Anthony's Research Projects

- SecML – Secure Machine Learning in the presence of adversaries
- Big Data Genomics – Algorithms and systems programming for population-scale genomics analyses

Adversarial Exploitation of ML

- Machine Learning is everywhere
  - Search rankings, spam detection, financial models, self driving cars …
- Adversaries everywhere! Traditional approach – Evading Adversary
  - Attacker determines decision boundary
  - Crafts (positive instance) content that is classified as negative
- Input data "drifts" over time, so must periodically retrain ML
  - Use previously classified input data
- Newer approach – Influencing Adversary
  - Patient attacker operates during periodic retraining stage by injecting "tricky" positive instances
  - Shifts decision boundary over time during retraining such that (positive instance) content is eventually classified as negative
- We need novel adaptive, robust ML techniques to defend against Influencing Adversaries
  - Also Active Learning with Humans-in-the-Loop

Our Current Domain and Dataset

- Windows malware detection
  - Ever changing, evading adversary with novel attacks
- Over 1 million unique executables from VirusTotal
  - Seen over 5 million times
  - Spanning January 2012 to June 2014
  - ~1% of VirusTotal’s executables for the period
- Labeled by 32 AV providers’ static engines
  - Threshold of four detections to label as malicious
- 85% malicious (3,000 to 406,000 families)
- The largest academic malware detection dataset
An ML Analysis Pipeline: From Labs to Production

- Goal: research AND production environment
  - Moonshot: Detect undetectable Advanced Persistent Threats

- Significant challenges to deploying ML in practice:
  - Increased human insight – how the system is making decisions
  - Scalable, interactive analysis – large-scale data, human-driven exploration
  - Time series analysis – repeatable scientific experiments to build confidence in ML
  - Resource management – variable cost analyses and experts
  - Adversarial resilience – detecting attempts to manipulate or evade detection

Secure Active Learning Testbed (SALT)

Detection Pipeline

Evaluation of Secure Active Learning Testbed (SALT)

More accurate than best x86 antivirus providers on VirusTotal\(^*\)

Process 2.5 year, 1.1 million executable dataset in hours

Some components in production at an AV provider

\(^*\)Using AV providers static engines

Active Learning Results

Comparison of Classifier Accuracy to Vendor Labels

Active learning queries to oracle

Built on Apache Spark for scalability
Active Learning Results

Comparison of Classifier Accuracy to Vendor Labels

80 queries/day
0 queries/day

80 queries/day
0 queries/day

Moonshot Results
2334 samples were undetected by all AV vendors
SALT with Active ML detected:
• 517 (22%) at 1% FPR
• 364 (16%) at 0.3% FPR

Lots of ongoing research into evading and influencing adversaries

SecML Acknowledgements

• UC Berkeley: David Fifield, Alex Kantchelian, Brad Miller, Qi Zhong, Vaishaal Shankar, Tony Wu, George Yu, Riyaz Faizullahbey, Doug Tygar, Anthony D. Joseph

• Drexel: Eleanor Cawthon, Edwin Dauber, Rachel Greenstadt

• ICSI: Sadia Afroz, Michael Tshantz

• Intel: Ling Huang, Rekha Bachwani, Catherine Huang

• Mitsubishi: Takumi Yamamoto

Why Do We Need Big Genomic Data?

Precision Medicine!
• Understand genetic causes of diseases
• Choose more effective drugs and avoid adverse reactions

A Precision Medicine First

• Joshua Osborn (14 years old)
  – Visited ER 3 times in 4 months with encephalitis (brain swelling)
  – Hospitalized after 3rd visit and placed into a medically induced coma due to uncontrollable seizures

• Doctors tried traditional medicine approaches
  – Over 100 viral/fungal/bacterial pathogens cause encephalitis
  – Lots and lots of tests for different pathogens
  – 1 cm³ of brain tissue biopsied

• Last resort – sent spinal fluid to UCSF for DNA sequencing
  – Needle in a haystack – throw out human reads (99.984%)
  – 475 reads out of 3M from rare Leptospira santarosai bacteria → given antibiotics, out of coma in week, discharged 4 wks
• US: 15,000 cases a year of encephalitis with 2,000 deaths (>70% of deaths underdiagnosed)
• Worldwide: 70,000 diagnosed cases a year with 25,000 deaths

Time to Break from Legacy Compute!

• Large cohorts generate massive amounts of data
  – 100+ GB of sequence data per sample
  – 1+ GB of variant data per sample
  – US Veterans Administration’s Million Veterans Project (210 PB)
    » Enrolled 500,000th veteran in August 2016
  – UK 100K Genome Project (21 PB)
    » One goal is to analyze rare diseases (<1 in 2,000 people)
  – US National Institutes of Health’s All of Us Project (210 PB)
    » 1+ million ordinary Americans

• Many historical issues:
  – Processing genomic data on expensive, non-scalable HPC
  – Legacy file formats designed for single machine
  – Algorithms tied to file formats

Our Approach

• Bring together cross cutting researchers
  – We have CS, Bio, HPC backgrounds

• Leverage fact that most genomic analysis steps are parallel

• Rethink design of genomics tools for cloud/commodity computing
  – Easy to write and use horizontally scalable genomics algorithms

• Apply Open Source Software ethos
  – We build tools that people freely use and build upon

• Bring emphasis on collaboration:
  – Work closely with UCSC CGL, MSR, OHSU, Stanford, Mt. Sinai
Cloud-scale Genomic Analyses

- Going from raw genomic reads (100+ GB) to aligned genomes (3+ GB) to the differences between us (1GB) to actionable insights

- Projects: Toil, ADAM, avocado, mango, gnocchi, cannoli

- Many algorithmic challenges:
  - Complex preprocessing and filtering methods
  - ML algorithms at genome- and population-scale

- Many infrastructure challenges:
  - Volume of data: VA MVP, NIH AoU Project
  - How do we run geo-distributed workflows efficiently?
  - Can bioinformatics be made reproducible?

ADAM: A Stack-based Design

A Well Designed Stack Simplifies Application Design

<table>
<thead>
<tr>
<th>Application Transformations</th>
<th>Enriched Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentation Enriched Models</td>
<td>Enriched Read/Variant</td>
</tr>
<tr>
<td>Evidence Access MapReduce/DBMS</td>
<td>Spark, Spark-SQL, Hadoop</td>
</tr>
<tr>
<td>Schema Data Models</td>
<td>Avro-Schema for reads, variants, and genotypes</td>
</tr>
<tr>
<td>Materialized Data Columnar Storage</td>
<td>Load data from Parquet and legacy formats</td>
</tr>
<tr>
<td>Data Distribution Parallel FS</td>
<td>HDF5, Tachyon, HPC file systems, S3</td>
</tr>
<tr>
<td>Physical Storage Attached Storage</td>
<td>Disk, SDD, block store, memory cache</td>
</tr>
</tbody>
</table>

Define a Schema, Query Genomic Data via Schema

```java
record SignalRead {
    union { null, string } chrom = null;
    union { null, long } start = null;
    union { null, long } end = null;
    union { null, int } mapq = null;
    union { null, string } readName = null;
    union { null, string } sequence = null;
    union { null, string } mateReference = null;
    union { null, long } mateAlignmentStart = null;
    union { null, string } cigar = null;
    union { null, string } qual = null;
    union { null, string } recordGroupName = null;
    union { int, null } basesTrimmedFromStart = 0;
    union { int, null } basesTrimmedFromEnd = 0;
    union { boolean, null } readPaired = false;
    union { boolean, null } properPair = false;
    union { boolean, null } readMapped = false;
    union { boolean, null } mateMapped = false;
    union { boolean, null } firstOfPair = false;
    union { boolean, null } secondOfPair = false;
    union { boolean, null } failedVendorQualityChecks = false;
    union { boolean, null } duplicateRead = false;
    union { boolean, null } readNegativeStrand = false;
    union { boolean, null } mateNegativeStrand = false;
    union { boolean, null } primaryAlignment = false;
    union { boolean, null } secondaryAlignment = false;
    union { boolean, null } supplementaryAlignment = false;
    union { null, string } mismatchingPositions = null;
    union { null, string } origQual = null;
    union { null, string } attributes = null;
    union { null, string } recordGroupSequencingCenter = null;
    union { null, string } recordGroupDescription = null;
    union { null, long } recordGroupRunDateEpoch = null;
    union { null, string } recordGroupFlowOrder = null;
    union { null, string } recordGroupKeySequence = null;
    union { null, string } recordGroupLibrary = null;
    union { null, int } recordGroupPredictedMedianInsertSize = null;
    union { null, string } recordGroupPlatform = null;
    union { null, string } recordGroupPlatformUnit = null;
    union { null, string } recordGroupSample = null;
    union { null, Contig } mateContig = null;
}
```
Parallelize Common Access Patterns

- Efficient parallel access methods

- We provide parallel methods for processing genomic data with Spark
- Makes use of genomics specific reference genome structure to accelerate queries
- E.g., region join for high performance overlap queries

Pick Appropriate Storage

- Efficient physical storage formats

- When accessing scientific datasets, we frequently slice and dice the dataset:
  - Algorithms may touch subsets of columns
  - We don’t always touch the whole dataset
- This is a good match for columnar storage like Apache Parquet
- Also, can support legacy formats

Want to Learn More?

- Check out the code!
  - ADAM: https://github.com/bigdatagenomics/adam
- Check out a demo!
  - https://databricks.com/blog/2016/05/24/genome-sequencing-in-a-nutshell.html
- Run ADAM in Databricks Community Edition!
  - http://goo.gl/xK8x7s

Acknowledgements

- UC Berkeley: Matt Massie, Timothy Danford, André Schumacher, Jay Kottalam, Karen Feng, Eric Tu, Niranjan Kumar, Ananth Pallaseni, Michael Heuer, Alyssa Morrow, Frank A. Nothaft, Devin Petersohn, Gunjan Baid, Justin Paschal, Anthony D. Joseph, Dave Patterson
- Mt. Sinai: Arun Ahuja, Neal Sidhwaney, Ryan Williams, Michael Linderman, Jeff Hammerbacher
- GenomeBridge: Carl Yeksigian
- Cloudera: Uri Laserson, Tom White
- Microsoft Research: Ravi Pandya, Bill Bolosky
- UC Santa Cruz: Benedict Paten, David Haussler, Hannes Schmidt, Beau Norgeot, Audrey Musselman-Brown, John Vivian, Jacob Pfeil
- And many other open source contributors, especially Neil Ferguson, Andy Petrella, Xavier Tordoir
- Total of >60 contributors to ADAM/BDG from >12 institutions
Administrivia

• Midterm 3 coming up on **Wed 4/25 6:30-8PM**
  – All topics up to and including Lecture 23
    » Focus will be on Lectures 17 – 23 and associated readings, and Projects 3
    » But expect 20-30% questions from materials from Lectures 1-16
  – LKS 245, Hearst Field Annex A1, VLSB 2060, Barrows 20, Wurster 102 (see Piazza for your room assignment)
  – Closed book
  – 2 pages hand-written notes both sides

• Please fill out the online course evaluation
Visual computing demands orders of magnitude more performance.

The biggest data is visual

YouTube: 400 hrs uploaded / min
[Brewer et al. 2016]
1.5 Terapixels/sec

250 M surveillance cameras,
2.5 B cell phone cameras, …

Rendering:
insatiable demand for computation

Modern game
2 M pixels
1 M polys
10 ms/frame

Tintin, Avatar
8 M pixels
5 G polys
5 hrs/frame

6 orders of magnitude more computation

Images by Valve, Weta

Pervasive sensing: “the cloud” is not enough data transfer » capture

Sensor + Read out
5 M pixels
~1 mJ/frame

LTE radio
50 Mbit/sec
1 W
~1 J/frame

transmission power costs 1,000x capture

[Wu et al. 2012]
Visual data analysis is expensive

One object recognition neural net:
250 Watt GPU $\rightarrow$ 0.05 megapixels at video rate

(Caffe + cuDNN on K40c running VGG16)

Simonyan & Zisserman 2015
image: Noh et al. 2015

Your data-intensive problem here...

Programmer productivity has exploded

1990s
C/C++

2010s
node.js

Writing high-performance code is hard

Reference:
300 lines C++

Adobe: 1500 lines
3 months of work
10x faster
My group’s research:
Compilers, systems, architectures, and algorithms for high-performance graphics & visual computing.

Reorganize computations & data.
- Simpler programs
- Order of magnitude faster
- Scalable on future architectures

How can we increase performance and efficiency?

Parallelism
“Moore’s law” scaling requires exponentially more parallelism.

Locality
Data should move as little as possible.

Communication dominates computation in both energy and time

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU op</td>
<td>1 pJ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load from SRAM</td>
<td>5 pJ</td>
<td>5x</td>
<td></td>
</tr>
<tr>
<td>Move 10mm on-chip</td>
<td>32 pJ</td>
<td>32x</td>
<td></td>
</tr>
<tr>
<td>Send off-chip</td>
<td>500 pJ</td>
<td>500x</td>
<td></td>
</tr>
<tr>
<td>Send to DRAM</td>
<td>1 nJ</td>
<td>1,000x</td>
<td></td>
</tr>
<tr>
<td>Send over LTE</td>
<td>&gt; 50 μJ</td>
<td>50,000,000x</td>
<td></td>
</tr>
</tbody>
</table>

Message #1: Performance requires complex tradeoffs
Where does performance come from?

- Hardware
- Program

Message #2: organization of computation is a first-class issue

Program:

- Algorithm
- Organization of computation
- Hardware

Reorganizing computation is painful

Reference:
- 300 lines C++
- Adobe: 1500 lines
- 3 months of work
- 10x faster (vs. reference)

Same algorithm, Different organization

Global reorganization breaks modularity

The algorithm uses 8 pyramid levels
Halide
a language and compiler
for image processing & vision
[SIGGRAPH 2012,
PLDI 2013,
SIGGRAPH 2016,
SIGGRAPH 2018, ...]

Algorithm vs. Organization: 3x3 blur

for (int x = 0; x < input.width(); x++)
for (int y = 0; y < input.height(); y++)
blurH(x, y) = (input(x-1, y) + input(x, y) + input(x+1, y))/3;

for (int y = 0; y < input.height(); y++)
for (int x = 0; x < input.width(); x++)
blurV(x, y) = (blurH(x, y-1) + blurH(x, y) + blurH(x, y+1))/3;

Same algorithm, different organization
One of them is 15x faster
Hand-optimized C++
9.9 → 0.9 ms/megapixel
void box_filter_3x3(const Image &in, Image &out) {
  // Allocate tile blur array
  __m128i one_third = _mm_set1_epi16(0); // Hand-optimized C++
  // Allocate tile blur array
  const uint16_t *inPtr = &in[0][0];
  // Allocate tile blur array
  __m128i one_third = _mm_set1_epi16(0); // Hand-optimized C++
  // Allocate tile blur array
  for (int yTile = 0; yTile < in.height(); yTile++) {
    // Allocate tile blur array
    __m128i one_third = _mm_set1_epi16(0); // Hand-optimized C++
    // Allocate tile blur array
    __m128i one_third = _mm_set1_epi16(0); // Hand-optimized C++
    for (int xTile = 0; xTile < in.width(); xTile++) {
      // Allocate tile blur array
      __m128i one_third = _mm_set1_epi16(0); // Hand-optimized C++
      // Allocate tile blur array
      __m128i one_third = _mm_set1_epi16(0); // Hand-optimized C++
      a = _mm_loadu_si128((__m128i*)&inPtr[yTile + xTile]);
      b = _mm_loadu_si128((__m128i*)&inPtr[yTile + xTile + 1]);
      c = _mm_loadu_si128((__m128i*)&inPtr[yTile + xTile + 2]);
      // Allocate tile blur array
      __m128i one_third = _mm_set1_epi16(0); // Hand-optimized C++
      // Allocate tile blur array
      __m128i one_third = _mm_set1_epi16(0); // Hand-optimized C++
      sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
      // Allocate tile blur array
      __m128i one_third = _mm_set1_epi16(0); // Hand-optimized C++
      // Allocate tile blur array
      __m128i one_third = _mm_set1_epi16(0); // Hand-optimized C++
      avg = _mm_mulhi_epi16(sum, one_third);
      // Allocate tile blur array
      __m128i one_third = _mm_set1_epi16(0); // Hand-optimized C++
      // Allocate tile blur array
      __m128i one_third = _mm_set1_epi16(0); // Hand-optimized C++
      _mm_store_si128(outPtr++, avg);
    }
  }
}

Optimized 3x3 blur in C++
parallelism
distribute across threads
SIMD parallel vectors

Tiled, fused
Vectorized
Multithreaded
Redundant computation
Near roof-line optimum

Traditional languages conflate algorithm & organization

Optimized 3x3 blur in C++
parallelism
distribute across threads
SIMD parallel vectors

Local reorganize computation: fuse two blurs, compute in tiles
(Re)organizing computation is hard

Optimizing parallelism, locality requires transforming program & data structure.

What transformations are legal?

What transformations are beneficial?

libraries don’t solve this:
BLAS, MKL, OpenCV, PyTorch, TensorFlow

optimized kernels compose into inefficient pipelines (no fusion)

Halide’s answer:
*decouple* algorithm from schedule

Algorithm: *what* is computed
Schedule: *where* and *when* it’s computed

The algorithm defines pipelines as pure functions

Pipeline stages are functions from coordinates to values

Execution order and storage are unspecified
no explicit loops or arrays

3x3 blur as a Halide algorithm:

\[
\begin{align*}
\text{blurH}(x, y) &= (\text{input}(x-1, y) + \text{input}(x, y) + \text{input}(x+1, y))/3; \\
\text{blurV}(x, y) &= (\text{blurH}(x, y-1) + \text{blurH}(x, y) + \text{blurH}(x, y+1))/3;
\end{align*}
\]

Domain scope
of the programming model

All computation is over regular grids.

Only feed-forward pipelines
Iterative computations are a (partial) escape hatch.

Iteration must have bounded depth.

Dependence must be inferable.
User-defined clamping can impose tight bounds, when needed.

Long, heterogeneous pipelines.
Complex graphs, deeper than traditional stencil computations.
How can we organize this computation?

Organizing a data-parallel pipeline

Simple loops execute **breadth-first** across stages

Parallelism
Breadth-first execution **sacrifices locality**

Interleaved execution (fusion) **improves locality**

Understanding dependencies
Stencils have overlapping dependencies

Sliding window execution **sacrifices parallelism**

Breaking dependencies with tiling

Decoupled tiles optimize **parallelism & locality**
Breaking dependencies introduces redundant work

Message #1: performance requires tradeoffs

Message #2: algorithm vs. organization

Order and interleaving radically alter performance of the same algorithm.
The schedule defines intra-stage order, inter-stage interleaving

For each stage:
1) In what order should we compute its values?
2) When should we compute its inputs?

This is a language for scheduling choices.
Schedule primitives **compose** to create many organizations

### Halide

0.9 ms/megapixel

**Func** box_filter_3x3(Func in) {
    Func blurH, blurV;
    Var x, y, xi, yi;

    // The algorithm - no storage, order
    blurH(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;
    blurV(x, y) = (blurH(x, y-1) + blurH(x, y) + blurH(x, y+1))/3;

    // The schedule - defines order, locality; implies storage
    blurV.tile(x, y, xi, yi, 256, 32).vectorize(xi, 8).parallel(y);
    blurH.compute_at(blurV, x).store_at(blurV, x).vectorize(x, 8);

    return blurV;
}

### C++

0.9 ms/megapixel

```cpp
void box_filter_3x3(const Image In, Image& blurV) {
    for (int yTile = 0; yTile < In.height(); yTile += 32) {
        for (int xTile = 0; xTile < In.width(); xTile += 32) {
            vectorize(y);
            parallel(x);
            blurV.tile(x, y, xi, yi, 256, 32).vectorize(xi, 8).parallel(y);
            blurH.compute_at(blurV, x).store_at(blurV, x).vectorize(x, 8);
        }
    }
}
```
Organization requires global tradeoffs

local Laplacian filters
[Paris et al. 2010, Aubry et al. 2011]

Adobe: 1500 lines
expert-tuned C++
multi-threaded, SSE
3 months of work
10x faster than original C++

Halide: 60 lines
1 day

Halide vs. Adobe:
2x faster on same CPU
10x faster on GPU

Halide is all around you!
open source at http://halide-lang.org

> 1000 pipelines
10s of kLOC in production
What’s next?

Differentiable programming languages: bringing gradient-based optimization to arbitrary programs.

What’s next?

Training compilers to automatically optimize organization using reinforcement learning.

What’s next?

Building the “programmable GPU” of image processing & vision.
What’s next?
Turning AWS Lambda into a supercomputer accessible from your laptop

Where you can learn more:
CS 164 - Compilers & Programming Languages
CS 294-jrk - Domain-Specific Languages
Talk to me about research!

Thank you!

• Let’s Thank the TAs!
• Good luck on midterm 3!