EHDSktch: A Generic Low Power Architecture for Sketching in Energy Harvesting Devices

PRIYANKA SINGLA
Indian Institute of Technology, Delhi, India

CHANDRAN GOODCHILD
University of Freiburg, Freiburg, Germany

PROF. SMRUTI R. SARANGI
Indian Institute of Technology, Delhi, India
Biography

Priyanka Singla
Ph.D. in Computer Science (2018 – present)
*Research Area:* Energy Harvesting Devices
School of Information Technology, IIT Delhi

Chandran Goodchild
M.Sc. in Embedded Systems Engg. (2017 – present)
University of Freiburg, Germany

Dr. Smruti R. Sarangi
Associate Professor in CSE and EE,
IIT Delhi
Introduction

Energy Harvesting Devices

- Extremely small amount of energy
- Memory: SRAM (~2-8KB), NVM (~64-128KB)

Convert ambient energy into electrical energy and store in a small storage unit (capacitor)
Introduction

Sensors
- Generate long data streams
- Need processing in real-time

Energy Harvesting Devices
- Extremely small amount of energy
- Memory: SRAM (~2-8KB), NVM (~64-128KB)

- Difficult to manage long streams on energy constrained devices

1. Body sensor
2. Temperature sensor
Introduction

Sensors
- Generate long data streams
- Need processing in real-time

Energy Harvesting Devices
- Extremely small amount of energy
- Memory: SRAM (~2-8KB), NVM (~64-128KB)

- Difficult to manage long streams on energy constrained devices

Solution: Use sketching algorithms

Sketching Algorithms
- Good accuracy, sub-linear space
- Energy and time efficient
Introduction

Background And Related Work
High-level Block Diagram

Sensor
Senses ambient, generates data-stream

MCU

NVM
Stores a sketch

Energy Harvesting Device

Application

1. Update(x)
2. Query(x)
3. Query(x)
4. Response
5. Response
CountMin (CM)

Maintains an **approximate count** of the number of occurrences of all the elements in a stream.

**Sketching Algorithm Example**

**Update(x)**

Increments the counter in value in each bucket

**Query(x)**

Computes the **minimum** of all the counters in corresponding buckets
Heavy-Hitters (HH)

Find elements whose frequency of occurrence is more than a threshold, $T$, where $T = \frac{N}{k+1}$, $k(>0)$ is specified by user.

The counter corresponding to each bit of the input element is incremented.
Counts the number of distinct elements in a data stream.

Flajolet-Martin (FM)

\[ C = \frac{2^{(k+1)}}{\varphi}, \text{ where } \kappa = \min\{i | \text{BITMAP}[i] = 0\} \]

\( C \): response, \( \varphi \approx 0.7 \) (correction factor)
Overview

**Sensors**
- Generate long data streams
- Need processing in real-time

**Energy Harvesting Devices**
- Extremely small amount of energy
- Memory: SRAM (~2-8KB), NVM (~64-128KB)

- Difficult to manage long streams on energy constrained devices

**Solution:** Use sketching algorithms

**Sketching Algorithms**
- Good accuracy, sub-linear space
- Energy and time efficient

- Application specific algorithms

**Solution:** Use a generic architecture

**EHDSktch**
- H/W implementation further provides energy and time efficiency
- Easily extensible; reusable components
Related Work

Key-value store. **Simplest.** Space complexity: $O(C)$, $C$ number of unique elements.

Reduce complexity to $O(1)$ by sampling. **Significant accuracy loss.**
Related Work

- Easily implementable on general-purpose processors.

- Sequential processing, pipeline overheads.

- Hardware implementations are an order of magnitude more efficient than general-purpose processors.
Related Work

Several FPGA-based implementations have been proposed for big data systems, aiming to achieve high throughput, using power-hungry LUTs and BRAMs.

LUT’s dynamic power consumption is around 500 times higher than an ASIC gate. EHD incompatible.

No emphasis on augmenting and extending the architecture. No memory and performance gains.
Generic Architecture
Background And Related Work
Introduction
Generic Architecture

Input element

Hash Engine

Hash 0
Hash 1
Hash D-1

Hash Stage

Index* Finder

Fetch Stage

Evaluation Stage
Extending the Architecture

Input element

Request Filter

Hash 0
Hash 1
Hash D-1
Hash Engine

Hash Stage

Request Resolver

Entry Finder*
Update*
Updated Value

Evaluation Stage

State*
Process State*
Subset Finder
Query Result

Fetch Stage

Mapper

Arbiter

B_0 Queue
B_1 Queue
B_{k-1} Queue

I/O Ports

B_0
B_1
B_{k-1}

SRAM

Extending the Architecture
Extending the Architecture

Idea and Significance

• Delay updates to the sketch
  + Handle update requests locally - expensive reads
• Performance gain and space reduction
Extending the Architecture

**Idea and Significance**
- Delay updates to the sketch
- Handle update requests locally - expensive reads
- Performance gain and space reduction

**Conceptual Description**
- Constant sized key-value store, evict entry with minimum priority
- Value field
  - CM: Counter and a *timestamp*
Extending the Architecture

Idea and Significance
- Delay updates to the sketch
  + Handle update requests locally - expensive reads
- Performance gain and space reduction

Conceptual Description
- Constant sized key-value store, evict entry with minimum priority
- Value field
  - CM: Counter and a timestamp
  - HH: counter, insertion and update timestamp

Priority for HH

(Update_timestamp – Insertion_timestamp)

- Frequently updated entry
- An older entry
Extending the Architecture

Implementation

- Two functionalities
  - Find a matching key-value pair

![Diagram showing a stream element and comparators for key-value pairs](image)
Extending the Architecture

Implementation

- Two functionalities
  - Find a matching key-value pair
  - Find an entry with minimum priority
- 16-bit, 5-stage, in-order processor, 16MHz
- Open source, cycle-accurate architectural simulator
- 2KB SRAM, 64KB FRAM, 4-port SRAM, 4 banks

- Follows the Zipfian distribution
- 16-bit data elements

- 32-bit counter values
- Depth (D): CM – 8, HH – 8
- Width (W): CM – 1024, HH – 64
- All signals are 16-bits wide
**H/W sketching compared to S/W sketching**

**Software Implementation:** Standard C++ STL based algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM</td>
<td>$8 \rightarrow 11\chi$</td>
</tr>
<tr>
<td>HH</td>
<td>$4 \rightarrow 6\chi$</td>
</tr>
<tr>
<td>FM</td>
<td>$&gt;10K\chi$</td>
</tr>
</tbody>
</table>

**EPU:** Energy per update, **EPQ:** Energy per query
H/W sketching compared to S/W sketching

CPU: Cycles per update
CPQ: Cycles per query

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM</td>
<td>$28 - 34\chi$</td>
</tr>
<tr>
<td>HH</td>
<td>$10 - 18\chi$</td>
</tr>
<tr>
<td>FM</td>
<td>$2K - 13K\chi$</td>
</tr>
</tbody>
</table>
## Performance of the request filter

<table>
<thead>
<tr>
<th>No. of Entries</th>
<th>Normalized Time</th>
<th></th>
<th>Normalized Energy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>512 Bytes</td>
<td>1024 Bytes</td>
<td>512 Bytes</td>
<td>1024 Bytes</td>
</tr>
<tr>
<td>CM</td>
<td>1.01</td>
<td>0.98</td>
<td>1.03</td>
<td>0.99</td>
</tr>
<tr>
<td>HH</td>
<td>1.05</td>
<td>0.98</td>
<td>1.07</td>
<td>1.00</td>
</tr>
</tbody>
</table>

### 3-6% performance gain, 50% SRAM area reduction

Performance increases by 5-8% with 16 additional entries. SRAM area reduced by 75%.

Normalization is done w.r.t. the straightforward hardware that has a 2 KB SRAM and no request filter.
### Performance of the request filter

<table>
<thead>
<tr>
<th>No. of Entries</th>
<th>Normalized Time 512 Bytes</th>
<th>Normalized Energy 512 Bytes</th>
<th>Normalized Time 1024 Bytes</th>
<th>Normalized Energy 1024 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>CM 1.01, HH 0.98</td>
<td>CM 1.03, HH 0.99</td>
<td>CM 1.03, HH 0.98</td>
<td>CM 1.06, HH 0.99</td>
</tr>
<tr>
<td>32</td>
<td>CM 1.05, HH 1.01</td>
<td>CM 1.07, HH 1.02</td>
<td>CM 1.08, HH 1.00</td>
<td>CM 1.10, HH 1.02</td>
</tr>
</tbody>
</table>

3-6% performance gain, 50% SRAM area reduction

Performance increases by 5-8% with 16 additional entries, SRAM area reduced by 75%

### Impact of the eviction policy on accuracy

<table>
<thead>
<tr>
<th># Entries</th>
<th>Relative Accuracy HH*</th>
<th>Relative Accuracy HH**</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 100</td>
<td>8 99.9</td>
</tr>
<tr>
<td></td>
<td>16 99.8</td>
<td>32 99.6</td>
</tr>
<tr>
<td></td>
<td>64 99.3</td>
<td>128 99.3</td>
</tr>
<tr>
<td></td>
<td>256 98.2</td>
<td></td>
</tr>
</tbody>
</table>

Eviction policy: *(update_timestamp – insertion timestamp), **LRU

Proposed eviction is 4-7% more accurate
Conclusion

Proposed to implement the algorithms in hardware, and created a **generic architecture**
- Instantiated three algorithms – CM, HH, and FM
- Extended the architecture

Experimentally evaluated our synthesized hardware, and compared against software implementations
- >4x reduction in energy
- >10x reduction in time

Evaluated the impact of the request filter
- 3-20% performance gain
- On-chip memory reduced by 50-75%
Contact: priyanka@cse.iitd.ac.in
http://www.cse.iitd.ac.in/~priyankas/