Efficient Computing for AI and Robotics

Vivienne Sze
Massachusetts Institute of Technology


Contact Info
email: sze@mit.edu
website: www.rle.mit.edu/eems
Processing at “Edge” instead of the “Cloud”

Communication  Privacy  Latency
Cameras and radar generate \(~6\) gigabytes of data every 30 seconds.

Self-driving car prototypes use approximately \(2,500\) Watts of computing power.

Generates wasted heat and some prototypes need water-cooling!
Existing Processors Consume Too Much Power

< 1 Watt

> 10 Watts
Transistors are NOT Getting More Efficient

Slow down of Moore’s Law and Dennard Scaling

*General purpose microprocessors not getting faster or more efficient*

- Need **specialized hardware** for significant improvement in speed and energy efficiency
- Redesign computing hardware from the ground up!
Energy-Efficient Computing with Cross-Layer Design

**Algorithms**

- Convolutions
- Pooling
- Convs
- Linear Classifier
- Object Categories / Positions
- F4 maps
- S2 feature maps
- C1 feature maps
- C3 feature maps

**Systems**

**Architectures**

- Link Clock: Core Clock
- Filter
- Input Image
- Decomp
- SRAM 108KB
- Output Image
- Comp
- ReLU
- DCNN Accelerator
- 14x12 PE Array
- Off-Chip DRAM
- 64 bits

**Circuits**

- On-Chip Buffer
- Spatial PE Array

---

MIT

RESEARCH LABORATORY OF ELECTRONICS AT MIT

MTL 

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
### Power Dominated by Data Movement

<table>
<thead>
<tr>
<th>Operation</th>
<th>Energy (pJ)</th>
<th>Relative Energy Cost</th>
<th>Area (µm²)</th>
<th>Relative Area Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>8b Add</td>
<td>0.03</td>
<td></td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>16b Add</td>
<td>0.05</td>
<td></td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>32b Add</td>
<td>0.1</td>
<td></td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>16b FP Add</td>
<td>0.4</td>
<td></td>
<td>1360</td>
<td></td>
</tr>
<tr>
<td>32b FP Add</td>
<td>0.9</td>
<td></td>
<td>4184</td>
<td></td>
</tr>
<tr>
<td>8b Mult</td>
<td>0.2</td>
<td></td>
<td>282</td>
<td></td>
</tr>
<tr>
<td>32b Mult</td>
<td>3.1</td>
<td></td>
<td>3495</td>
<td></td>
</tr>
<tr>
<td>16b FP Mult</td>
<td>1.1</td>
<td></td>
<td>1640</td>
<td></td>
</tr>
<tr>
<td>32b FP Mult</td>
<td>3.7</td>
<td></td>
<td>7700</td>
<td></td>
</tr>
<tr>
<td>32b SRAM Read (8KB)</td>
<td>5</td>
<td></td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>32b DRAM Read</td>
<td>640</td>
<td></td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

Memory access is **orders of magnitude** higher energy than compute

[Horowitz, ISSCC 2014]
Deep Neural Networks (DNNs) have become a cornerstone of AI.

- **Computer Vision**
- **Speech Recognition**
- **Game Play**
- **Medical**
DNNs for Understanding the Environment

Depth Estimation

Semantic Segmentation

State-of-the-art approaches use **Deep Neural Networks**, which require **up to several hundred millions of operations and weights** to compute!

> 100x more complex than video compression
Properties We Can Leverage

• Operations exhibit **high parallelism** → **high throughput** possible

• Memory Access is the Bottleneck

```
Memory Read       MAC*              Memory Write

DRAM → filter weight → ALU → updated partial sum
DRAM ↣ image pixel ↣ ALU ↣ partial sum
DRAM ↣ partial sum ↣ ALU ↣ updated partial sum

200x 1x

* multiply-and-accumulate
```

**Worst Case:** all memory R/W are **DRAM** accesses

• Example: AlexNet has **724M** MACs → **2896M** DRAM accesses required
Properties We Can Leverage

- Operations exhibit **high parallelism** → **high throughput** possible
- **Input data reuse** opportunities (**up to 500x**)
Exploit Data Reuse at Low-Cost Memories

Specialized hardware with small (< 1kB) low cost memory near compute

**Normalized Energy Cost**

- 1× (Reference)
- 1×
- 2×
- 6×
- 200×

* measured from a commercial 65nm process

Farther and larger memories consume more power
Weight Stationary (WS)

- Minimize weight read energy consumption
  - maximize convolutional and filter reuse of weights

- Examples:
  - [Chakradhar, ISCA 2010]
  - [nn-X (NeuFlow), CVPRW 2014]
  - [Park, ISSCC 2015]
  - [Origami, GLSVLSI 2015]
Output Stationary (OS)

- Minimize partial sum R/W energy consumption
  - maximize local accumulation

- Examples:
  [Gupta, ICML 2015]  [ShiDianNao, ISCA 2015]
  [Peemen, ICCD 2013]
Row Stationary Dataflow

- Maximize row **convolutional reuse** in RF
  - Keep a **filter** row and **fmap** sliding window in RF
- Maximize row **psum accumulation** in RF
Row Stationary Dataflow

Optimize for **overall energy efficiency** instead for only a certain data type
Evaluate Reuse in Different Dataflows

- **Weight Stationary**
  - Minimize movement of filter weights

- **Output Stationary**
  - Minimize movement of partial sums

- **No Local Reuse**
  - Don’t use any local PE storage. Maximize global buffer size.

- **Row Stationary**

**Evaluation Setup**
- Same Total Area
- AlexNet
- 256 PEs
- Batch size = 16

**Normalized Energy Cost**
- 1× (Reference)
- 1×
- 2×
- 6×
- 200×
DS uses 1.4× – 2.5× lower energy than other dataflows

[Chen et al., ISCA 2016]
Dataflow Comparison: CONV Layers

RS optimizes for the best **overall** energy efficiency

[Chen et al., ISCA 2016]
Exploit Sparsity

Method 1. Skip memory access and computation

- No R/W
- No Switching

Method 2. Compress data to reduce storage and data movement

- 45% power reduction

[Chen et al., ISSCC 2016]
Eyeriss: Deep Neural Network Accelerator

Exploits data reuse for **100x** reduction in memory accesses from global buffer and **1400x** reduction in memory accesses from off-chip DRAM

Overall **>10x energy reduction** compared to a mobile GPU (Nvidia TK1)

[Joint work with Joel Emer]
Features: Energy vs. Accuracy

Accuracy (Average Precision)

- VGG16
- AlexNet
- HOG

Energy/ Pixel (nJ)

* Exponential
* Linear

Measured in 65nm*

- [Suleiman, VLSI 2016]
- [Chen, ISSCC 2016]

* Only feature extraction. Does not include data, classification energy, augmentation and ensemble, etc.

Measured in on VOC 2007 Dataset
1. DPM v5 [Girshick, 2012]
Energy-Efficient Processing of DNNs

A significant amount of algorithm and hardware research on energy-efficient processing of DNNs

Hardware Architectures for Deep Neural Networks

ISCA Tutorial
June 24, 2017

Website: [http://eyeriss.mit.edu/tutorial.html](http://eyeriss.mit.edu/tutorial.html)

V. Sze, Y.-H. Chen, T-J. Yang, J. Emer,

We identified various limitations to existing approaches
Design of Efficient DNN Algorithms

- Popular efficient DNN algorithm approaches

Network Pruning

before pruning

after pruning

pruning synapses

pruning neurons

Compact Network Architectures

Examples: SqueezeNet, MobileNet

... also reduced precision

- Focus on reducing number of MACs and weights
- Does it translate to energy savings?
Data Movement is Expensive

Energy of weight depends on memory hierarchy and dataflow

Normalized Energy Cost*

- DRAM
  - ALU
  - 0.5 – 1.0 kB
  - RF
  - 1× (Reference)
- NoC: 200 – 1000 PEs
  - PE
  - 1×
- 100 – 500 kB
  - Buffer
  - 2×
- 200 – 1000 PEs
  - PE
  - 6×
- 200 – 1000 PEs
  - PE
  - 200×

* measured from a commercial 65nm process
Energy-Evaluation Methodology

DNN Shape Configuration
(# of channels, # of filters, etc.)

DNN Weights and Input Data
[0.3, 0, -0.4, 0.7, 0, 0, 0.1, …]

Tool available at: https://energyestimation.mit.edu/

[Yang et al., CVPR 2017]
Key Observations

- Number of weights *alone* is not a good metric for energy
- All data types should be considered

Energy Consumption of GoogLeNet

- Output Feature Map: 43%
- Input Feature Map: 25%
- Weights: 22%
- Computation: 10%

[Yang et al., CVPR 2017]
Energy-Aware Pruning

Directly target energy and incorporate it into the optimization of DNNs to provide greater energy savings

- Sort layers based on energy and prune layers that consume most energy first
- EAP reduces AlexNet energy by 3.7x and outperforms the previous work that uses magnitude-based pruning by 1.7x

Pruned models available at http://eyeriss.mit.edu/energy.html

[Yang et al., CVPR 2017]
NetAdapt: Platform-Aware DNN Adaptation

- Automatically adapt DNN to a mobile platform to reach a target latency or energy budget
- Use empirical measurements to guide optimization (avoid modeling of tool chain or platform architecture)

Pretrained Network

<table>
<thead>
<tr>
<th>Metric</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>3.8</td>
</tr>
<tr>
<td>Energy</td>
<td>10.5</td>
</tr>
</tbody>
</table>

Adapted Network

Budget

<table>
<thead>
<tr>
<th>Metric</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>3.8</td>
</tr>
<tr>
<td>Energy</td>
<td>10.5</td>
</tr>
</tbody>
</table>

Empirical Measurements

<table>
<thead>
<tr>
<th>Metric</th>
<th>Proposal A</th>
<th>...</th>
<th>Proposal Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>15.6</td>
<td>...</td>
<td>14.3</td>
</tr>
<tr>
<td>Energy</td>
<td>41</td>
<td>...</td>
<td>46</td>
</tr>
</tbody>
</table>

Network Proposals

[Yang et al., ECCV 2018]

In collaboration with Google’s Mobile Vision Team
Problem Formulation

\[
\max_{\text{Net}} \text{Accuracy(Net)} \text{ subject to } \text{Resource}_j(\text{Net}) \leq \text{Budget}_j, j = 1, \ldots, m
\]

Break into a set of simpler problems and solve iteratively

\[
\max_{\text{Net}_i} \text{Acc}(\text{Net}_i) \text{ subject to } \text{Res}_j(\text{Net}_i) \leq \text{Res}_j(\text{Net}_{i-1}) - \Delta R_{i,j}, j = 1, \ldots, m
\]

*Acc: accuracy function, Res: resource evaluation function, \(\Delta R\): resource reduction, Bud: given budget

**Budget incrementally tightens** \(\text{Res}_j(\text{Net}_{i-1}) - \Delta R_{i,j}\)

- Supports multiple resource budgets at the same time
- Guarantees that the budgets will be satisfied because the resource consumption decreases monotonically
- Generates a family of networks (from each iteration) with different resource versus accuracy trade-offs
- Intuitive and can easily set one additional hyperparameter (\(\Delta R_{i,j}\))

[Yang et al., ECCV 2018]
Simplified Example of One Iteration

1. Input
   - Network from Previous Iteration
     - Layer 1: Latency: 100ms, Budget: 80ms
     - Layer 4: Latency: 100ms, Budget: 80ms

2. Meet Budget
   - Layer 1
     - 100ms
     - 90ms
     - 80ms
   - Layer 4
     - 100ms
     - 80ms
   - Selected Layers: 80ms

3. Maximize Accuracy
   - Acc: 60%
   - Acc: 40%
   - Selected Layers: 60%

4. Output
   - Network for Next Iteration
     - Latency: 80ms, Budget: 60ms

[Yang et al., ECCV 2018]
• NetAdapt boosts **the real inference speed** of MobileNet by up to 1.7x with higher accuracy

*Tested on the ImageNet dataset and a Google Pixel 1 CPU*

**Reference:**


[Yang et al., ECCV 2018]
FastDepth: Fast Monocular Depth Estimation

Depth estimation from a single RGB image desirable, due to the relatively low cost and size of monocular cameras.

Auto Encoder DNN Architecture (Dense Output)

[Joint work with Sertac Karaman]
FastDepth: Fast Monocular Depth Estimation

Apply *NetAdapt*, *compact network design*, and *depth wise decomposition* to decoder layer to enable depth estimation at **high frame rates on an embedded platform** while still maintaining accuracy.

Models available at [http://fastdepth.mit.edu](http://fastdepth.mit.edu)

[Wofk*, Ma* et al., ICRA 2019]
Many Efficient DNN Design Approaches

Network Pruning

Before pruning

After pruning

pruning synapses

pruning neurons

Compact Network Architectures

Reduce Precision

32-bit float

8-bit fixed

Binary

No guarantee that DNN algorithm designer will use a given approach.

Need flexible hardware!

[Chen et al., SysML 2018]
• Specialized DNN hardware often rely on certain properties of DNN in order to achieve high energy-efficiency

• **Example:** Reduce memory access by amortizing across MAC array
Limitation of Existing DNN Architectures

- **Example:** Reuse and array utilization depends on # of channels, feature map/batch size
  - Not efficient across all network architectures (e.g., compact DNNs)
Limitation of Existing DNN Architectures

• **Example:** Reuse and array utilization depends on # of channels, feature map/batch size
  – Not efficient across all network architectures (e.g., compact DNNs)

Example mapping for depth wise layer

- Number of input channels
- Number of filters (output channels)
- MAC array (spatial accumulation)
- Feature map or batch size
- Number of filters (output channels)
- MAC array (temporal accumulation)
Limitation of Existing DNN Architectures

- **Example:** Reuse and array utilization depends on # of channels, feature map/batch size
  - Not efficient across all network architectures (e.g., compact DNNs)
  - Less efficient as array scales up in size
  - Can be challenging to exploit sparsity
Need Flexible Dataflow

• Use flexible dataflow (Row Stationary) to exploit reuse in any dimension of DNN to increase energy efficiency and array utilization

Example: Depth-wise layer
Need Flexible NoC for Varying Reuse

- When reuse available, need **multicast** to exploit spatial data reuse for energy efficiency and high array utilization.
- When reuse not available, need **unicast** for high BW for weights for FC and weights & activations for high PE utilization.
- An all-to-all satisfies above but too expensive and not scalable.

[Chen et al., JETCAS 2019]
Hierarchical Mesh

[Chen et al., JETCAS 2019]
Eyeriss v2: Balancing Flexibility and Efficiency

Efficiently supports

- Wide range of filter shapes
  - Large and Compact
- Different Layers
  - CONV, FC, depth wise, etc.
- Wide range of sparsity
  - Dense and Sparse
- Scalable architecture

Over an order of magnitude faster and more energy efficient than Eyeriss v1

[Chen et al., JETCAS 2019]

[Joint work with Joel Emer]

<table>
<thead>
<tr>
<th></th>
<th>v1.5 &amp; MobileNet</th>
<th>v2 &amp; MobileNet</th>
<th>v2 &amp; sparse MobileNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONV1_DW</td>
<td>10.9</td>
<td>5.6</td>
<td>12.6</td>
</tr>
<tr>
<td>CONV2_DW</td>
<td>12.6</td>
<td>5.6</td>
<td>10.9</td>
</tr>
<tr>
<td>CONV3_DW</td>
<td>20.8</td>
<td>6.6</td>
<td>10.9</td>
</tr>
<tr>
<td>CONV4_DW</td>
<td>20.8</td>
<td>6.6</td>
<td>10.9</td>
</tr>
<tr>
<td>CONV1_1</td>
<td>20.8</td>
<td>6.6</td>
<td>10.9</td>
</tr>
<tr>
<td>CONV1_2</td>
<td>20.8</td>
<td>6.6</td>
<td>10.9</td>
</tr>
<tr>
<td>FC</td>
<td>20.8</td>
<td>6.6</td>
<td>10.9</td>
</tr>
<tr>
<td>Overall</td>
<td>20.8</td>
<td>6.6</td>
<td>10.9</td>
</tr>
</tbody>
</table>

Speed up over Eyeriss v1 scales with number of PEs

<table>
<thead>
<tr>
<th></th>
<th># of PEs</th>
<th>256</th>
<th>1024</th>
<th>16384</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td></td>
<td>17.9x</td>
<td>71.5x</td>
<td>1086.7x</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td></td>
<td>10.4x</td>
<td>37.8x</td>
<td>448.8x</td>
</tr>
<tr>
<td>MobileNet</td>
<td></td>
<td>15.7x</td>
<td>57.9x</td>
<td>873.0x</td>
</tr>
</tbody>
</table>
Energy-Efficient Autonomous Navigation

Navion Chip
Localization and Mapping at 2mW (full integration on-chip)

Enable energy-efficient navigation for Search and Rescue

http://navion.mit.edu

[Zhang et al., RSS 2017],
[Suleiman et al., VLSI 2018]

In collaboration with Sertac Karaman (AeroAstro)
Visual-Inertial Localization

Determines location/orientation of robot from images and IMU (also used by headset in Augmented Reality and Virtual Reality)

*Subset of SLAM algorithm (Simultaneous Localization And Mapping)

[Joint work with Sertac Karaman (AeroAstro)]
Frontend: Processing Sensors Data

Camera

Vision Frontend (VFE)

IMU Frontend (IFE)

IMU

[Zhang et al., RSS 2017]
**Frontend: Processing Sensors Data**

[Image of diagram showing data flow from Camera to Vision Frontend (VFE) and IMU Frontend (IFE), with feature tracks and Kalman Filters (KF1, KF2, KF3, KF4).]

[Image of feature tracks labeled with L1, L2, L3, L4, etc., with a camera image in the background showing the environment.]
Frontend: Processing Sensors Data

Camera

Vision Frontend (VFE)

Feature Tracks

Estimated State (Pose & Location)

IMU Frontend (IFE)

IMU

KF_1

KF_2

KF_3

KF_4

ΔR_{12}, ΔT_{12}

ΔR_{23}, ΔT_{23}

Preintegration

Preintegration

Gyro. & Acc. Measurements

[Zhang et al., RSS 2017]
Backend: Reduce Inconsistency

Update states ($x_i$) to minimize inconsistencies between measurements across time

[Zhang et al., RSS 2017]
**Backend: Factor Graph to Infer State of Drone**

Non-linear least squares factor graph optimization

\[
\min_x \sum_{(i,j) \in F} \| r_{IMU}(x, \Delta \tilde{R}_{ij}, \Delta \tilde{p}_{ij}, \Delta \tilde{v}_{ij}) \|^2 + \sum_{k \in \mathcal{L}} \sum_{i \in F_k} \| r_{CAM}(x, l_i, u_{ik}, u'^{IK}_{ik}) \|^2 + \| r_{PRIOR}(x) \|^2
\]

**IMU Factors**  
**Vision Factors**  
**Other Factors**

**Factor Graph**

- **Estimated State** (Pose & Location)
- **Feature Tracks**
- **Camera**
- **Vision Frontend (VFE)**
- **IMU Frontend (IFE)**
- **IMU**
- **Backend (BE)**

[Zhang et al., RSS 2017]

4000+ factors
Backend: Factor Graph to Infer State of Drone

Non-linear least squares factor graph optimization

\[
\min_x \sum_{(i,j) \in F} \| r_{IMU}(x, \Delta \tilde{R}_{ij}, \Delta \tilde{p}_{ij}, \Delta \tilde{v}_{ij}) \|^2 + \sum_{k \in \mathcal{L}} \sum_{i \in \mathcal{F}_k} \| r_{CAM}(x, l_k, u_{ik}^l, u_{ik}^r) \|^2 + \| r_{PRIOR}(x) \|^2
\]

Features:
- IMU Factors
- Vision Factors
- Other Factors

Factor Graph

Updated States \((x_i)\)

Sparse 3D Map

[Zhang et al., RSS 2017]
Navion is a fully integrated system:
No off-chip storage or processing

[Suleiman et al., VLSI 2018]
**Key Methods to Reduce Data Size**

*Navion*: Fully integrated system – no off-chip processing or storage

Use **compression** and exploit sparsity to reduce memory down to 854kB

[Suleiman et al., VLSI 2018]
Frame Buffer Memory

**Compress**

4x4 pixels example

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>6</th>
<th>6</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

- Line Buffer (1.4 kB)
- Find Min. & Max.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Original Image (352.5 kB)
- Compressed Image (79.4 kB)

- Frame Memory (78 kB)
- Compress
- Decompress

- Find Min. & Max.
- Thresh
- >>

- 5 bits
- Thresh
- 16 bits
- 5 bits
- Min.

- 0000010001111000
- 00111
- 00100

- 4.4x reduction

**Decompress**

- 4
- 4
- 4
- 4
- 4
- 7
- 4
- 4
- 4
- 7
- 4
- 4
- 4

- 16 bits
- 16 bits
- 16 bits

- 1.625-bit/pixel

[Suleiman et al., VLSI 2018]
Linear Solver and Hessian Memory

Linear Solver: \( H \Delta x = \varepsilon \), solve for \( \Delta x \)

1. \( H = L L^T \)  
   Calculate \( L \)

2. \( L u = \varepsilon \)  
   Solve for \( u \)

3. \( L^T \Delta x = u \)  
   Solve for \( \Delta x \)

**Linearize**

**Cholesky**

**Back-substitute**

Memory wrapper

Input

Sparse-based Control

Row

Column

Read/Write

Physical Address

Non-zero Hessian (134 kB)

Output

Zero

Non-zero entry

Masked Read/Write

Full Sym Sparse

Memory size

Processing time

Full

48.2 ms

Sparse

6.7 ms

7.2x

Back-substitution

2x

5.2x

Full

Sym

Sym + Sparse

Sparsity pattern in both \( H \) & \( L \)

(Non-zero: black)

[Suleiman et al., VLSI 2018]

MIT Research Laboratory of Electronics at MIT

MIT MicroSystems Laboratory

Research Laboratory of Electronics at MIT

MIT MicroSystems Laboratory
Factor Graph Memory

One Memory
(962 kB)

Two-stage Memory
(177 kB)

[Suleiman et al., VLSI 2018]
Navion Evaluation

- **Peak Performance @ Maximum Configuration**
  - VFE: 28 – 171 fps (71 fps average)
  - BE: 16 – 90 fps (19 fps average)
  - Average Power Consumption: 24mW
  - Trajectory Error: 0.28%

- **Real-Time Performance @ Optimized Configuration**
  - VF: 20 fps
  - BE: 5 fps
  - Average Power Consumption: 2mW
  - Trajectory Error: 0.27%

Over 250 configurable parameters to adapt to different sensors and environments

http://navion.mit.edu

Evaluated on EuRoC dataset

[Suleiman et al., VLSI 2018]
Navion System Demo

https://youtu.be/X5VZkPo_704
**Robot Exploration:** Decide where to go by computing Shannon Mutual Information

1. **Select candidate scan locations**
2. **Compute Shannon MI and choose best location**
3. **Move to location and scan**
4. **Update Occupancy Map**

Where to scan? **Mutual Information**

Exploration with a mini race car using motion capture for localization

**Occupancy map with planned path**

**MI surface**
Challenge is Data Delivery to All Cores

Process multiple beams in parallel

Data delivery from memory is limited
Specialized Memory Architecture

Break up map into separate memory banks and novel storage pattern to minimize read conflicts when processing different beams in parallel.

Compute the mutual information for an entire map of 20m x 20m at 0.1m resolution in under a second \(\rightarrow\) a 100x speed up versus CPU for 1/10\(^{th}\) of the power.

[Joint work with Sertac Karaman (AeroAstro)]

[Li et al., RSS 2019]
Low Power 3D Time of Flight Imaging

• Pulsed Time of Flight: Measure distance using round trip time of laser light for each image pixel
  – Illumination + Imager Power: 2.5 – 20 W for range from 1 - 8 m

• Use computer vision techniques and passive images to estimate changes in depth without turning on laser
  – CMOS Imaging Sensor Power: < 350 mW

Real-time Performance on Embedded Processor
VGA @ 30 fps on Cortex-A7 (< 0.5W active power)

[Noraky et al., ICIP 2017]
Results of Low Power Depth ToF Imaging

Mean Relative Error: 0.7%
Duty Cycle (on-time of laser): 11%

[Noraky et al., ICIP 2017]
Dementia affects 50 million people worldwide today (75 million in 10 years) [World Alzheimer’s Report]

---

**Mini-Mental State Examination (MMSE)**

Q1. What is the year? Season? Date?
Q2. Where are you now? State? Floor?
Q3. Could you count backward from 100 by sevens? (93, 86, ...)

---

• Neuropsychological assessments are **time consuming** and require a trained specialist
• Repeat **medical assessments** are **sparse**, mostly **qualitative**, and suffer from **high retest variability**

---

**In collaboration with Thomas Heldt (IMES)**
Use Eye Movements for Quantitative Evaluation

Eye movements can be used to quantitatively evaluate severity, progression or regression of neurodegenerative diseases.

High-speed camera

Substantial head support

IR illumination

Clinical measurements of saccade latency are done in constrained environments that rely on specialized, costly equipment.

Phantom v25-11

SR EYELINK 1000 PLUS

Measure Eye Movements Using Phone

Eye movements → Smartphone → Count

Eye movement feature

Develop algorithm to measure eye movement using a consumer-grade camera rather than high-cost research-grade camera.

Enable low-cost in-home longitudinal measurements.

---

[Saavedra Peña et al., EMBC 2018] [Lai et al., ICIP 2018]
Summary of Key Insights

• Data movement dominates energy consumption
  – Use dataflow that maximizes data reuse for all data types

• Design considerations for co-design of algorithm and hardware
  – Incorporate direct metrics into algorithm design for improved efficiency
  – Diverse workloads requires a flexible dataflow and NoC to exploit data reuse in any dimension and increase core utilization for speed and scalability

• Diverse compact representations to reduce data storage
  – Adapt compression based on key properties (dense or sparse; structured or unstructured) to maximize compression efficiency and minimize overhead

• Limited memory BW affects speed of highly parallel algorithms
  – Balance banking and arbitration cost to minimize energy and maximize core utilization

Today’s slides available at www.rle.mit.edu/eems

For Updates  🌐 Follow @eems_mit
Acknowledgements

Research conducted in the MIT Energy-Efficient Multimedia Systems Group would not be possible without the support of the following organizations:

For updates on our research  
Follow @eems_mit
References

- **Energy-Efficient Hardware for Deep Neural Networks**
  - Project website: [http://eyeriss.mit.edu](http://eyeriss.mit.edu)

- **Limitations of Existing Efficient DNN Approaches**
References

• Co-Design of Algorithms and Hardware for Deep Neural Networks

• Energy-Efficient Visual Inertial Localization
  – Project website: http://navion.mit.edu
References

• Fast Shannon Mutual Information for Robot Exploration

• Low Power Time of Flight Imaging

• Monitoring Neurodegenerative Disorders Using a Phone