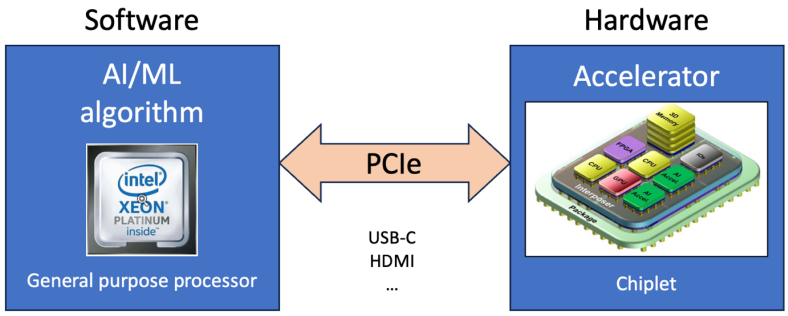
Week 2

Codefest: Bootstrapping the main project!

ECE 410/510
Spring 2025

Main project goals

- **Design, test, and benchmark a co-processor chiplet that accelerates parts of some AI/ML code/algorithm of your choice** (see list at the end). Start with a blank slate for your design.
- [ALTERNATIVE] Design a stand-alone chiplet. Your chiplet could, for example, include an ARM core.
- You will be deciding what design constraints to impose (e.g., power budget) and what metrics to use (e.g., throughput).
- Apply HW/SW co-design principles to decide what part of your algorithm is executed in HW.
- The chiplet should be described in a HW-description language (e.g., Verilog). The design should be synthesizable in order to obtain your relevant metrics.
- The deeper you can go into the HW design, the better. The ultimate goal is to go all the way down to an ASIC description. E.g., by using OpenLane's workflow to convert HDL into GDS: https://www.zerotoasiccourse.com/terminology/openlane.



Project dos and dont's

Dos

- Pick something exciting, at the forefront of technology
- Pick something that challenges you,
- Failure should be an option.
- Something new that you want to learn
- Start with a blank slate, but it's OK to draw inspiration from existing architectures.

Dont's

- No relying on current accelerators, e.g., GPUs
- No traditional CPUs (except for executing non-accelerated code).
- No cache algorithms, no branch predictors, etc.!
- No teamwork allowed. You must have your own project. If several want to work on the same AI/ML problem, you'd be competing with each other (in a friendly way!)

During codefests

- Practice "vibe coding" if you don't have the background to code on your own.
- Slack will be our main collaborative platform. Keep an eye on it.
- Post questions, solutions, insights in the `#codefests` channel.
- Present what you've got, whether it works or not.

Challenge #9

Learning goals:

- Understand the problem.
- Answer the Heilmeier questions.
- Analyze the algorithm, identify bottlenecks, generate data-flow graphs, profile the code, generate call graphs.
- Draw a high-level block diagram and/or flow chart of your algorithm (unless you can find one online).
- Get a first idea of what parts of your code would benefit from a HW acceleration.

Suggested tasks:

1. Answer the Heilmeier questions for your project idea:
   https://www.darpa.mil/about/heilmeier-catechism
2. If you have good answers to these questions, you likely have a solid project.
3. For question 2, use Google Scholar and ResearchRabbit.
4. Post your answers in the following Google Doc:
   https://docs.google.com/document/d/1fDuM5bUIuZBPGqjvlTHfYhtRlWyl6sWodc8ZgmDZwF4
5. Find code for your algorithm, write your own, or use "vibe coding." The code would ideally be in Python because of the availability of great tools and libraries. But any other high-level language that allows for profiling, data-flow analysis, etc., is fine too.
6. Establish an initial software benchmark that will serve as a baseline for comparing your accelerator HW later on.
7. Analyze the algorithm, identify bottlenecks, generate data-flow graphs, profile the code, generate call graphs. **For all of these tools, LLMs will be happy to provide sample codes**. You may want to use some of the following tools for that:
   - Python profiling tools like `cProfile`, `line_profiler`, or `py-spy` allow you to identify bottlenecks and optimize code for specific hardware.
   - `memory_profiler` can be used analyze memory usage patterns.
   - `Doxygen` can generate call graphs showing what functions/methods a particular function/method calls, which is a great way to understand the code flow. For your algorithm, this will visualize the relationships between different functions and methods. More at https://dzone.com/articles/understanding-code-call-graphs
   - `StaticFG` (Static Control Flow Graph) is a package that generates control flow graphs for Python 3 programs. While it focuses on control flow rather than data flow, these CFGs can be visualized with `Graphviz` and used as a foundation for static analysis, including data flow analysis.
   - `CodeQL` provides capabilities for data flow analysis in Python programs. It allows you to track the flow of data through a program to points where the data is used, with support for both local data flow (within a single method) and global data flow (throughout the entire program)
   - `PyFlowGraph` is a tool developed by IBM that can generate dataflow graphs from Python code, though it primarily performs dynamic analysis rather than static analysis. It outputs data in graph.ml format.
   - `PYCFG` is a tool that can be used to draw control flow graphs for Python code. While primarily for control flow, it can be extended to incorporate data flow information.
   - In Python, the Abstract Syntax Tree (`AST`) module is a powerful tool for analyzing and manipulating Python code programmatically. For creating custom dataflow analysis tools in Python, you can use:
     - Python's built-in ast module for parsing code into an abstract syntax tree
     - NetworkX for building and visualizing graph structures
     - Static analysis libraries like StaticFG for control flow graphs
8. Draw a high-level block diagram and/or flow chart of your algorithm.

   - You can either do that based on info about the algorithm you find online and/or
   - Based on the analysis and data of the tools above.
9. Get a first idea of what parts of your code would benefit from a HW acceleration.
   - `PyRTL`, `MyHDL`, or `pynq can be used to` design or interface with FPGAs or other hardware from Python.
   - `libusb, pyserial,` or `RPi.GPIO` can be used to interact with devices drivers through libraries.
   - `cocotb or` MyHDL can be used for Python-based hardware verification and co-simulation.

Typical AI/ML applications

- **Computer Vision**
  - Image classification and object detection
  - Facial recognition (with privacy considerations)
  - Medical image analysis
  - Quality control in manufacturing
  - Autonomous vehicles and robotics
- **Natural Language Processing**
  - Machine translation
  - Sentiment analysis
  - Text summarization
  - Chatbots and virtual assistants
  - Information extraction from documents
- **Recommendation Systems**
  - E-commerce product recommendations
  - Content recommendations (streaming services, news)
  - Advertisement targeting
  - Social media feed curation
- **Predictive Analytics**
  - Financial forecasting and risk assessment
  - Predictive maintenance for equipment
  - Supply chain optimization
  - Healthcare outcome prediction
  - Customer churn prediction
- **Anomaly Detection**
  - Fraud detection in finance
  - Network security monitoring
  - Manufacturing defect detection
  - Health monitoring systems
- **Time Series Analysis**
  - Stock market prediction
  - Energy load forecasting
  - Weather forecasting
  - Sales forecasting
- **Reinforcement Learning**
  - Game playing agents
  - Robotics control
  - Resource management optimization
  - Autonomous systems
- **Speech Recognition and Synthesis**
  - Voice assistants
  - Transcription services
  - Accessibility tools

  - Voice-based authentication
- **Generative AI**
  - Text generation (creative writing, code, content)
  - Image, video, and audio generation
  - Design assistance (architecture, fashion, graphics)
  - Synthetic data generation