teuscher.:Lab
teuscher-lab.com
Emerging Computing Models and Technologies

Christof Teuscher
ECE 486/586
Spring 2016

Portland State
UNIVERSITY

ECE 486/586, Spring 2016

Final Project: A Simple Pipelined Neuromorphic Processor

Learning Goals

The main goal of this project is to expose you to the learning experience of designing a processor from the beginning to the end. You will explore different instruction set architectures for the task at hand, pick the most appropriate one, and then build a functional simulator of simple pipelined processor specialized to perform the operations for neural networks.

Additional learning goals:

- Explore design trade-offs.
- Learn how to build a functional simulator.
- Apply what we learned in class to a real-world problem.
- Propose a solution to a challenging problem you may not be familiar with.
- Expand your horizon, learn new things.
- Deal with ambiguity .
- Work on a team.
- Write a report.

Introduction and Motivation

Neuromorphic chips are chips that model functions similar to what the human brain performs. The following two articles provide an initial overview:

- http://semiengineering.com/neuromorphic-chip-biz-heats-up
- https://www.technologyreview.com/s/526506/neuromorphic-chips

There are many ongoing projects, both nationally and internationally, with the goal to replicate some of the brain's functionality at a large scale. Some projects are based on general-purpose processors (e.g., ARM processors for the SpiNNaker project, http://apt.cs.manchester.ac.uk/projects/SpiNNaker/project), while other projects chose to build highly specialized architectures based on novel types of devices (e.g., IBM's TrueNorth chip, http://www.research.ibm.com/articles/brain-chip.shtml, or Intel's spintronic-based approach, https://www.technologyreview.com/s/428235/intel-reveals-neuromorphic-chip-design).

For the purpose of this project, we will be working with a very simplified version of a neural network: feed-forward linear threshold neural networks. Note that most of today's neuromorphic architectures are non-linear and recurrent, but we do not need to go to that level of complexity to learn about the basic trade-offs a computer architect faces with neuromorphic architectures.

Your task is to design, model, simulate, test, and benchmark a simple pipelined neuromorphic processor for feed-forward linear threshold neural networks. The processor should be optimized for throughput, i.e., the more data you can feed through the neural network in a given time interval, the better.

To model, simulate, and test your processor design, you will write your own functional simulator in a high-level language of your choice (such as C, C++, Java, Python, etc.). The goal is not to build a simulator that is hardware-accurate for each gate or circuit involved. However, your simulator needs to be cycle-accurate, i.e., be able to capture accurately the total number of clock cycles a program takes to execute. If you prefer (in case you are not sufficiently familiar with a high-level language), you can use a hardware description language, such as VHDL or Verilog, and build a more realistic circuit-level simulation. Note that this is neither required nor will it get you any extra points.

All the typical parts of the architecture need to be simulated, such as the **instruction decoder**, **ALU**, **registers** (if applicable), **pipeline** (if applicable), data and instruction **memory**, etc. The functional simulator that you will build needs to capture the effect of running the simulated program on the simulated machine state (which includes the typical parts listed above).

Your simulator needs to be able to load a text file that contains a memory image as an input. The memory image represents the initial state of the simulated program. It needs to contain the program that simulates your neural network, the network weights, and the data to be fed into the network. That means you also have to simulate a basic memory (see more details below). The output of the simulator needs to be another memory image (dump) that shows output data of the neural network. The output can be appended to the input file or written into a separate file.

It is up to the team to define the file format that is appropriate for their implementation. The following is simply an example:

- The data in the memory image should be 32-bits wide. Each line in the text file represents one word (4 bytes) of memory shown in the hexadecimal (base-16) format. The addresses start at "0," from the first line in the image and then increase by "4" at every next line. The addresses do not need to be specified in the memory image and are implicitly obvious from the line number in the image file. For example, if you want to read the contents of memory address $(20)_{10}$, then those contents can be found on the sixth line in the memory image.

Assume that the PC and all the general purpose registers (if applicable) are initialized to "0." As you simulate each instruction in the program, you will need to keep track of the machine state and make necessary changes to the state based on the impact of the simulated instruction. For example, if an instruction writes a new value to register R6, then you will update the register R6 with the instruction result. Each instruction will also update the PC, which will in turn cause a new instruction to be fetched and simulated. You will continue the simulation until you encounter some sort of "HALT" instruction.

The Neural Network to be Simulated

Your task is to simulate a fixed feed-forward threshold neural network as depicted in Figure 1. The network has 4 input neurons, 4 hidden neurons, and 4 output neurons. Feed-forward means that the network does not have any cycles. In other words, it can be organized into layers that are only connected by "forward" connections as shown in Figure 1. There is no learning happening in this network. The weights are fixed and should be stored in the initial memory file.

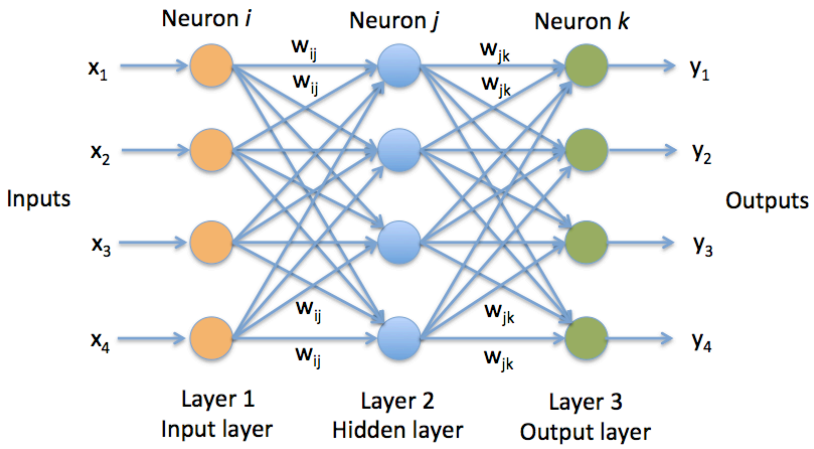

Figure 1. A 3-layer neural network with 4 input neurons, 4 hidden neurons, and 4 output neurons.

- **Input signals**. The input signals are binary inputs, i.e., 0 or 1.
- **Weights**. Each connection among the neurons has an associated weight. The value of each weight can be -1, 0, and 1. There are 32 weights in the network shown in Figure 1.
- **Input neurons**. These neurons do not perform any function. Their sole purpose is to distribute the inputs to the hidden layer through connections with an associated weight on them.
- **Hidden and output neurons**.
    - Each of the hidden and the output neurons first sums up the input signals multiplied by the weights, as shown in Equation 1. Think of this as an intermediate internal value.

$$z = \sum_{i=1}^{4} x_i w_{ij}$$

Equation 1. Each neuron sums up the inputs multiplied by the weights.

    - The resulting sum *z* is then fed through a threshold activation function *f(z)* that determines the actual output *y* of the neuron. This is described in Equation 2.

$$y = f(z) = \begin{cases} 1 \ if \ z \geq 0 \\ 0 \ if \ z < 0 \end{cases}$$

Equation 2. The threshold function determines the output *y* of a neuron.

- **How does data flow through this network?**
    - *Step 1*: Apply input values to inputs.
    - *Step 2*: Calculate the output of the hidden layer by considering the inputs and the weights between layer 1 and 2. I.e., use equation 1 and 2 for each neuron in the hidden layer.
    - *Step 3*: Calculate the outputs of the neural network by considering the outputs of the hidden layer and the weights between layer 2 and 3. I.e., use equation 1 and 2 for each neuron in the output layer.

We will spend some class time on the functioning of neural networks.

Data and Instruction Memories

We assume that data and instructions can be read from/written to your simulated memory/memories in **one** clock cycle.

Pipeline

The processor needs to be pipelined. The team decides how many pipeline stages are appropriate, what structure the pipeline has, how hazards are dealt with (if at all), etc. Remember: the goal is to achieve a high throughput for simulating the type of neural network described above.

Multicore and Superscalar Implementations

No multicore and superscalar implementations are allowed. Please stick to a simple single-issue pipeline so that the results of the teams can be compared. However, you are allowed and encouraged to make as many optimizations as you want/can in your pipeline otherwise. E.g., out-of-order execution, speculation, prediction, etc. Be creative!

Instruction Set

It is the team's task to determine what types of assembly instructions (defined by opcodes, operands, addressing modes, etc) are appropriate and needed to simulate feed-forward linear threshold neural networks efficiently. I recommend that you draw inspiration from the MIPS instructions as a starting point. Try to minimize the number of instructions you need as well as the addressing nodes. Keep things as simple as possible (i.e., RISC approach: simple = fast = beautiful).

Benchmarking and Testing your Processor

To test and benchmark your processor, do the following steps:

1. Randomly generate 1,000 random binary input vectors. These vectors should be stored in your initial memory image.
2. Randomly generate 32 fixed weights (with values -1, 0, or 1). These weights should be stored in the initial memory image as well.
3. Apply each of the 1,000 input vectors and determine the neural network output.
4. Store the outputs in the output memory image (or append it to the input memory image).
5. Measure how many clock cycles this entire process requires. The lower that number, the better ☺.

For debugging purposes, I recommend you store a trace of the processor state for each time step in a text file.

Measuring Performance

You will need to maintain a global clock that allows you to keep track of the total number of cycles needed to execute your assembly code.

Weekly Progress Reports

In order to keep you on track, the project is broken down into weekly deliverables. Submitting weekly progress reports will get you 3pts per report. In addition, you will get valuable feedback. The feedback should allow you to know whether you are on track and are doing the right things.

Weekly deliverables can be fulfilled in three ways:

1. Submit a brief progress report on D2L, 1-2 pages at most.
2. Go to the TAs office hour (Fri, noon-2pm) and explain him the details of what you have been working on for that week. The explanation should be succinct (5min at most).
3. Come to one of my office hours (Tue/Thu, 8-9am).

The deliverables below are merely suggestions. Teams can go faster or slower, and/or skip steps. **What we care about first and foremost is to hear about your progress so that we can give you helpful feedback, in particular if we feel the team is headed in the wrong direction.** The criteria for getting the 3pts per weekly deliverable will be based solely on our judgment whether you made progress or not (compared to the previous week).

- 0pts: No progress report submitted or little to no progress shown
- 1pt: Some progress
- 2pts: Good/average progress
- 3pts: Excellent progress

List of suggested deliverables:

Week 3:

- Pick team partners. Teams should be no bigger than 3. If you can't find partners, post a message on D2L.

- Get organized, split up tasks.
- Study the basics of feed-forward threshold neural networks as explained above.
- **Weekly deliverable**: group composition

Week 4:
- Write some (pseudo) assembly code for the neural network described above. You could, for example, use the MIPS assembly instructions.
- If you want to, test your code with the MIPS MARS simulator to get a rough idea on the performance you could achieve on a standard MIPS architecture: http://courses.missouristate.edu/KenVollmar/MARS
- **Weekly deliverable**: pseudo assembly code for simulating your neural network

Week 5:
- Design a draft pipeline for your processor with a given number of stages.
- **Weekly deliverable**: Initial pipeline draft.

Week 6:
- **Weekly deliverable**: Refined pipeline (possibly with considering hazards)

Week 7:
- **Weekly deliverable**: Evidence that first parts of your code are working properly.

Week 8:
- **Weekly deliverable**: More evidence that most of your code works properly.

Week 9:
- **Weekly deliverable:** A quick demo of your entire processor with some initial performance data.

Week 10:
- Polish everything! Have the documentation proof-read by a friend who does not know anything about the project. Do things make sense to her/him?
- Upload the submission at least 24h before the deadline. If you need to make last-minute changes, simply over-write the previous submission(s).
- **Deliverable: final project.** See below for more details on want needs to be part of the final deliverables.
- **Submission deadline: Sun, Jun 5, 11:59pm.**

For Graduate Students Only

Graduate students taking this course at the 500-level are required to complete the following additional tasks:
- Search for **4 peer-reviewed journal papers** that describe neuromorphic architectures.
- **Summarize** each of the papers in your own words.
- Compared the 4 papers, i.e., highlight differences and similarities.

Optional Competition

Groups who provide evidence that they can simulate much larger networks than the one specified in Figure 1 (**NETWORK 1**) can enter a competition to win extra points. I will establish a ranking of the groups that can simulate the largest network in the lowest amount of clock cycles. The following network sizes and configurations are eligible for the competition:

- **NETWORK 2**: 1,500 neurons: 500 input neurons, 500 hidden neurons, 500 output neurons. 10,000 input patterns.
- **NETWORK 3**: 2,600 neurons: 500 input neurons, 2,000 hidden neurons, 100 output neurons. 20,000 input patterns.
- **NETWORK 3**: 22,000 neurons: 1,000 input neurons, 20,000 hidden neurons, 1,000 output neurons. 100,000 input patterns.

Extra points:

- 1st place: 8 extra points
- 2nd place: 6 extra points
- 3rd place: 4 extra points
- 4th place: 2 extra points
- 5th place: 1 extra points

Extra points above 100pts in the final project category will roll over into the other grade categories.

Final Demo

Teach team will have to do a brief demo (~5min) of their processor during finals week.

Final Project Deliverables

1. Your main deliverable will be a written report. The goal of the report is for me to evaluate and appreciate your work. Include al the necessary details in the report that I will need to know to evaluate your work. Do you rely on my going through your code and trying to figure out how you implemented your super-duper processr. **It is your job to present things nicely and in a comprehensive way in the report.**
2. The report **must include at least the following information/parts**:
    - **No** title page is required, but please make sure you list your names somewhere, e.g., in the document header.
    - **Introduction and motivation**. Clearly state the problem and the constraints that were given to you.
    - A description and a diagram (or several) of your complete processor architecture. Including a detailed description of your **pipelined microarchitecture**.
    - A detailed description of your **instruction formats** (mnemonics and hex codes).
    - **Justifications of your design choices**.
    - A description of your input/output memory image formats.
    - A section where you describe the **performance** of your processor. Include the number of clock cycles for at least NETWORK 1. If you want to claim extra points for NETWORKS 2-4, include that data as well.
    - Conclude the report summarizing the strengths **and** weaknesses of your design.
    - Make sure you state all your sources. You may want to add a bibliography at the end of your report. The bibliography counts toward your page limit.
3. If you are a graduate student, you must also submit the paper summaries of your 4 papers as well as the comparison (see section “For Graduate Students Only” for more details). You get two additional pages for that part.
4. Maximal length of report:
    - **10 pages (single-sided, single-spaced, one or two columns) for undergrads.**
    - **12 pages single-sided, single-spaced, one or two columns) for grad students**
5. Organize all your files neatly into folders, e.g., source code, documentation, examples, etc.
    - Make sure your source code has ample comments.
6. Include your **memory image** (in hex) and the **assembly code for your neural network** (in text format).

7. Include a README file that contains all the necessary instructions to compile and run your code.
8. Zip up everything and label the archive with YOURLASTNAME.zip.

- Upload the zip file into the D2L Final Project Dropbox. No hardcopies are accepted.

Grading

Category	**Undergraduate**	**Graduate**
Weekly deliverables (7x)	7 x 3pts = 21pts	7 x 3pts = 21pts
Final report	55pts	45pts
Creativity and innovation of your proposed solution	15pts	15pts
Demo	35pts	35pts
Literature summaries	-	10pts
Total	**126pts**	**126pts**

Additional Fine Print

- **This is a group project.** Groups sink or float as a group. See syllabus for more info on group projects.
- **Groups are composed of at most 3 team members.**
- **The solution you submit must be entirely your group's solution. No collaboration between groups is allowed.**
- Figures in the final report should not be hand-drawn. Use a drawing program to make them look more professional.
- The maximal length for the report is enforced to ensure equity. Please do not submit reports that are longer.
- The final project is due on **Sun, Jun 5, 11:59pm.**
- Demos will be scheduled during the final week.
- To keep a uniform policy and to ensure equity, no late submissions will be accepted.
- **State all your sources** of any materials (images, text, ideas, etc) you have used and make sure you have permission to use the material.
- This assignment may be updated if necessary. Updates will be posted on D2L.
- A list with project FAQs will be kept on D2L. It is your responsibility to check this list regularly. The list is part of this assignment.

Students who were in ECE 486/586 in the winter term and are re-taking this course have a choice between (1) doing the new project or (2) submitting an improved version of their winter term 2016 project. Please get in touch with me so that we can work out the details and agree on the expectations.