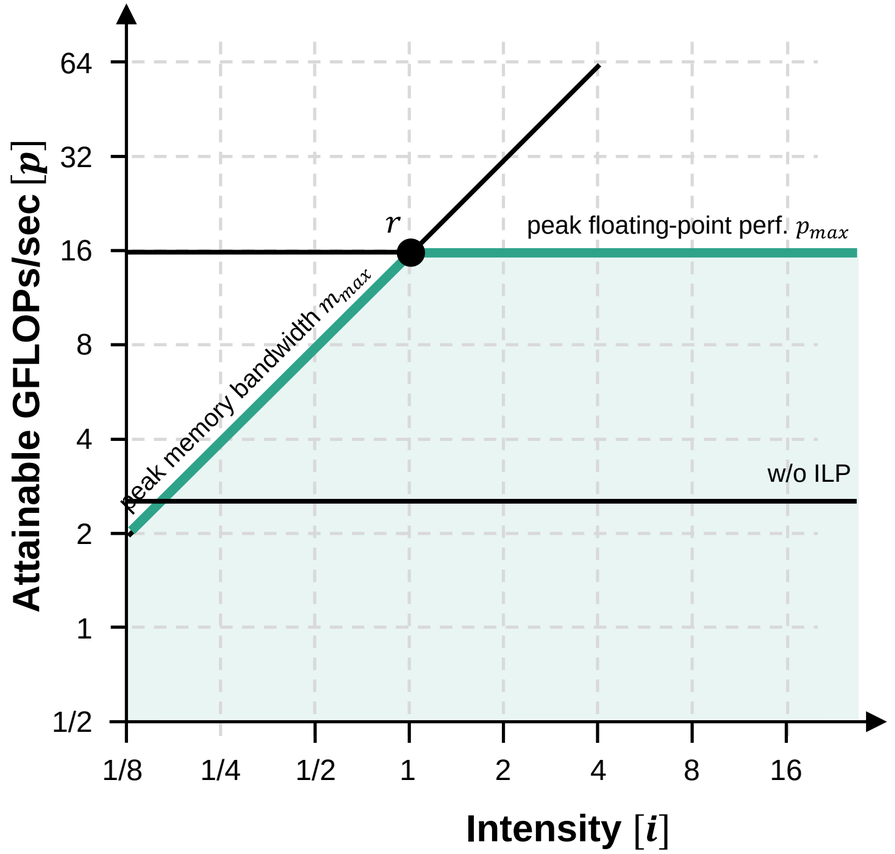
Improve ILP

- Pipelining can increase frequency but it can also increase latency (bubbles)
- Programmer and compiler must ensure independent instructions are issued in sequence
- Loop unrolling:
  - Unoptimized:

```
float list[N]; float sum = 0;
for (int i=0; i < N; ++i) {
    sum += list[i];
}
```

  - Optimized:

```
float list[N];
float sum1 = 0, sum2 = 0;
for (int i=0; i < N-1; ++i) {
    sum1 += list[i];
    sum2 += list[++i]; }
sum1 += sum2;
```

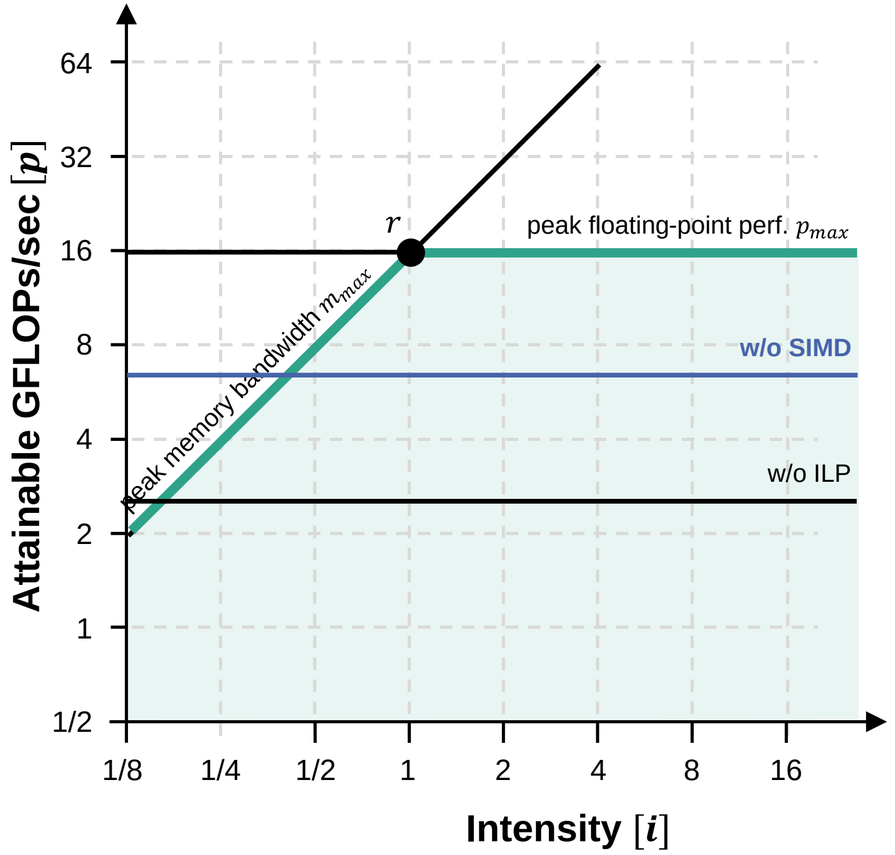


The Roofline Model

Use SIMD

- SIMD instructions allow performing operations on all elements of a vector simultaneously
- Requires large data sets and parallel execution, e.g.
  - Image processing
  - Sound processing
  - Operations on vectors, matrices, ...
- ```
  #include <stdio.h>
  typedef int v4sf __attribute__
  ((mode(V4SF)));

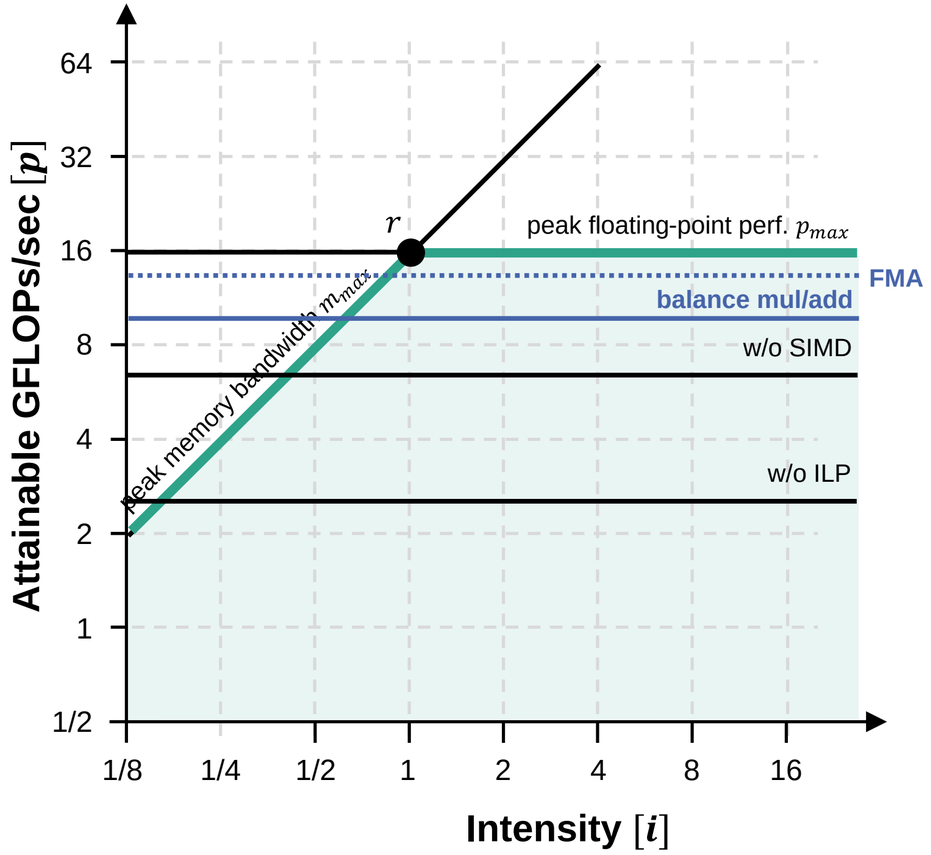
  union f4vector {
    v4sf v;
    float f[4];
  }
  ```


  ```

The Roofline Model

Balance floating mul/add

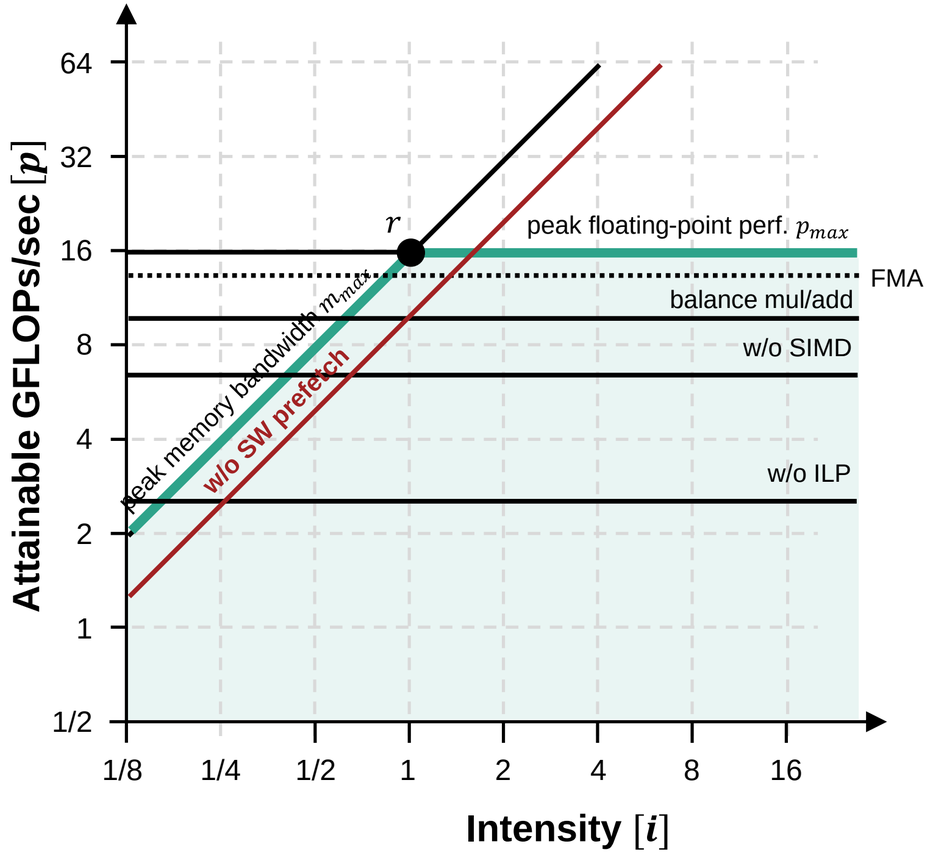
- Modern mircroprocessors have several execution units (e.g., 2+ integer units, 1-2 floating point addition, 1-2 floating point multiplication)
- Aim for balanced mix of + and *
- Example:
  - Unoptimized:
```
for (int i = 0; i < N; ++i) {
    c[i] = 2.f * a[i] * b[i];
}
```
  - Optimized:
```
for (int i = 0; i < N; ++i) {
    float tmp = a[i] * b[i];
    c[i] = tmp + tmp;
}
```
- Fused multiply-add FMA (i.e., `a * b + c` in one instruction)



The Roofline Model

SW prefetching

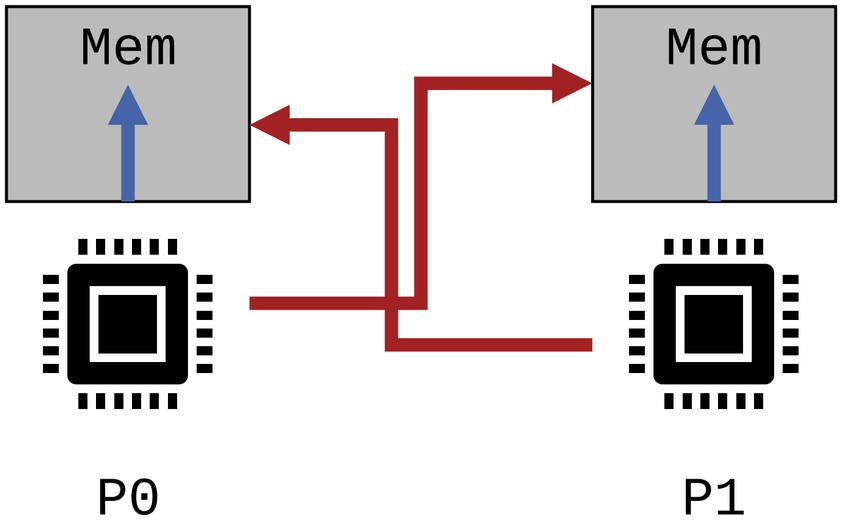
- Prefetch instructions initiate a load of a cache line into the cache but do not stall waiting for the data to arrive
  - Must happen early enough to arrive in the next cache level on time
  - Must not happen too late to be evicted before usage
- Modern processors do this automatically, not much need for doing it manually

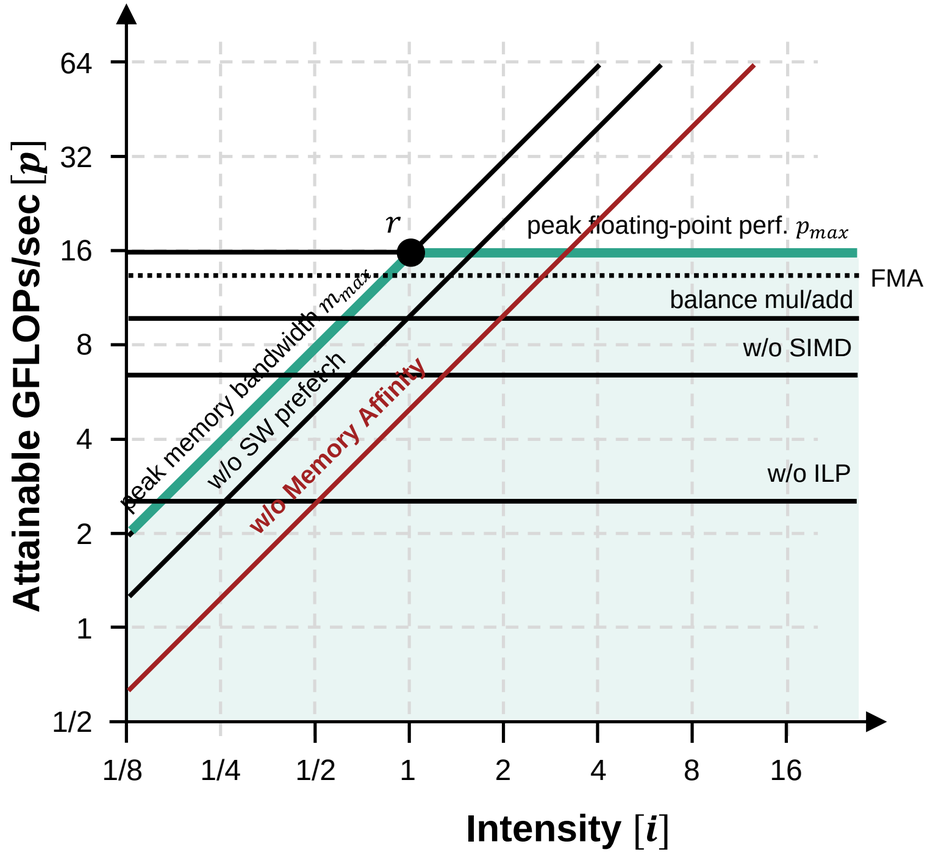


The Roofline Model

Memory Affinity

- On multicore machines, consider local data (per core) and remote data
  - Local data is accessed faster
  - Remote data is accessed slower
- Accessing remote data can also cause invalidations, ...
- Make areas of memory (pages, ...) local to specific processors to minimize memory traffic





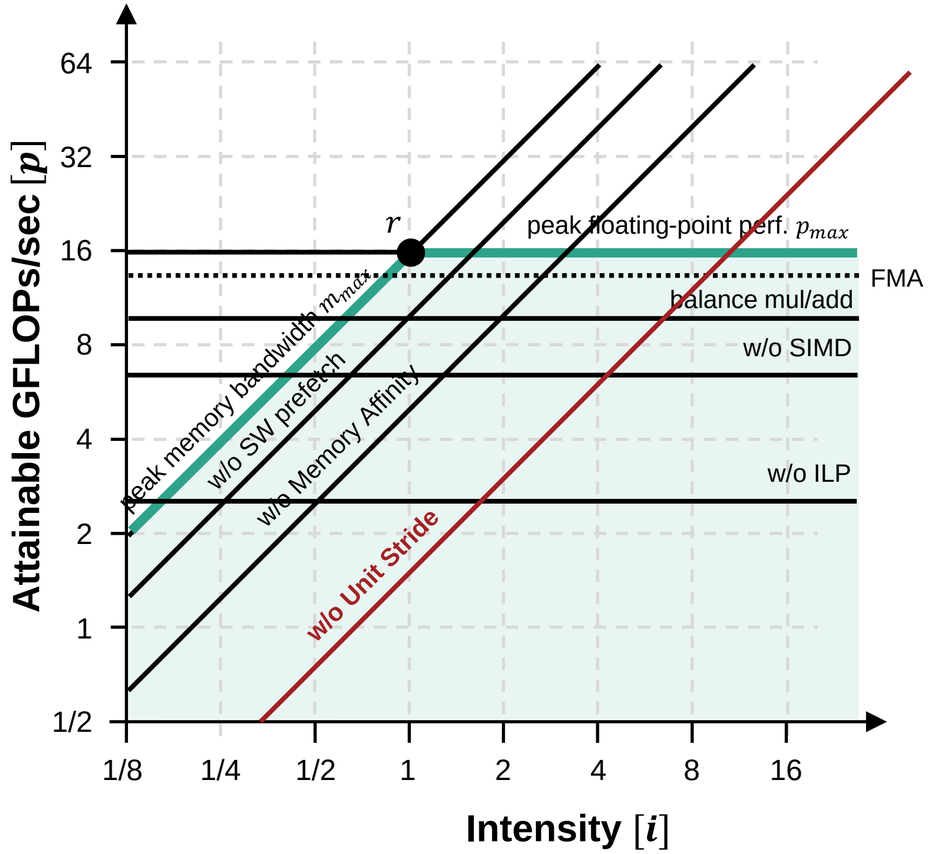
The Roofline Model

Unit Stride Access

- Consider

```
for (int i = 0; i < 4; ++i) {
    for (int j = 0; j < 4; ++j) {
        a[j][i] = b[j][i]*c[j][i];}}
```

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16



In memory:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Access with stride 4:

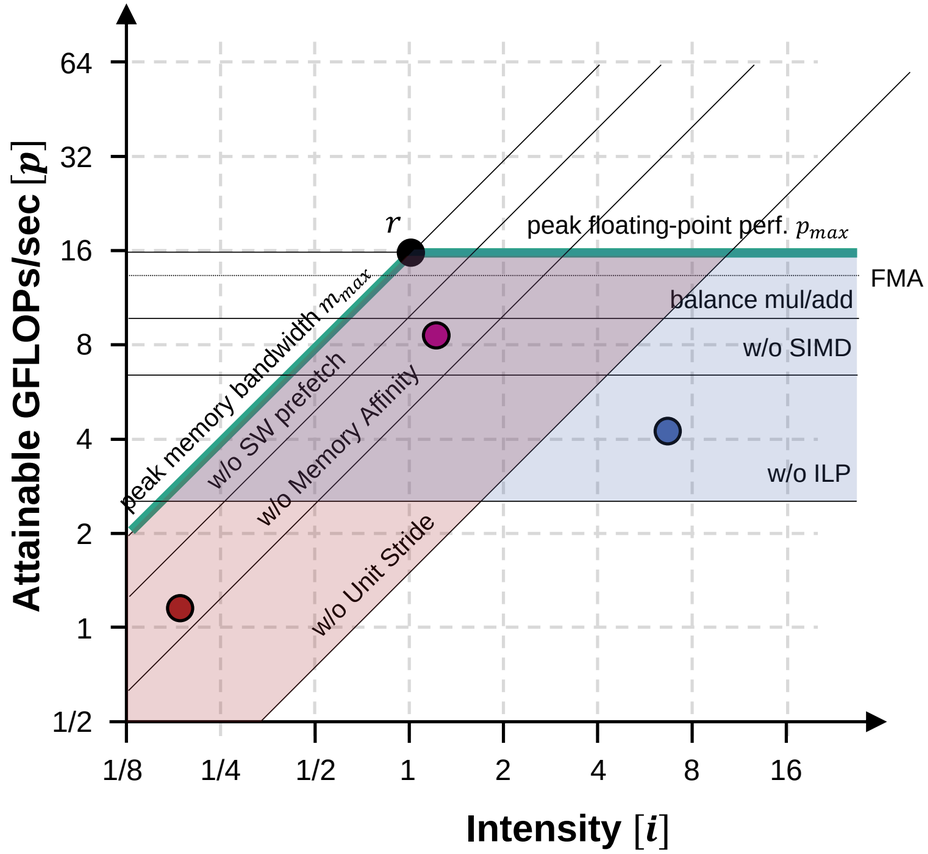
1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16

The Roofline Model

Roofline Regions, Optimization Order

- Ceilings partition the graph into 3 regions
  - Blue: focus on compute optimizations
  - Red: focus on memory optimizations
  - Purple: try both
- Height of gaps: potential reward
- Order of optimizations:
  - Bottom: inherent in algorithm
  - Middle: what an optimizing compiler or programmer can provide
  - Top: what might never be exploited for this kernel
  - Depends on the actual kernel!

There is no single roofline model!



The Roofline Model

Cache Performance

- The Roofline model requires an estimate of total data movement
- For some kernels, operational intensity increases with problem size (e.g., FFT)
- Intensity is tied to cache performance (data movement) on cache-based architectures:

$$\bar{t}_{ma} = t_{hit} + r_{miss} * t_{miss}$$

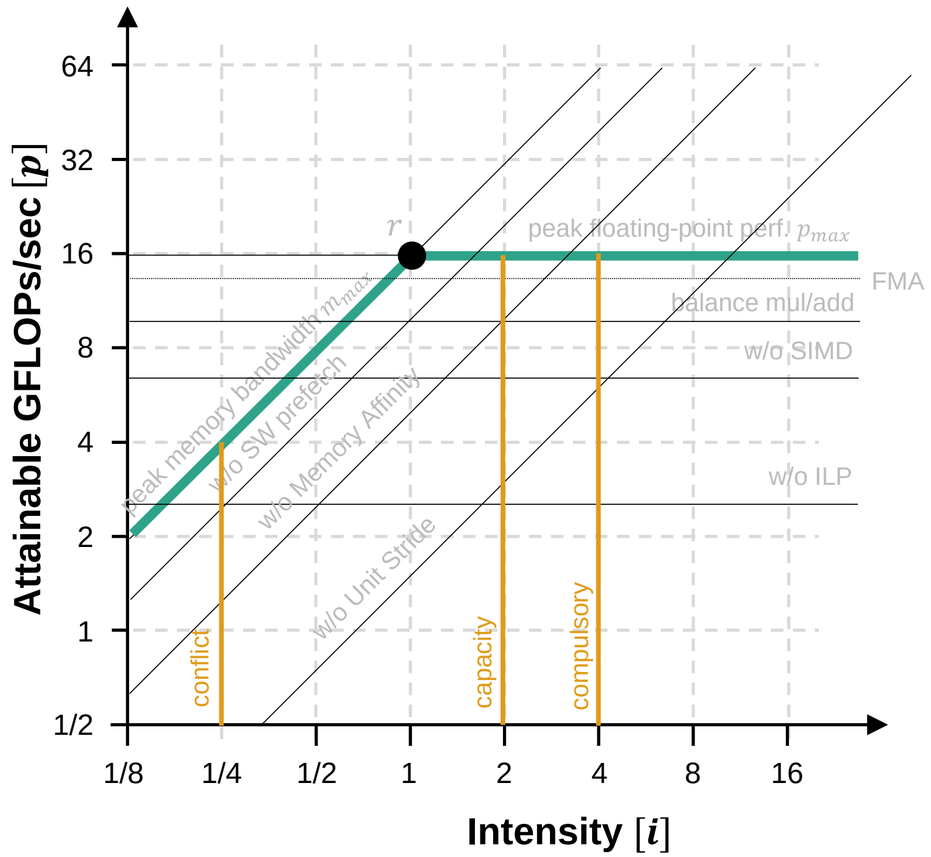
  - $\bar{t}_{ma}$: Average memory-access time
  - t_{hit}: Hit time
  - r_{miss}: Miss rate
  - t_{miss}: Miss penalty

CCC

- Reasons for data movement (Cache I/O) can be distinguished into:
  - Compulsory miss: item has never been in the cache (unavoidable!)
  - Capacity miss: item has been in the cache but it was forced out (capacity limit)
  - Conflict miss: non-compulsory, non-capacity; byproduct of the cache mapping algorithm
- Capacity and conflict misses can increase data movement and decrease intensity
- Additionally, superfluous write-allocations can result in doubled data movement:

```
x[i] = 0.0;
```

The Roofline Model



CCC and Roofline

- Compulsory, Capacity, Conflict misses
- Can be expressed in the model (given some kernel $\mathcal{A}$ and machine $\mathcal{M}$!)
- Conflict misses may destroy performance

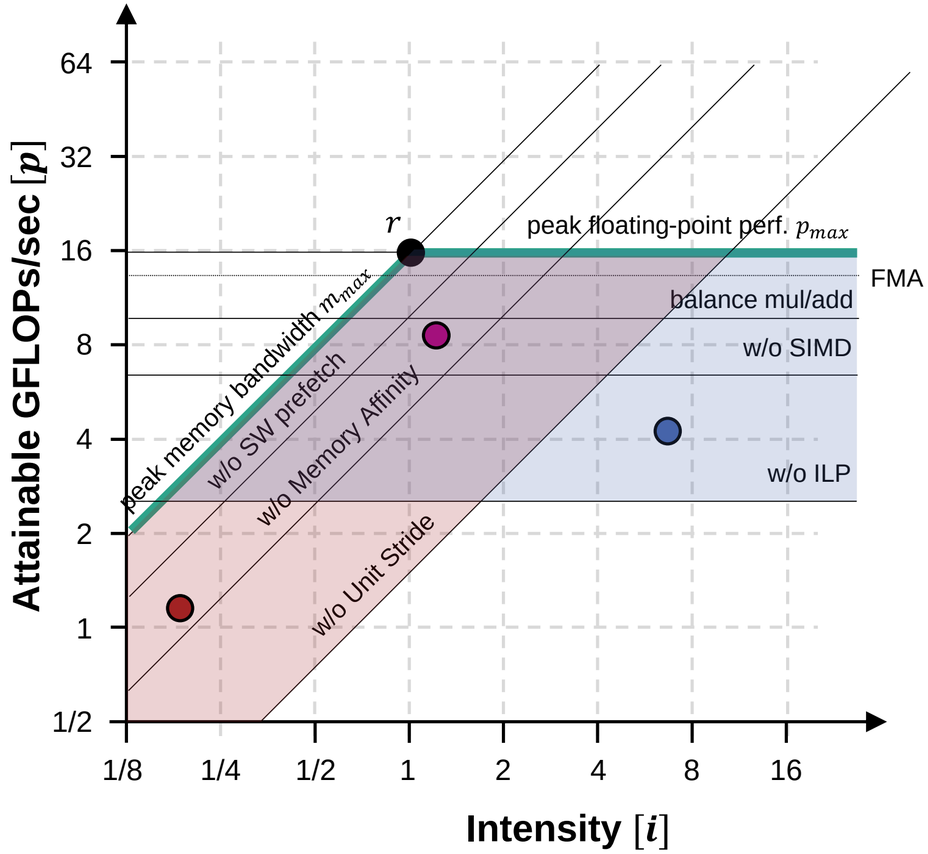
(Compiler) Optimizations

- Merging Arrays
- Loop Interchange
- Loop Fusion
- Blocking
- …

The Roofline Model

Model Adaption

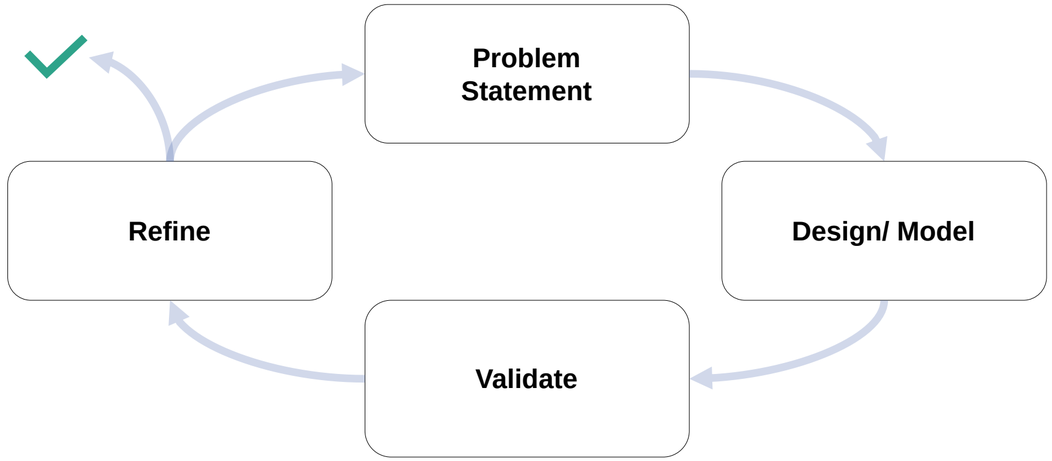
- Depending on your kernel and architecture the possible optimization can be limited
  - E.g., no balance of mul/add possible if not used inside your kernel
  - E.g., no SIMD if there are no such instructions
- Also, the order of optimization steps can vary!



The Roofline Model

Accept

- Know when to stop
- ECM (Execution-Cache-Memory) Model
  - Can be seen as generalized form of the roofline model
  - Can be used to describe single-core performance on a multicore chip
  - Same fundamental idea (data transfer or CPU time limit performance)
  - Starting Point: https://arxiv.org/abs/1410.5010
- For the sake of this lecture, we're be happy with the roofline model for now 😊

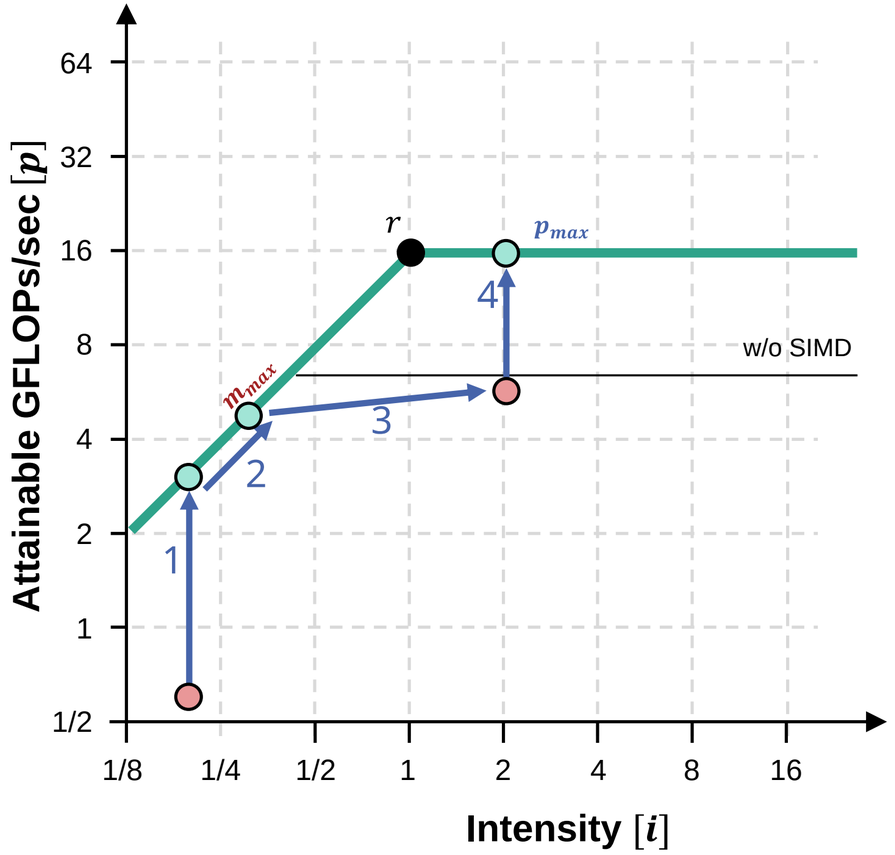


The Roofline Model

Working with the model *(Example)*

1. Hit bandwidth bottleneck (by good serial code)
2. Increase intensity (better use of bandwidth bottleneck)
3. Increase intensity (we become memory bound)
4. Hit core bottleneck

Overall approach: Bound and bottleneck



The Roofline Model

Modeling Toolkits

- Kerncraft (Roofline, ECM)
  https://github.com/RRZE-HPC/kerncraft,
  https://arxiv.org/pdf/1702.04653.pdf
- Intel® Advisor (Roofline, GPU; *2021)
  https://www.intel.com/content/www/us/en/developer/tools/oneapi/advisor.html
- LLVM Machine Code Analyzer
  https://llvm.org/docs/CommandGuide/llvm-mca.html
- OSACA
  https://github.com/RRZE-HPC/OSACA?tab=readme-ov-file
- IACA († 2019)
  https://www.intel.com/content/www/us/en/developer/articles/tool/architecture-code-analyzer.html

The Roofline Model

Discussion

- Bound and bottleneck approach
- Intuitive graph for kernel analysis and optimization
- Versatile model
  - Could also use integer, pixels, ... as performance criterion
  - Could also use other bandwidths (L2, PCIe, Network,...)
- Roofline(s) must be calculated for each set of performance metrics and machine...
- ...but only once
- Some features of modern processors are taken into account rather implicitly (cache hierarchy, hardware prefetching)

The Roofline Model

Discussion

- Ceilings are not kernel-specific
- Assumption:
  - Steady state *(start-up/ wind-down effects are neglected)*
  - Overlap *(data transfers and arithmetic code execution overlap perfectly. As a consequence, the "slowest bottleneck" wins, no matter by which margin, and there is no interaction between bottlenecks)*
  - No-latency *(data paths work at their highest achievable bandwidth, latency effects are ignored)*
  - Saturation *(it must be possible, on the hardware at hand, to actually saturate the memory bandwidth)*
- Optimistic model, overestimates attainable performance
- Considers only a single data transfer bottleneck, in-cache performance is not correctly predicted

Takeaways

Takeaways

- There is not the one performance criterion
  - We need to consider CPU time and memory bandwidth
- There is not *the one* performance model
  - Beware of overfitting, choose the easiest model that "does the job" - i.e., answers your questions
  - The roofline model is a versatile, intuitive and machine-specific model to do so
  - All Models are wrong - but some are useful!

Thank You!