Course Project

The final component of this course is a project; there is no final exam. The topic would ideally be selected/proposed by week 5.

Quick Facts

- Due Date: June 12th, Midnight
- Subject: If it has a relationship to ML & hardware, its acceptable!
- Work in teams: Yes, 1-4 Students (preferrably 2-3)
- Deliverables: Project Report + Source Code tarball (will not distribute)

Project Content

The goal of the project is to give you an opportunity to get hands-on experience with architecture experimentation and maybe even research.

Propose a research idea and evaluate it using any means you like. You are free to combine with ongoing research from your own studies, or with another course, provided the scope of the project implementation submitted for this course is sufficient. Please see me if this is unclear.

Report guidelines:

The final project report should document the work that you have performed and the findings. You should strive to make the presentation of your report the same quality as the papers you have read during the quarter, even if it ends up being much shorter. The paper should stand alone as well – the concepts should be understandable by your classmates without having to read additional papers (ie. including relationship to existing work). Please format the paper nicely, and organize the paper with good structure. *Finally, please include a statement of work which describes how each student contributed to the project.*

As you may have noticed, papers typically follow one of a few canonical structures. The one below is a reasonable approach.

- Abstract: Summarize the main contribution(s), and list any key findings.
- Introduction: Describe the problem your work addresses, why it is important, and overview the solution (if you are proposing one).
- Related Work: Overview the relationship of your work to prior work, with citations. It's of course okay if the work is not 100% novel.
- Methods: Detail the proposed design or methods (this may span one or more sections). This includes the design of any algorithms, hardware structures, interface strategies, codesign techniques, etc. Be sure to use drawings/figures/code-examples that can help explain the ideas.
- Methodology: Describe your approach for how you evaluated the work, and explain why it is fair or valid.
- Evaluation: Provide and analyze any quantitative results.
- Conclusions: Summarize the findings and main contributions, as well as any ideas for future work.
- Statement of Work: For each person in the group, describe the tasks that they performed.
- References: List all references cited.

Project Ideas

A few broad project ideas are below. These are meant as examples; projects that don't fit into any of these categories are acceptable.

Accelerator for Machine Learning Kernel:

This project is straightforward, but very broad: design/evaluate a hardware accelerator for an ML algorithm of interest. There are different facets of this to think about to create an interesting problem:

- Algorithm to target: Any ML algorithm of interest, or set of algorithms, could be chosen. Simple FC/CNN layers are acceptable, but to make things more interesting compared to what we've focused on in class, you could also consider specializing to different networks like LSTMs, or transformer/BERT. Going in a completely different direction are other classic ML algorithms (top 5 facebook algos) like GBDT, SVM, LR.
- Algorithm optimizations: Could be interesting to try different optimizations for accelerators (eg. winograd?), or combine different sources of efficiency in one accelerator (how to combine exploiting sparsity with bit-level optimizations?).
- Type of accelerator: The accelerator can either be fixed-function, take algorithm-specific commands (eg. NN layer parameters), or be more general to a whole domain. Each would offer different tradeoffs and can be interesting.

The evaluation criteria could include how large, fast, and power hungry the design is as compared to some relevant baseline. You might choose a baseline to be an existing CPU/GPU code. For evaluation, you could use a custom simulator, or an analytical model. Feel free to use an FPGA if you already have some experience.

Optimization Framework

Timeloop Timeloop/Accelergy and MAESTRO are tools that allow the modeling of performance and energy based on the dataflow. These modeling tools can be enhanced in a number of ways:

- Add new algorithm features to the model. For example, add the ability to do model inter-layer parallelism.
- Add some new hardware feature: perhaps support for different kinds of sparsity that we discuss in class.
- These tools require enormous amounts of time to exhaustively search the space of transformations to find best one. Propose a technique (perhaps a heuristic technique) to search the space more quickly.
- Add the ability to automatically search over hardware features under certain area/power/performance constraints.

CPU Tensor Cores (or other CPU-core enhancements):

Implement a tensor core for a CPU. Tensor cores have been added to GPUs to help improve their performance on high-compute-density operations. This basically boils down to a few new instructions in the ISA for doing matrix multiply tiles. One could add these to a CPU instruction set, and evaluate with gem5.

More broadly, perhaps there are other challenges for CPUs that limit their performance on eg. CNNs. Another project could be to run some CNN layers on a CPU simulator, identify the bottlenecks, and show how to improve their microarchitecture (or add other instructions).

Novel GPU Parallelization:

Generally, kernels on GPU are parallelized one-at-a-time (ie. one layer at a time across all CUDA cores). Sometimes this is inefficient because there isn't enough work per kernel. What if two or more kernels were running at the same time on the GPU (each running its own element of a batch?)?
This could lead to more efficient execution for a variety of networks.

This could also be done for an accelerator as well!

One could also look into parallelizing other ML algorithms on GPUs like gradient-boosting tree training, or various algorithms for recommender systems like alternating least squares.

Machine learning models for microarchitecture policy optimization:

There are many microarchitectural policies within programmable architectures (CPUs/GPUs) that are 1. Complicated to design and analyze, 2. Brittle between hardware versions – requiring redesign, 3. Only optimal for particular workloads. A machine learning-based approach can potentially mitigate the above. Some example areas would be:

- Branch Prediction: This is a classic area in computer architecture that's seen great success with neural network based prediction. Would be interesting to explore different models here.
- Prefetching: Microarchitectural prefetchers examine an address stream at some point in the cache hierarchy (eg. from l2 to l3) and learn the patterns. They then issue requests in advance to hide memory latency. Perhaps more advanced ML techniques can be used here instead of the simple techniques used so far.
- GPU Warp Scheduling: One challenging aspect of achieving good hardware utilization on GPUs is to determine how to choose a Warp (this is the GPU word for a thread) to execute next – this is called warp scheduling. Could we maybe use reinforcement learning here?

Evaluation: For any computer-architecture study, you can always use a generic simulator with generic workloads. For example, you could use gem5

- SPECINT 2017, which I can provide instructions for downloading.

The other option is to use the infrastructure from prior microarchitecture competitions.

1. Championship branch prediction
2. Data Prefetching Championship
3. Championship Value Prediction
4. Memory Scheduling Championship

Universal Approximation Accelerators:

Interestingly, some ML algorithms are quite efficient to compute because of how simple/regular they are. What's bizarre though, is that if some error can be tolerated, they can be even sometimes be more efficient than an existing *exact* algorithm!

The NPU paper (which we'll discuss) developed an accelerator which is integrated with a CPU which computes approximate versions of general functions using neural networks. They showed that a fully-connected NN-approximated code, running on their accelerator, is actually faster than precise computations for certain workloads.

However, there is still an open question on what the right ML model for automated approximate computing should be. The project here could explore different types of ML models (and model hardware implementations?) which could trade-off accuracy for performance.