Combining “real” and “artificial” intelligence for performance engineering: a toolbox approach

Ana-Lucia Varbanescu

with some inputs from Stephen Swatman

UNIVERSITY OF AMSTERDAM

UNIVERSITY OF TWENTE.

Some terminology

- "RI" – real intelligence
  - Relying on knowledge and analytical skills to do things "by hand"
  - Assumed white-box
- "AI" – artificial intelligence
  - Relying on data and statistical methods to automatically learn
  - Assumed black-box
- Performance Engineering
  - A **systematic**, **quantitative** approach to the design and development of software to **meet performance requirements** [1]
- Toolbox
  - A collection of tools to be **selected** and **used** for various tasks

[1] Connie U. Smith - https://www.spe-ed.com/classic-site/speis.htm

This talk is based on …

- **PE – Performance Engineering**
  - **MSc course on performance analysis, modelling, improvement** [2]
- CPP/DPP – Concurrent/Distributed and Parallel Processing
  - BSc course on parallel computing
- PMMS – Programming multi- and many-core systems
  - MSc course on shared memory and GPGPU programming
- A24 – A Programmer's Guide to HPC
  - Graduate course on HPC,from systems to application design to performance.

[2] Varbanescu et. al – "Performance Engineering for Graduate Students: a View from Amsterdam

Course structure

Lectures

Theoretical and empirical concepts.

Combine fundamental methods and tools with modern, state-of-the art approaches.

Teaches students how to expand their knowledge.

Labs

Link theoretical aspects with processing and tools that facilitate their application in practice

Small-scale assignments, limited coding

Focus on performance analysis, modeling, and prediction

Project

Experience performance engineering for a real case-study application.

Understand the limitations and challenges of the provided methods and tools

Course structure

Lectures

Theoretical and empirical

Define the toolbox.

the art approaches.

Teaches students how to expand their knowledge.

Labs

Link theoretical aspects with

Test the tools

Small-scale assignments, limited coding

Focus on performance analysis, modeling, and prediction

Project

Experience performance

Use the tools

challenges of the provided methods and tools

Assessment

Exam: 20-25%

Test theoretical knowledge.

Test the understand of methodological aspects of performance engineering.

Augmented with in-class quizzes to stimulate students' interest in these aspects during lectures.

Assignments: 25-30%

Grade the ability of solving specific aspects of performance engineering.

Reports worth more than coding/tools.

Showcase the practical challenges of performance engineering.

Project: 50%

Assess the ability of students to plan, execute, and document a complete performance engineering project.

Also include a communication aspect, with intermediate and final results.

Assessment

Exam: 20-25%

Test theoretical knowledge.

Understand the methods

Augmented with in-class quizzes to stimulate students' interest in these aspects during lectures.

Assignments: 25-30%

Grade the ability of solving

Demonstrate skills with the tools

coding/tools.

Showcase the practical challenges of performance engineering.

Project: 50%

Assess the ability of students to

Use the entire toolbox

Also include a communication aspect, with intermediate and final results.

Agenda

- ~~Terminology and setup~~
- Building the toolbox
- Practice makes perfect
- Did it work?
- Take home message
- Dreams for the future

Part 1: Building the toolbox

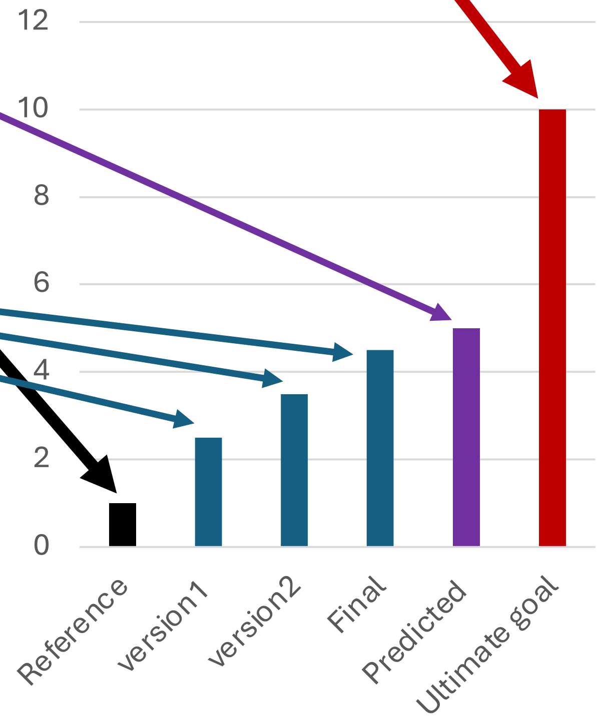
Systematic Performance Engineering

1. Collect and analyse (user) performance requirements.
2. Understand current performance.
3. Assess feasibility of the requirements.
4. Assess suitable approaches to meet the requirements (including algorithm and/or system (co-)design).
5. Apply tuning and optimization.
6. Assess progress and iterate back to steps 3–5.
7. Analyse and document the process and result

Main goal: a student finishing the class must know how to execute this process on a given system for a given application.

Systematic Performance Engineering

1. Collect and analyse (user) performance requirements.
2. Understand current performance.
3. Assess feasibility of the requirements.
4. Assess suitable approaches to meet the requirements (including algorithm and/or system (co-)design).
5. Apply tuning and optimization.
6. Assess progress and iterate back to steps 3–5.
7. Analyse and document the process and result



Systematic Performance Engineering

Project

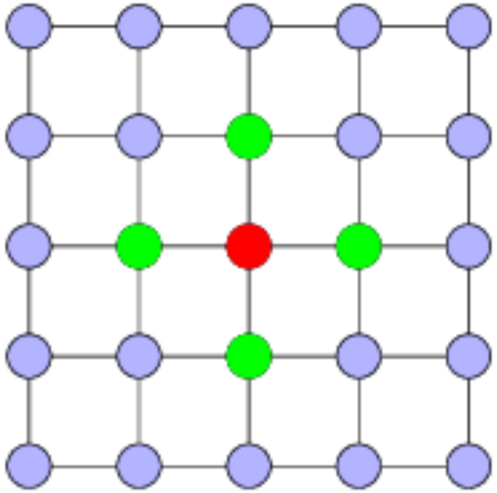
1. Collect and analyse (user) performance requirements.
2. Understand current performance.

Lectures

3. Assess feasibility of the requirements.
4. Assess suitable approaches to meet the requirements (including algorithm and/or system (co-)design).
5. Apply tuning and optimization.
6. Assess progress and iterate back to steps 3–5.
7. Analyse and document the process and result

Running example

Calculate the heat dissipation (2D stencil operation, iterative) in a metal cylinder.



1. Performance requirements

- **Types** of performance requirements
  - Real-time performance
  - Best possible performance
  - *N* times faster than reference implementation
  - *X* % or more hardware utilization
- **Metrics** for performance and beyond
  - Generic: speed-up, scalability, efficiency
  - Application-specific **metrics**
    - How to define, measure, interpret, explain

Calculate heat dissipation for a 10K x 10K cylinder in 1ms.
15 ops per point x $(10K)^2$ = 1.5GFLOP => Throughput = 1.5 / 0.001 = 1500 GFLOPS

2. Current performance

- Experimental setup
  - Collect/infer relevant use-**scenarios**
    - Input data included !
  - Per **scenario**:
    - Profile the code => **identify hot-spots**
    - Measure performance in detail **=> identify bottlenecks**
- Benchmarking
  - Methods and tools

Current speed for a 10K x 10K cylinder is 1s => We need 1000x improvement!

3. Can it be done?

- Feasibility analysis based on modeling.
  - Analytical modeling
  - Statistical/ML-based learning
  - Simulation
  - Benchmarking

Maybe 1500 GFLOPs is too much?

CPU peak performance = 100 GFLOPs << 1500 GFLOPs => CPU not feasible!
GPU peak performance = 2000 GFLOPs > 1500 GFLOPs => GPU feasible ... with 75% utilization! => maybe ...

4. How can it be done?

- Select the methods and tools for tuning.
  - Identify feasible actions
    - Better hardware/OS/…
    - Tuning – parameters, compiler-options, etc.
    - Implementation – better constructs, more efficient data structures
    - Restructuring / refactoring – better algorithms, methods, parallelization, …
  - Rank & select options in terms of gain **using performance models**

Key challenge: accurate models!!

Code optimization: Apply SIMD => 2x , Improve caching => 1.5x, …
Different algorithms: Handle boundary conditions, aggressive recalculation, …
Better hardware: Use a GPU! => 20x

5+6. Tuning & Evaluation

- Implement the selected tuning methods
  - Apply one action at a time
  - Re-evaluate after each step
    - Performance => “update” models
    - Tuning steps => “update” plan
  - Are you there yet? If not: continue.

Mostly implementation, benchmarking, and model refinement.
Reorder optimizations depending on current results.

7. Analyze & document the result

- Document design options & choices
- Document models and benchmarks
- Reflect on the results
  - Cost, effort, sustainability
- Document future steps, and their requirements
  - Cost, effort

Selected algorithm A because … => see model!
Used a GPU because the CPU was not feasible => see model!
Further implementation to make … => see model!

What's in the toolbox: methods

- Types of performance (1)
- Metrics (1)
- Design an experimental setup (2,6)
- Run measurement experiments (2,3)
- Benchmarking (2)
- Microbenchmarking (2,3)

Modeling HW & SW (3,4)
  - Analytical
  - Statistical
  - Simulation
- Performance improvement methods (5)
- Performance assessment (6)
- Documentation (7)

What's in the toolbox: systems and languages

- Systems
  - CPUs, GPUs, Distributed (super)computers
- Benchmarking suites
  - STREAM, uOps, OSACA, LLVM-MCA/iaca
- (micro)benchmarking tools
  - LIKWID, PAPI, Perf
- Programming languages/models
  - C/C++, CUDA, OpenCL/SyCL, MPI
  - Others allowed, but not actively supported

What's in the toolbox: "actual" tools

- Roofline model and tools
  - Vtune, nsight, the roofline toolkit, …
- Simulators
  - GEM5, AccelSim,
- Measurement tools
  - LIKWID, PAPI, Perf, nvisia-smi, nsight, …
- Optimization tools
  - Autotuning tools
  - Libraries!
  - Polyhedral model and experimental compilers
    - Various online resources, the Polly benchmark

What's in the toolbox – modelling

- Analytical modelling
  - Roofline model & ECM model
  - Bottleneck analysis
  - Queuing theory
- ML-based modelling
  - Data collection and management
  - Examples of basic statistics/ML
  - Complex DNN
- Simulators & benchmarks

Topics

Topic	Stages 1	2	3	4	5	6	7	Learning Obj. 1	2	3	4	5	6	7	8
Basics of performance	✓							✓							
Code tuning and optimization				✓									✓	✓	
Roofline model and extensions		✓						✓	✓	✓				✓	
Analytical modeling		✓	✓					✓	✓	✓	✓				
(Micro)benchmarking	✓		✓								✓		✓	✓	✓
Data-driven and stat. modeling	✓		✓					✓	✓	✓			✓	✓	
Simulation and simulators		✓		✓		✓					✓		✓	✓	✓
Perf. counters and patterns		✓			✓	✓							✓	✓	✓
Scale-out to distributed systems	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓	✓	✓
Queuing theory			✓	✓					✓	✓					
Polyhedral model					✓								✓	✓	✓

Learning objectives

1. **Quantify** (using the appropriate tools and methods) performance using the relevant metrics;
2. **Use** and **compare modelling methods**, and **assess** their usefulness
3. **Classify** and **use performance prediction methods**, and assess their usefulness
4. **Design** an empirical performance analysis process for an application, **analyse results**, and **recommend performance improvement** solutions;
5. **Design** and **use** a suitable **model** for **accurate performance prediction** for a given application;
6. **Apply** and **assess** different **optimization techniques** to parallel and distributed codes;
7. Design, develop, apply, and assess a **complete performance engineering process** for a given application;
8. Use different **performance engineering tools** (e.g., profilers, microbenchmarks and benchmarks, performance counters libraries, etc.).

Part 2: Practice makes perfect

Quizzes (examples later?)

- At least one per class
  - Focus on training curiosity and intuition
- Graded for bonuses
  - And acting as a bridge between theory/lectures
- Linked to exam questions
  - Direct application

Assignments

1. The Roofline Model for a simple kernel
   - Applied for sequential code, parallel code, optimized/accelerated code
   - Demonstrate the model can assess the differences
2. Analytical Modeling and Microbenchmarking
   - Design, calibrate, and evaluate analytical models
   - Design/reuse microbenchmarks for calibration
3. Statistical Modeling
   - Choose and use machine learning models
   - Assess their cost and accuracy
4. Performance Counters and Performance Patterns
   - Learn how to collect and understand detailed performance data
   - Use performance patterns to diagnose and solve performance problems

Assignments

1. The Roofline Model for a simple kernel
   - Applied for sequential code, parallel code, optimized/accelerated code
   - Demonstrate the model can assess the differences
2. Analytical Modeling and Microbenchmarking

This is all in “sanitized” conditions.

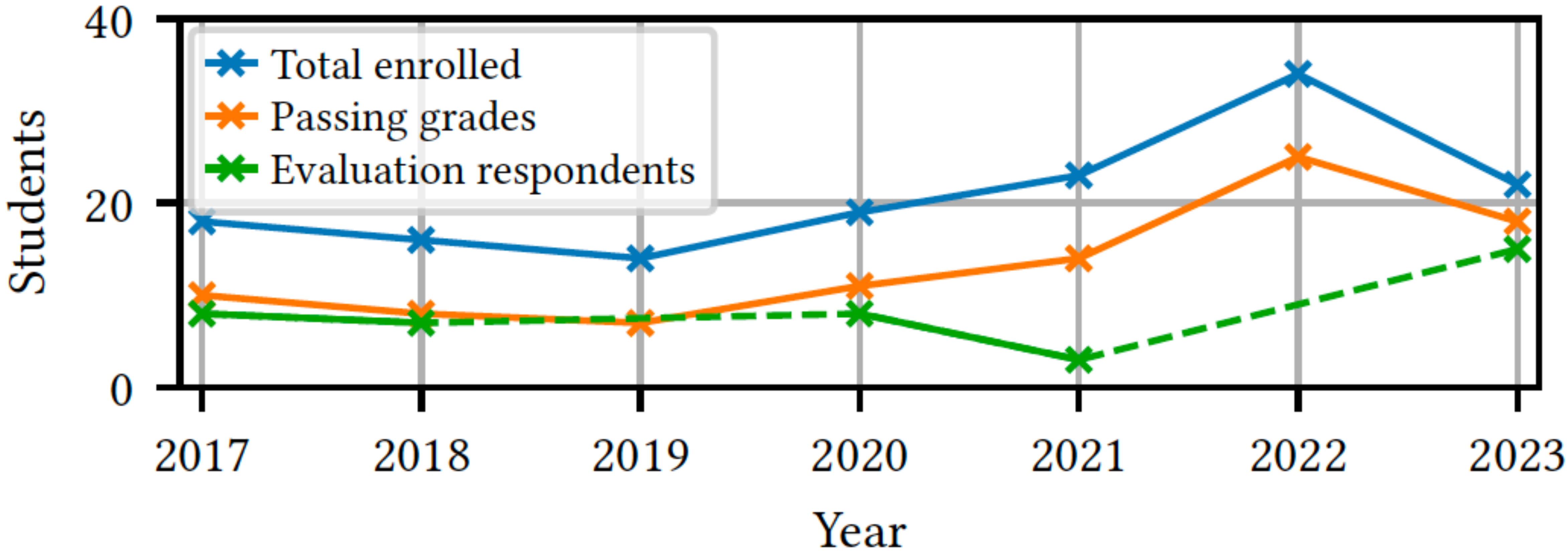
   - Assess their cost and accuracy
4. Performance Counters and Performance Patterns
   - Learn how to collect and understand detailed performance data
   - Use performance patterns to diagnose and solve performance problems

Project

- Let’s redo that in real-world conditions.
- Examples
  - Optimize ADAM
  - Optimize graph processing
  - Optimize EveOnline and LunarLockout
  - Optimize Wordle
  - Optimize stencil operations
- Documentation and reflection – 60% of the grade
  - This is a true discriminator!

Part 3: Did it work?

Results


Total enrolled
Passing grades
Evaluation respondents
40
20
0
Students
2017
2018
2019
2020
2021
2022
2023
Year

Feedback

- High grades across all categories.
- High praise for the course structure and interconnection between components.
- High workload students spend 20–50% more time than officially allocated.

Statement	Firmly Disagree	Disagree	Neutral	Agree	Firmly Agree	*M*
"The course …"						
Taught me a lot	0	0	1	17	18	4.5
Was clearly structured	0	2	3	19	13	4.2
Was intellectually challenging	0	0	2	9	25	4.6
"I acquired, learned, or developed …"						
Factual knowledge	0	0	1	13	13	4.4
Fundamental principles	0	1	2	16	11	4.2
Current scientific theories	0	3	5	13	9	3.9
To apply subject matter	0	0	0	7	22	4.8
Professional skills	0	0	3	13	15	4.4
Technical skills	0	0	6	14	9	4.1
"… helped me understand the subject"						
Assignment 1	0	1	1	12	16	4.4
Assignment 2	0	0	1	11	16	4.5
Assignment 3	1	1	1	17	10	4.1
Assignment 4	0	1	1	12	13	4.4

Note: In 2017-2018, assignments were evaluated with a single score; these scores have been duplicated across the four separate assignments in this table.

Lessons learned

1. Performance Engineering is appealing when treated like a puzzle.
    - We appeal to students' curiosity to understand why applications behave weirdly on different systems.
2. Provide both methods and tools for each part of the course.
    - Students appreciate the theory much better when they can link it to concrete examples.
3. Do not underestimate empirical analysis efforts, especially when experimental design is missing, and/or automation is not properly defined.
    - We spend time and provide many examples on how this should be done correctly and efficiently.
4. Projects stimulate creativity, and students should be allowed exploration time and space.
    - We provide no endline for our projects, and allow students to try different things
5. Stimulate critical thinking by reporting on both positive and negative results.
    - We grade the process and the actual insights, and not ultimate speed-up or high-accuracy; understanding why and how methods and tools work and fail is fundamental to such a course.
6. This is an intensive course for both teachers and students.
    - We offer a course that students can rely on and apply for their real-life performance engineering projects

Practical aspects - positive

- Applications that work
  - matrix multiplication
  - histogram
  - SpMV
  - Graph processing

- Systems that work
  - CPUs, GPUs, heterogeneous, distributed

- Tools that *exist*
  - Many many many, and we keep updating them

Practical aspects – challenging*

- Lack of background
  - Math knowledge
  - Statistics knowledge
  - Programming skills
  - Data management
- Lack of curiosity / goals / … ?
  - Difficult to select projects
- Lack of big thinking

Performance Education in a Data-Driven World

- Performance Engineering is a big data problem
  - Many systems
  - Many input datasets
  - Many metrics and data to collect
- Performance Engineering with ML is exciting …
  - But still requires to explain why
  - Usually a starting point
  - Leads to interesting patterns
- Collecting data and traces leads to very creative performance analysis & optimization methods

Motivating students

- Provide a mix of theory + practice + real-world application
- Treat performance like a puzzle
- Do lots of hands-on things: quizzes, project, exercises, labs
- Provide a lot of examples and war stories
- Combine hardware (systems) with software (programming) and modelling (not-a-lot-of-math)
- Target metrics - ”fast”, “efficient”, “energy-saving” …
- Target practical skills and cultivate appetite for theory

Take home message

Take ~~home~~ message
to-the-office

- Pro's:
  - The toolbox approach exposes students to many methods and their implementation into tools
  - They practice on "controlled" applications
  - They "apply" on a "real-world" project
  - They choose how and when to use them
  - They learn what is needed today for system-level performance modeling
- Con's
  - "Demand-driven" - principles follow practice
  - Not a lot of math and not a lot of theory

Take ~~home~~ message
to-the-office

- We were lucky to embed the course within a series of courses
  - Making puzzles was easy
- We had an excellent set of TAs
- We spent a lot of time in quizzes and report reading and providing feedback
- We embraced machine learning from the very beginning
  - Students wanting to use it, can …
    - And find out it is non-trivial.

What next ?

- A general curriculum for performance engineering ?

- A “compendium” of theory and practice ?
    - Keep up-to-date with models and tools

- Expand to sustainability metrics ?

- Expand to ...
    - Computing continuum e
    - Embedded/cyber-physical systems
    - Model-based (co-)design