What can in-memory computing deliver, and what are the barriers?

Naveen Verma (nverma@princeton.edu), L.-Y. Chen, H. Jia, M. Ozatay, Y. Tang, H. Valavi, B. Zhang, J. Zhang

March 20th, 2019
The memory wall

- Separating memory from compute fundamentally raises a communication cost.

More data → bigger array → larger comm. distance → more comm. energy.
So, we should **amortize** data movement

- Reuse accessed data for compute operations
- Specialized (memory-compute integrated) architectures

\[
\vec{c} = A \times \vec{b}
\]

\[
\begin{bmatrix}
  c_1 \\
  \vdots \\
  c_M
\end{bmatrix} =
\begin{bmatrix}
  a_{1,1} & \cdots & a_{1,N} \\
  \vdots & \ddots & \vdots \\
  a_{M,1} & \cdots & a_{M,N}
\end{bmatrix}
\begin{bmatrix}
  b_1 \\
  \vdots \\
  b_N
\end{bmatrix}
\]

---

**Processing Element (PE)**

- Reuse accessed data for compute operations
- Specialized (memory-compute integrated) architectures

\[
\vec{c} = A \times \vec{b}
\]

\[
\begin{bmatrix}
  c_1 \\
  \vdots \\
  c_M
\end{bmatrix} =
\begin{bmatrix}
  a_{1,1} & \cdots & a_{1,N} \\
  \vdots & \ddots & \vdots \\
  a_{M,1} & \cdots & a_{M,N}
\end{bmatrix}
\begin{bmatrix}
  b_1 \\
  \vdots \\
  b_N
\end{bmatrix}
\]
In-memory computing (IMC)

\[
\vec{c} = A\vec{b} \quad \Rightarrow \\
\begin{bmatrix}
    c_1 \\
    \vdots \\
    c_M
\end{bmatrix} = \\
\begin{bmatrix}
    a_{1,1} & a_{1,N} \\
    \vdots & \vdots \\
    a_{M,1} & a_{M,N}
\end{bmatrix} \begin{bmatrix}
    b_1 \\
    \vdots \\
    b_N
\end{bmatrix}
\]

- In SRAM mode, matrix A stored in bit cells row-by-row
- In IMC mode, many WLs driven simultaneously
  \(\rightarrow\) amortize comm. cost inside array
- Can apply to diff. mem. Technologies
  \(\rightarrow\) enhanced scalability
  \(\rightarrow\) embedded non-volatility

[J. Zhang, VLSI’16][J. Zhang, JSSC’17]
The basic tradeoffs

CONSIDER: Accessing $D$ bits of data associated with computation, from array with $\sqrt{D}$ columns $\times \sqrt{D}$ rows.

**Traditional**

Memory ($D^{1/2} \times D^{1/2}$ array)

$D^{1/2}$

Computation

**IMC**

Memory & Computation ($D^{1/2} \times D^{1/2}$ array)

<table>
<thead>
<tr>
<th>Metric</th>
<th>Traditional</th>
<th>In-memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>$1/D^{1/2}$</td>
<td>1</td>
</tr>
<tr>
<td>Latency</td>
<td>$D$</td>
<td>1</td>
</tr>
<tr>
<td>Energy</td>
<td>$D^{3/2}$</td>
<td>$\sim D$</td>
</tr>
<tr>
<td>SNR</td>
<td>1</td>
<td>$\sim 1/D^{1/2}$</td>
</tr>
</tbody>
</table>

- IMC benefits energy/delay at cost of SNR
- SNR-focused systems design is critical (circuits, architectures, algorithms)
Data Movement:

1. $b_{n,k}$'s broadcast minimum distance due to high-density bit cells
2. (Many) $a_{m,n}$'s stationary in high-density bit cells
3. High-dynamic-range analog $c_{m,k}$'s computed in distributed manner
IMC as a spatial architecture

Assume:
- 1k dimensionality
- 4-b multiplies
- 45nm CMOS

<table>
<thead>
<tr>
<th>Operation</th>
<th>Digital-PE Energy (fJ)</th>
<th>Bit-cell Energy (fJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>Multiplication</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>Accumulation</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>40</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>590</strong></td>
<td><strong>55</strong></td>
</tr>
</tbody>
</table>
Where does IMC stand today?

- Potential for $10\times$ higher efficiency & throughput
- Limited scale, robustness, configurability
Challenge 1: analog computation

- Use analog circuits to ‘fit’ compute in bit cells
  → SNR limited by analog-circuit non-idealities
  → Must be feasible/competitive @ 16/12/7nm

[J. Zhang, VLSI’16][J. Zhang, JSSC’17]
Algorithmic co-design(?)

- **Chip-specific weight tuning**

- **Chip-generalized weight tuning**

**E.g.: BNN Model (applied to CIFAR-10)**

\[
L = |y - \hat{y}(x, \theta, G)|^2
\]

\[
L = |y - \hat{y}(x, \theta)|^2
\]

Normalized MRAM cell standard dev.
Challenge 2: programmability

- Matrix-vector multiply is only 70-90% of operations
  → IMC must integrate in programmable, heterogeneous architectures

General Matrix Multiply
(~256×2300=590k elements)

Single/few-word operands
(traditional, near-mem. acceleration)

[B. Fleischer, VLSI’18]
Challenge 3: efficient application mappings

- IMC engines must be ‘virtualized’
  → IMC amortizes MVM costs, not weight loading. But…
  → Need new mapping algorithms (physical tradeoffs very diff. than digital engines)

Activation Accessing
- \( E_{\text{DRAM→IMC}/4-\text{bit}}: 40\text{pJ} \)
- Reuse: \( N \times l \times j \) (10-20 lyrs)
- \( E_{\text{MAC,4-b}}: 50\text{fJ} \)

Weight Accessing
- \( E_{\text{DRAM→IMC}/4-\text{bit}}: 40\text{pJ} \)
- Reuse: \( X \times Y \)
- \( E_{\text{MAC,4-b}}: 50\text{fJ} \)

Opportunity Related to Memory Bound and Compute Bound

\( I_{x,y,z} \) (\( X \times Y \times Z \) input activations)

\( W^n_{i,j,k} \) (\( N - l \times j \times k \) filters)

(output activations)
Path forward: charge-domain analog computing

1. Digital multiplication
2. Analog accumulation

~1.2fF metal capacitor (on top of bit cell)

[H. Valavi, VLSI'18]
### 2.4Mb, 64-tile IMC

![Image of Neuron Array](image)

#### Neuron Transfer Function

<table>
<thead>
<tr>
<th>Neuron Array (8x8 Neuron Tiles)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Neuron Array" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Technology</th>
<th>Moons, ISSCC’17</th>
<th>Bang, ISSCC’17</th>
<th>Ando, VLSI’17</th>
<th>Bankman, ISSCC’18</th>
<th>Valavi, VLSI’18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area (mm²)</td>
<td>1.87</td>
<td>7.1</td>
<td>12</td>
<td>6</td>
<td>17.6</td>
</tr>
<tr>
<td>Operating VDD</td>
<td>1</td>
<td>0.63-0.9</td>
<td>0.55-1</td>
<td>0.8/0.8 (0.6/0.5)</td>
<td>0.94/0.68/1.2</td>
</tr>
<tr>
<td>Bit precision</td>
<td>4-16b</td>
<td>6-32b</td>
<td>1b</td>
<td>1b</td>
<td>1b</td>
</tr>
<tr>
<td>on-chip Mem.</td>
<td>128kB</td>
<td>270kB</td>
<td>100kB</td>
<td>328kB</td>
<td>295kB</td>
</tr>
<tr>
<td>Throughput (GOPS)</td>
<td>400</td>
<td>108</td>
<td>1264</td>
<td>400 (60)</td>
<td>18,876</td>
</tr>
<tr>
<td>TOPS/W</td>
<td>10</td>
<td>0.384</td>
<td>6</td>
<td>532 (772)</td>
<td>866</td>
</tr>
</tbody>
</table>

- 10-layer CNN demos for MNIST/CIFAR-10/SVHN at energies of 0.8/3.55/3.55 μJ/image
- Equivalent performance to software implementation

[H. Valavi, VLSI’18]
Programmable IMC

- **Programmable IMC**

  - **CPU** (RISC-V)
  - **AXI Bus**
  - **DMA**
  - **Timers**
  - **GPIO**
  - **UART**
  - **Program Memory (128 kB)**
  - **Data Memory (128 kB)**
  - **Boot-loader**
  - **Timers**
  - **GPIO**
  - **UART**
  - **32 Program Memory (128 kB)**
  - **Boot-loader**
  - **Data Memory (128 kB)**
  - **Config. Regs.**
  - **Compute-In-Memory Unit (CIMU)**
    - 590 kb
    - 16 bank
  - **APB Bus**
  - **To E²PROM**
  - **To DRAM Controller**
  - **32b Reshaping Buffer**
  - **Sparsity/AND-logic Controller**
  - **Memory Read/Write I/F**
  - **Compute-In-Memory Array (CIMA)**
  - **Row Decoder/WL Drivers**
  - **Config.**
  - **f(ŷ = A ť)**

Bit-scalable mixed-signal compute

(E.g., $B_A=3, B_X=3$)

- SQNR different that standard integer compute

Mask bit for sparsity

$\begin{align*}
x_0[B_X-1:0]: & \quad 0-1-0 \\
M_0: & \quad 0 \\
\end{align*}$

$\begin{align*}
x_{N-1}[B_X-1:0]: & \quad 0-0-0 \\
M_{N-1}: & \quad 1 \\
\end{align*}$

$\begin{align*}
x_{2303}[B_X-1:0]: & \quad X-X-X \\
M_{2303}: & \quad 1 \\
\end{align*}$

Dynamic Range: $N+1$

(padding with one)

Dynamic Range: 256

Development board

To Host Processor

To Off-chip Mem. Controller

E²PROM for Bootloading

Custom Processor IC

GPIO Extension Board
Design flow

1. Deep-learning Training Libraries (Keras)

Standard Keras libs:
- `Dense(units, ...)`
- `Conv2D(filters, kernel_size, ...)`

Custom libs:
- `QuantizedDense(units, nb_input=4, nb_weight=4, chip_quant=True, ...)`
- `QuantizedConv2D(filters, kernel_size, nb_input=4, nb_weight=4, chip_quant=True, ...)`

2. Deep-learning Inference Libraries (Python, MATLAB, C)

High-level network build (Python):
- `chip_mode = True`
- `outputs = QuantizedConv2D(inputs, weights, biases, layer_params)`
- `outputs = BatchNormalization(inputs, layer_params)`

Function calls to chip (Python):
- `chip.load_config(num_tiles, nb_input=4, nb_weight=4)`
- `chip.load_weights(weights2load)`
- `chip.load_image(image2load)`
- `outputs = chip.image_filter()`

Embedded C:
- `chip_command = get_uart_word();`
- `chip_config();`
- `load_weights(); load_image();`
- `image_filter(chip_command);`
- `read_dotprod_result(image_filter_command);`
Demonstrations

Multi-bit Matrix-Vector Multiplication

![Graphs showing SQNR (dB) vs. B_A for different bit widths B_x and B_A.]

Neural Network Demonstrations

<table>
<thead>
<tr>
<th></th>
<th>Network A (4/4-b activations/weights)</th>
<th>Network B (1/1-b activations/weights)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy of chip</td>
<td>92.4% (vs. 92.7%)</td>
<td>89.3% (vs. 89.8%)</td>
</tr>
<tr>
<td>Energy/10-way Class.</td>
<td>105.2 μJ</td>
<td>5.31 μJ</td>
</tr>
<tr>
<td>Throughput</td>
<td>23 images/sec.</td>
<td>176 images/sec.</td>
</tr>
</tbody>
</table>

Neural Network Topology

- L1: 128 CONV3 – Batch norm
- L3: 256 CONV3 – Batch norm.
- L5: 256 CONV3 – Batch norm.
- L7: 1024 FC – Batch norm.
- L8: 10 FC – Batch norm.

Conclusions

Matrix-vector multiplies (MVMs) are a little different than other computations
→ high-dimensionality operands lead to data movement (memory accessing)

Bit cells make for dense, energy-efficient PE’s in spatial array
→ but require analog operation to fit compute, and impose SNR tradeoff

Must focus on SNR tradeoff to enable
scaling (technology/platform) and architectural integration

In-memory computing greatly affects the architectural tradeoffs,
requiring new strategies for mapping applications

Acknowledgements: funding provided by ADI, DARPA, NRO, SRC/STARnet