Harvard CS 242
At-Scale Computing

(Course Overview)
1/25/2016

Professor HT Kung
Harvard John A. Paulson School
of Engineering and Applied Sciences

Course abstract: http://www.eecs.harvard.edu/htk/courses/
Outline

- Motivations and background
- Approach and content of the course
- Administrative information
Why Do We Need This New Course on “At-Scale Computing”?  

- Data science, machine learning and big data are hot  
- Students need to find ways to differentiate themselves in this crowded field  
- This course will help by preparing students for upcoming new computational challenges  

How? See following slides
Emerging

- Deep learning: 1 billion parameters to train
  - Appetite for computing is unlimited. More precisely, *computational intensity* increases as the network depth grows. However, energy consideration is essential

- Internet of Things (IoT): 20 billion embedded devices expected in the next several years
  - Most of these devices will be battery powered and severely resource constrained in memory, computing power, etc.
  - In spite of these limitations, these devices need to perform intelligent tasks and be easy to use
Data-driven Deep Learning

Deep learning has been hugely successful for some important tasks in speech recognition, computer vision, and text understanding. For example:

- **Speech Recognition**: According to Microsoft’s speech group:
  - Using DL

- **ImageNet Classification with Deep Conv. Neural Networks**
  - Until 2012: Leading methods used hand-crafted features + encoding methods (e.g., SIFT+Bag-of-Words+SVM)
  - NIPS 2012, Alex Krizhevsky et. al
  - Significant improvement w.r.t other methods: ImageNet performance
    - 2010: ~28% (pre-convnet)
    - 2012: 16% (Krizhevsky)
    - 2013: 12%
    - 2014: 6.7%
Challenge in Convolutional Neural Network

Example of CNN architecture for handwritten digit recognition

- CNNs are successfully used for different visual pattern recognition applications such as action recognition, face detection, and object classification
- Major challenge is the long training time, which makes experimentation difficult

Image source: Efficiency Optimization of Trainable Feature Extractors for a Consumer Platform by Peemen et al.
## Face Recognition Example

Frame rate for face recognition CNN on three programmable platforms

<table>
<thead>
<tr>
<th>platform</th>
<th>input pixels</th>
<th>frames per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6 GHz Intel Pentium IV [3]</td>
<td>384x288</td>
<td>4.0</td>
</tr>
<tr>
<td>2.33 GHz 8-core Intel Xeon [1]</td>
<td>640x480</td>
<td>7.4</td>
</tr>
<tr>
<td>128-Core GPU Nvidia Tesla C870 [1]</td>
<td>640x480</td>
<td>9.5</td>
</tr>
</tbody>
</table>

- Results assume a relative low resolution and not yet meet real-time requirements
  - Processing 20 frames per second for 1280x720 HD video streams

- Problem gets even worse when the implementation platform is a low cost consumer platform (e.g., smartphone)

Source: Efficiency Optimization of Trainable Feature Extractors for a Consumer Platform by Peemen et al.
Parallelism

**Data Parallelism**
Distribute data, use same model

Forward

<table>
<thead>
<tr>
<th>Hidden layer I</th>
</tr>
</thead>
<tbody>
<tr>
<td>+200GB/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hidden layer II</th>
</tr>
</thead>
<tbody>
<tr>
<td>+200GB/s</td>
</tr>
</tbody>
</table>

| +200GB/s  |

Backward

<table>
<thead>
<tr>
<th>Exchange gradients</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x16GB/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exchange gradients</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x16GB/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exchange gradients</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x16GB/s</td>
</tr>
</tbody>
</table>

| +206GB/s          |

GPU cluster communication

| 5GB/s |

**Model Parallelism**
Distribute model, use same data

Forward

<table>
<thead>
<tr>
<th>Hidden layer I</th>
</tr>
</thead>
<tbody>
<tr>
<td>+200GB/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hidden layer II</th>
</tr>
</thead>
<tbody>
<tr>
<td>+200GB/s</td>
</tr>
</tbody>
</table>

| +200GB/s  |

<table>
<thead>
<tr>
<th>Hidden layer II</th>
</tr>
</thead>
<tbody>
<tr>
<td>+200GB/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exchange outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x16GB/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exchange outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x16GB/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Exchange outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x16GB/s</td>
</tr>
</tbody>
</table>

| +206GB/s          |

GPU cluster communication

| 5GB/s |

Sync needed for both forward and backward pass (after each dot product with the weight matrix)

(i.e., take their average, after each pass through a mini batch of training data)

**Pro:** Larger network can be trained
**Con:** Cost of sync increases with the number of GPU

**Pro:** Larger network can be trained
**Con:** Sync is necessary in all layers

Image source: http://timdettmers.com/
Scaling Computation and Energy: A Balanced Approach Desired

- **Scale up computation size** with only modestly increased energy consumption
  - We need such balanced computation scaling for deep neural networks
- **Scale down energy usage** with only modestly decreased outcome quality (e.g., prediction accuracy)
  - We need such balanced energy scaling for IoT devices and wearables

These considerations are related to a new field called “approximate computing,” which trades off quality with the effort expended
Balanced Scaling via Parallel and Distributed Computing

- Scale up computation size
  - Parallel use of many small, energy efficient compute nodes to accommodate large computations

- Scale down energy consumption
  - Coordinated use of multiple small, energy efficient compute nodes to support intelligent computations
Parallel and Distributed Computing

• Parallel computing: simultaneous use of multiple compute nodes over shared memory
  – E.g., computing over multi-core CPU on a single chip, or on an integrated CPU/GPU chip

• Distributed computing: simultaneous use of multiple compute nodes each working on their own memory
  – E.g., computing over clusters of servers on an interconnection network or an Ethernet in a datacenter, or on a wide-area network
Moore’s Law

• In 1965, Gordon Moore, an Intel co-founder, made a prediction that would set the pace for our modern digital revolution. From careful observation of an emerging trend, Moore extrapolated that computing would dramatically increase in power, and decrease in relative cost, at an exponential pace

• But slower pace is expected in the future. In July 2015 Intel CEO Brian Krzanich said, “the exponential advances in semiconductor manufacturing that enable faster and cheaper computing and storage every two years are now going to come closer to a rate of every two and half years”

• Rather than continuing to make microprocessors more and more powerful, we will work on new architectures and computing models which will leverage parallel processing to provide balanced computation and energy scaling, as noted earlier
Dennard Scaling Has Broken Down, So Lots of Dark Silicon (50~80% at 8nm)

- **Dennard scaling** states, roughly, that as transistors get smaller their power density stays constant, so that the power use stays in proportion with area: both voltage and current scale (downward) with length.
- Since around 2005–07, **Dennard scaling appears to have broken down**, so even though Moore's law continued for several years after that, it has not yielded dividends in improved performance. The primary reason cited for the breakdown is that at small sizes, current leakage poses greater challenges, and also causes the chip to heat up [Wiki].
- **Dark silicon** refers to circuitry on a chip that cannot be powered-on in order to keep the chip temperature in the safe operating range.
- **Implication**: We want designs which can dynamically select parts of the chip to power on (this course will give such design examples).
Outline

● Motivations and background
● Approach and content of the course
● Administrative information
This Course Is About Mapping Computations onto Multiple Compute Nodes

- In the old days, we worked on algorithms in isolation, such as Gaussian elimination in solving systems of linear equations.
- Today, we are concerned with efficient mapping of computation onto parallel and distributed systems.
- This is challenging due to system concerns such as (1) I/O, (2) communication, (3) synchronization, (4) memory allocation, (5) power consumption, and (6) software reuse.
- The output of our work is such mapping algorithms and software.
Mapping Computations for Parallel and Distributed Computing

This mapping task requires coordination between three areas:

- **Computational algorithms**
  - Computation is partitioned to support parallel processing with low communication and I/O overheads

- **Computer organizations**
  - Compute nodes are of various forms in their computing capabilities, energy consumptions, memory hierarchies, processor interconnections, etc.

- **Software abstractions**
  - High-level programming languages and models can be used to express and automate mapping
Approach of the Course

Students will acquire an integrated understanding of computational algorithms, computer organizations, and software abstractions via the following methods:

- Study end-to-end mapping examples
  - Two applications will be covered: (1) nearest neighbor computation and (2) convolutional neural networks
- Grasp basic concepts and principles
  - E.g., computational intensity, latency hiding, lower bounds
- Learn implications of various computer organizations
  - E.g., CPU, GPU, ASIC, FPGA, and clusters
- Gain experience by using open-source software platforms
  - E.g., MapReduce, GraphX/Apache Spark, OpenCL and TensorFlow
- Follow latest research
  - E.g., autotuning, automata processors, and neuromorphic spike-based computing

(We will have guest lectures by experts in industry and other institutions)
End-to-end: From Special Purpose to General Purpose, or Vice Versa

• ASIC
  – Voice recognition on Intel Curie with Pattern Matching ASIC

• IoT Microcontroller
  – Voice recognition on MediaTek LinkIt Assist 2502

• FPGA
  – Voice recognition on FPGAs

• Multi-Core CPU
  – Action recognition on Multi-Core CPU

• GPGPU
  – Action recognition on GPU

• Distributed Servers
  – For Data Clustering (K-means)
  – Action recognition on distributed servers

High-performance/ general purpose computing

Low-energy/ specialized accelerator

• Video encoder
• FFT
• Filtering
• Quantization
• Sparse Coding
• Convolutional Deep Neural Network (CDNN)
FPGA vs. GPU

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Best prior CNN on Virtex 7 485T [5]</td>
<td>-</td>
<td>46 images/sec³</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: Comparison of Image Classification Throughput and Power.

FPGAs can be more efficient than general-purpose CPUs on a performance-per-watt basis.

In November 2015, Intel announced that its first processor with performance-boosting FPGA will ship Q1 2016. This dual-chip package (integrated FPGA) can double the performance gain from an FPGA compared with using it as a discrete component. Qualcomm and IBM have similar efforts.
FPGA vs. ASIC

FPGA

- **Faster time-to-market**: No layout, masks or other manufacturing steps are needed
- **No upfront non-recurring expenses (NRE)**: Costs typically associated with an ASIC design
- **Simpler design cycle**: Due to software that handles much of the routing, placement, and timing
- **More predictable project cycle**: Due to elimination of potential re-spins, wafer capacities, etc.
- **Field reprogramability**: A new bitstream can be uploaded remotely
- **Reconfiguration time**: For large FPGAs it can take several hundred msec to reconfigure the board

ASIC

- **Full custom capability**: For design since device is manufactured to design specs
- **Lower unit costs**: For very high volume designs
- **Smaller form factor**: Since device is manufactured to design specs
An FPGA-Based CNN Processor Using a Data Flow Architecture

• The main components are: (1) a Control Unit (implemented on a general purpose CPU), (2) a grid of Processing Tiles (PTs), and (3) a Smart DMA interfacing external memory via a standard controller.

• The operators in the PTs are fully pipelined to produce one result per clock cycle.

Computation at a node takes place when operands arrive. There are not overheads such as instruction fetching and scheduling, which general-purpose processors would incur.
ASIC Architecture

• The architecture has several key components: *calculator*, *streamer*, and *flow-cpu*
ASIC Implementation and Hardware Performance Comparisons

- The design is synthesized and implemented in IBM 45 nm SOI STD-cell technology

<table>
<thead>
<tr>
<th></th>
<th>CPU¹</th>
<th>mGPU²</th>
<th>GPU³</th>
<th>neuV6⁴</th>
<th>neuIBM⁵</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak GOPs</td>
<td>10</td>
<td>182</td>
<td>1350</td>
<td>160</td>
<td>160</td>
</tr>
<tr>
<td>Real GOPs</td>
<td>1.1</td>
<td>54</td>
<td>294</td>
<td>147</td>
<td>147</td>
</tr>
<tr>
<td>Power (W)</td>
<td>30</td>
<td>30</td>
<td>220</td>
<td>10</td>
<td>0.579</td>
</tr>
<tr>
<td>GOPs/W</td>
<td>0.04</td>
<td>1.8</td>
<td>1.34</td>
<td>14.7</td>
<td>254</td>
</tr>
</tbody>
</table>

¹ CPU: Intel DuoCore, 2.7GHz, optimized C code
²-³ mGPU, GPU: a mobile Nvidia GT335m and a high-end GTX480
⁴ neuV6: neuFlow prototyped Xilinx Virtex 6 FPGA
⁵ neuIBM: 45nm IBM SOI process neuFlow (this work)
Performance Comparisons in Applications

- Performance results on a number of canonical tasks (face recognition, MNIST, etc.)
A Grant Challenge in Reducing Energy in Computing

• C. Mead “Neuromorphic Electronic Systems,” Proc. IEEE 1990:
  – The human brain has $10^{16}$ synapses, with a nerve pulse arriving at each synapse per 100ms. This means $10^{16}$ complex operations per second. At the power dissipation of 1 W (instead of 20W for simplicity here), the brain does each operation only $10^{-16}$ J
  – In contrast, the ultimate silicon technology will dissipate $10^{-9}$ J for each operation

• In the meantime, we need speed improvement in “instant perception.” We notice things instantly without having to go through many layers of neurons
Neuromorphic Computing

- Morphic is defined as in a specific shape or form. **Neuromorphic** means having behaviors of neurons.

- Many early efforts
  - Neurogrid is a multi-chip system developed by Kwabena Boahen and his group at Stanford, 2005 (6,000 synapses)
  - Fast Analog Computing with Emergent Transient States (FACETS), 2005 (50 million synapses)
  - Torres-Huitzil’s FPGA Model, 2005
  - …

- One of the latest efforts: IBM’s TrueNorth is one of several (next slide)
IBM’s TrueNorth Architecture: Towards Neuromorphic Computing

Fig. 2. TrueNorth architecture. Panels are organized into rows at three different scales (core, chip, and multicore) and into columns at four different views (neuroscience inspiration, structural, functional, and physical). (A) The neurosynaptic core is loosely inspired by the idea of a canonical cortical microcircuit. (B) A network of neurosynaptic cores is inspired by the cortex’s two-dimensional sheet. (C) The multichip network is inspired by the long-range connections between cortical regions shown from the macaque brain (J0). (D) Structure of a neurosynaptic core with axons as inputs, neurons as outputs, and synapses as directed connections from axons to neurons. Multicore networks at (E) chip scale and (F) multichip scale are both created by connecting a neuron on any core to an axon on any core with point-to-point connections. (G) Functional view of core as a crossbar where horizontal lines are axons, cross points are individually programmable synapses, vertical lines are neuron inputs, and triangles are neurons. Information flows from axons via active synapses to neurons. Neuron behaviors are individually programmable, with two examples shown. (H) Functional chip architecture is a two-dimensional array of cores where long-range connections are implemented by sending spike events (packets) over a mesh routing network to activate a target axon. Axonal delay is implemented at the target. (I) Routing network extends across chip boundaries through peripheral merge and split blocks. (J) Physical layout of core in 28-nm CMOS fits in a 240-μm-by-390-μm footprint. A memory (static random-access memory) stores all the data for each neuron, a time-multiplexed neuron circuit updates neuron membrane potentials, a scheduler: buffers incoming spike events to implement axonal delays, a router relays spike events, and an event-driven controller orchestrates the core’s operation. (K) Chip layout of 64-by-64 core array wafer, and chip package. (L) Chip periphery to support multichip networks.
TrueNorth’s Energy Saving

- The TrueNorth consumes about $10^{-13}$ J per operation, rather than $10^{-9}$ J in general-purpose computing.
- Recall that for human brain this number is $10^{-16}$ J. Thus we still have three orders of magnitude to go.
Eight Modules of the Course (in Rough Chronological Order)

- Big picture
  - Use of parallel and distributed computing to achieve high performance and energy efficiency
- End-to-end example 1
  - Mapping nearest neighbor computation onto parallel computing units in the forms of CPU, GPU, ASIC and FPGA
- Communication and I/O
  - Latency hiding with prediction, computational intensity, lower bounds
- Computer architectures and implications to computing
  - Multi-cores, CPU, GPU, clusters, accelerators, and virtualization
- End-to-end example 2
  - Mapping convolutional neural networks onto parallel computing units in the forms of CPU, GPU, ASIC, FPGA and clusters
- Great inner loops and parallelization for feature extraction, data clustering and dimension reduction
  - PCA, random projection, clustering (K-means, GMM-EM), sparse coding (K-SVD), compressive sensing, FFT, etc.
- Software abstractions and programming models
  - MapReduce (PageRank, etc.), GraphX/Apache Spark, OpenCL and TensorFlow
- Advanced topics
  - Autotuning, automata processors, and neuromorphic spike-based computing
Outline

- Motivations and background
- Approach and content of the course
- Administrative information
A Student’s Work: Deliverables

• Readings and midnight quizzes: Students will be responsible for completing reading assignments before lectures, and answering a few short questions by midnight before the next lecture
• Two programming assignments
• Midterm: There will be an open book midterm based on lecture material to be held April 13 in class
• Course project: Students will complete a project of their own design in 2-person or 3-person groups. The work for the project will include a project proposal presentation, project discussion with teaching staff, interim checkpoint reviews, a final project presentation, and a final project report
• Study research papers in the field and give presentations
• Attend classes and participate in classroom discussion
Final Course Projects

- The projects will need to be original with compelling application/systems motivations
- Each project must produce a piece of software related to mapping computations to parallel and distributed computing systems
  - For example, you may implement an asynchronous parallel clustering algorithm (this can actually be really cool!)
- Students will formulate their projects early in the course, so there will be sufficient time for discussion and iterations with the teaching staff, as well as for system design and implementation
Collaboration Policy

1. Students may talk generally about the subject matter of the class and lectures with one another, as this is an excellent way to better learn and understand the material.

2. However, they cannot talk about or collaborate on the midnight quizzes. If clarifications are needed on questions, students should contact the teaching staff directly via Piazza or come to office hours. When students ask for clarifications on midnight quizzes, responses will be posted on Piazza.

3. Students form 2-person or 3-person teams may collaborate on course projects and related research presentations. For this work, students may only work within their own groups. Projects can be small, but must be original work by students for this course. Copying work used in another course or projects by others is not allowed.
Late Day Policy

• No late days are allowed for midnight quiz submissions
• Note, however, that the bottom two quiz scores received on submitted questions will be ignored for each student
• A total of five late days are allowed for the two programming assignments
Administrative Information

1. Instructor:
   - HT Kung <kung@harvard.edu>
     (see http://www.eecs.harvard.edu/htk/)

2. TFs:
   - Miriam Cha <miriamcha@g.harvard.edu>
   - Marcus Comiter <mcomiter@g.harvard.edu>

3. Lecture notes and lab materials will be available. No textbooks are required

4. Lab sessions will be announced online

5. Course Website:
   - https://canvas.harvard.edu/courses/9365
   - Course message board on Piazza. To enroll, visit http://piazza.com/harvard/spring2016/cs242
Course Grading Formula

1. Answers to midnight quizzes: 20%
2. Helpful comments on Piazza: 5%
3. Classroom participation and discussion: 5%
4. Programming assignments: 25%
5. Open-book midterm: 10%
6. Research papers presentation: 10%
7. Project proposal: 10%
8. Project presentation and report: 15%
Prerequisites

1. Programming experience (Python, MatLab or C/C++ should be fine)
2. Basic knowledge in systems and machine organization
3. Familiarity in data structures and algorithms
4. Maturity in mathematics (e.g., undergraduate linear algebra and statistics)

For students with strong interest in the subject matter and related research topics, one of these four requirements may be waived. Labs and extra support will provide preparation in the first weeks of the semester to help students quickly obtain the background necessary to excel in the course.
For Students, Expected Intellectual Outcome of The Course

- Techniques and examples of mapping computations into forms of great energy efficiency while delivering acceptable quality in the computing results
- Programming models that can facilitate such mapping schemes
- Appreciation of the close interaction among computational algorithms, software abstractions and computer organizations
Reasons for Taking This Course

- Learn principles, algorithms, architectures and programming skills in scaling computation and energy
- Gain experience with how to carry out a project (final course project), make presentations, and write technical reports
- Develop teamwork skills, and, hopefully, also do some research
- Enjoy an interactive, seminar-style classroom experience
- Differentiate yourselves in job markets, or prepare for research in computer science, and computing in general
Course Sign Up

1. If you are interested in taking the class (I hope you are), please email: 
   kung@harvard.edu
   with YES on the subject line

2. This will help the teaching staff determine resources required for the course

3. This will also let you receive course announcements immediately