Hardware for Event-Driven Architectures

David Schall,
Andreas Sandberg, Boris Grot
Event-Driven Architectures

Traditional Architecture:
• Implements functionality as a sequence of instructions and control commands
• Application prompts for input
• Program defines control flow

Good for small applications

Event-driven Architecture:
• Breaks functionality into many functional independent event handlers
• Event handlers react and communicate via events
• Events determine control flow

Decoupling improves flexibility and enables large applications

Event-Driven architectures are booming!
Event-Driven Application on a CPU

**Traditional Workload**

- Long and constantly running
- Large piece of code with fixed control flow

**Event-Driven Workload**

- Short execution times, run only on-demand
- Tiny code fractions in arbitrary order

**Traditional Applications**

- Caching can exploit temporal, spatial locality
- Prefetcher, branch predictor (BPU) are able to learn control flow

\[ \rightarrow \text{CPU performance is high } \smiley \]

**Event Handlers**

- Caching has no benefit: Too short, too infrequent execution
- Prefetchers and BPU cannot learn from the random order of invocations

\[ \rightarrow \text{CPU performance is low } \frown \]

**CPUs are not designed for event-driven applications!**
We study serverless functions as a use case of event-driven architectures for better understanding.

Interested in what we found?

Hardware for Event-Driven Architectures
David Schall\textsuperscript{1}, Andreas Sandberg\textsuperscript{2}, Boris Grot\textsuperscript{1}
\textsuperscript{1}University of Edinburgh, \textsuperscript{2}Arm Research

<table>
<thead>
<tr>
<th>Event-Driven Architectures</th>
<th>What is Event-Driven?</th>
<th>Why Event-Driven?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td></td>
<td>Disadvantages of Sequential Architectures:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>\cdot Gets quickly complex for large applications (spaghetti code)</td>
</tr>
</tbody>
</table>