FPGA Acceleration for the Frequent Item Problem
Jens Teubner, René Müller, Gustavo Alonso
ETH Zurich, Systems Group
not (only) about FPGAs

not about a new solution
to the frequent item problem
Frequent Item Problem:

Given a stream \( S \) of items \( x_i \), which items occur most often in \( S \)?

Solution: [Mettwally et al. 2006]

```plaintext
1 foreach stream item \( x \in S \) do
2     find bin \( b_x \) with \( b_x.item = x \);
3     if such a bin was found then
4         \( b_x.count \leftarrow b_x.count + 1 \);
5 else
6     \( b_{min} \leftarrow \) bin with minimum count value;
7     \( b_{min}.count \leftarrow b_{min}.count + 1 \);
8     \( b_{min}.item \leftarrow x \);
```
(Intel T9550 @ 2.66 MHz; code by Cormode and Hadjieleftheriou, VLDB 2008)
Tricks on FPGAs:

content-addressable memory
  “hash table on steroids”

dual-ported memory
min-heap maintenance speed-up
→ data dependent, not scalable 😞
Lesson 1:

FPGAs are not a silver bullet.
Idea: Parallelize

1. **Broadcast** input item $x_i$ to all bins.
2. **Reduce** to determine $b_x$ and $b_{\min}$.
3. **Update** $b_x / b_{\min}$.
number of items monitored

throughput [million items / sec]

\( z = \infty \)

\( z = 1.5 \)

\( z = 0 \)

\( \rightarrow \text{still not scalable} \) 😞
What went wrong?

Lesson 2:
Avoid long-distance communication.
Can we keep processing local?

(avoid long-distance communication)
Pipeline-Style Processing:

1. **Compare** input item $x_1$ to content of bin $b_i$ (and increment count value if a match was found).
2. **Order** bins $b_i$ and $b_{i+1}$ according to count values.
3. **Move** $x_1$ forward in the array and **repeat**.

→ Drop $x_1$ into last bin if no match can be found.
$O(1) \rightarrow O(\#\text{bins})$ ?

But: Can be parallelized well.
Pipeline Parallelism:

\[
\begin{align*}
\mathcal{O}(\#\text{bins}) & \rightarrow \frac{1}{\#\text{bins}} \cdot \mathcal{O}(\#\text{bins}) \\
\end{align*}
\]
The graph compares the throughput (in million items/sec) of different systems as a function of the number of items monitored.

- **FPGA (pipeline parallel)**: A horizontal line at 80 million items/sec, indicating a consistent throughput regardless of the number of items monitored.
- **FPGA (data parallel)**: A green line that decreases as the number of items monitored increases, showing a drop in throughput.
- **Software**: A red line that also decreases with the number of items monitored, but at a slower rate compared to the FPGA (data parallel) system.

The data points for FPGA (data parallel) are marked with green circles, and the software data points are marked with red diamonds.
Lesson 3:

Pipelining → scalability, performance.
Lessons learned:

1. **FPGAs are not a silver bullet.**
   Straightforward s/w → h/w mapping will **not** do the job.

2. **Avoid long-distance communication.**
   Signal propagation delays will limit scalability.

3. **Pipelining → scalability, performance.**
   Keep communication and synchronization cheap.

**Frequent item solution:**

- three times faster than software, data independent.

This work was supported by the **Swiss National Science Foundation**.