ASSASIN: Architecture Support for Stream Computing to Accelerate Computational Storage

Chen Zou\textsuperscript{1}, Andrew A. Chien\textsuperscript{1,2}
\textsuperscript{1}Department of Computer Science, University of Chicago
\textsuperscript{2}Mathematics and Computer Science Division, Argonne National Laboratory
\{chenzou@, achien@cs\}.uchicago.edu
people.cs.uchicago.edu/~aachien/lssg/research/
Lightning: Systematically Architect Compute Inside SSD

How to integrate compute into SSD? What should the compute look like?

Feature A?
Feature B?
Feature C?
Datacenter Storage

C: CPU

D: Drive

Compute Nodes

Storage Nodes

Network
Disruptive Storage technology:

- Flash, NVM, …
- Continuous bandwidth scaling
- Data center network bottleneck
Datacenter Storage

Disruptive Storage technology:

- Flash, NVM, …
- Continuous bandwidth scaling
- Data center network bottleneck

Computational Storage:
Datacenter Storage

Disruptive Storage technology:

- Flash, NVM, …
- Continuous bandwidth scaling
- Data center network bottleneck

Computational Storage:

- Offload functions in/near storage to keep up with storage bandwidth
- Filesystem, Data Analytics, …
Datacenter Storage

Disruptive Storage technology:

- Flash, NVM, …
- Continuous bandwidth scaling
- Data center network bottleneck

Computational Storage:

- Offload functions in/near storage to keep up with storage bandwidth
- Filesystem, Data Analytics, …
- Largely reduce data moved to CPUs
Single Computational SSD
Single Computational SSD
Single Computational SSD

Diagram showing the components of a single computational SSD:
- Firmware Processor
- FlashCtrl
- Flash
- Controller
- DRAM
- SSD SOC
- Flash channels

Legend:
- Flash
- Controller
- DRAM
- Firmware Processor
- SSD SOC
Single Computational SSD
Single Computational SSD
Single Computational SSD

State-of-the-art generic computational SSD architecture

Captures design of [YourSQL, Biscuit, Summarizer, IceClave, BlockIF]
Problem: Memory Wall Inside Computational SSD

Data flows Flash => DRAM => ComputeElement=> DRAM

High DRAM traffics

Caches don’t help (because of workload)

+Management traffic for Firmware/FTL
Problem: Memory Wall Inside Computational SSD

Data flows Flash => DRAM => ComputeElement=> DRAM
High DRAM traffics
Caches don’t help (because of workload)
+Management traffic for Firmware/FTL

With increasing Flash BW: SSD DRAM bottleneck!

An 8-channel flash array of 12.8 GB/s would require at least 25.6 GB/s DRAM bandwidth.
> DDR4 channel
>> LPDDR4 (in SSD products)

How to eliminate bottleneck and deliver high performance?
Offloads are often Streaming Operators via systematic survey of common storage applications and offloads in industry and research. [see paper]

<table>
<thead>
<tr>
<th>File System</th>
<th>Streaming</th>
<th>Function State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptography</td>
<td>Data blocks / Code blocks</td>
<td>Keys &amp; GF table</td>
</tr>
<tr>
<td>(De)compress</td>
<td>Data and history</td>
<td>Dictionary indexes</td>
</tr>
<tr>
<td>Deduplicate</td>
<td>Data blocks</td>
<td>Block metadata</td>
</tr>
<tr>
<td>Erasure coding</td>
<td>Data blocks / Code blocks</td>
<td>Galois Field (GF) table</td>
</tr>
<tr>
<td>Replicate</td>
<td>Data &amp; Replicates</td>
<td>–</td>
</tr>
<tr>
<td>Database</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Filter</td>
<td>Tuples</td>
<td>Flags</td>
</tr>
<tr>
<td>Select</td>
<td>Tuples</td>
<td>–</td>
</tr>
<tr>
<td>Parse</td>
<td>Tuples</td>
<td>State machines</td>
</tr>
<tr>
<td>Statistics</td>
<td>Tuples</td>
<td>Accumulators</td>
</tr>
<tr>
<td>Data Science</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN Training</td>
<td>Training data</td>
<td>Model parameters</td>
</tr>
<tr>
<td>NN Inference</td>
<td>Inference input</td>
<td>Model parameters</td>
</tr>
<tr>
<td>Graph Analysis</td>
<td>Edge list / Vertex list</td>
<td>Statistics</td>
</tr>
</tbody>
</table>
Offloads are often Streaming Operators

via systematic survey of common storage applications and offloads in industry and research. [see paper]

<table>
<thead>
<tr>
<th>File System</th>
<th>Streaming</th>
<th>Function State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptography</td>
<td>Data blocks / Code blocks</td>
<td>Keys &amp; GF table</td>
</tr>
<tr>
<td>(De)compress</td>
<td>Data and history</td>
<td>Dictionary indexes</td>
</tr>
<tr>
<td>Deduplicate</td>
<td>Data blocks</td>
<td>Block metadata</td>
</tr>
<tr>
<td><strong>Erasure coding</strong></td>
<td><strong>Data blocks / Code blocks</strong></td>
<td><strong>Galois Field (GF) table</strong></td>
</tr>
<tr>
<td>Replicate</td>
<td>Data &amp; Replicates</td>
<td>–</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Database</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter</td>
<td>Tuples</td>
<td>Flags</td>
</tr>
<tr>
<td>Select</td>
<td>Tuples</td>
<td>–</td>
</tr>
<tr>
<td>Parse</td>
<td>Tuples</td>
<td>State machines</td>
</tr>
<tr>
<td>Statistics</td>
<td>Tuples</td>
<td>Accumulators</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Science</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NN Training</strong></td>
<td><strong>Training data</strong></td>
<td><strong>Model parameters</strong></td>
</tr>
<tr>
<td><strong>NN Inference</strong></td>
<td><strong>Inference input</strong></td>
<td><strong>Model parameters</strong></td>
</tr>
<tr>
<td>Graph Analysis</td>
<td>Edge list / Vertex list</td>
<td>Statistics</td>
</tr>
</tbody>
</table>
Offloads are often Streaming Operators via systematic survey of common storage applications and offloads in industry and research. [see paper]

<table>
<thead>
<tr>
<th>File System</th>
<th>Streaming</th>
<th>Function State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptography</td>
<td>Data blocks / Code blocks</td>
<td>Keys &amp; GF table</td>
</tr>
<tr>
<td>(De)compress</td>
<td>Data and history</td>
<td>Dictionary indexes</td>
</tr>
<tr>
<td>Deduplicate</td>
<td>Data blocks</td>
<td>Block metadata</td>
</tr>
<tr>
<td>Erasure coding</td>
<td>Data blocks / Code blocks</td>
<td>Galois Field (GF) table</td>
</tr>
<tr>
<td>Replicate</td>
<td>Data &amp; Replicates</td>
<td>–</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Database</th>
<th>Streaming</th>
<th>Function State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter</td>
<td>Tuples</td>
<td>Flags</td>
</tr>
<tr>
<td>Select</td>
<td>Tuples</td>
<td>–</td>
</tr>
<tr>
<td>Parse</td>
<td>Tuples</td>
<td>State machines</td>
</tr>
<tr>
<td>Statistics</td>
<td>Tuples</td>
<td>Accumulators</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Science</th>
<th>Streaming</th>
<th>Function State</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN Training</td>
<td>Training data</td>
<td>Model parameters</td>
</tr>
<tr>
<td>NN Inference</td>
<td>Inference input</td>
<td>Model parameters</td>
</tr>
<tr>
<td>Graph Analysis</td>
<td>Edge list / Vertex list</td>
<td>Statistics</td>
</tr>
</tbody>
</table>
Architecture Support for Stream computing to Accelerate computational Storage

ASSASIN computes offloads on flash data directly. Data is not first staged in DRAM. Eliminates DRAM Bottleneck.
Architecture Support for Stream computing to Accelerate computational Storage

ASSASIN computes offloads on flash data directly. Data is not first staged in DRAM. Eliminates DRAM Bottleneck.
ASSASIN computes offloads on flash data directly. Data is not first staged in DRAM. **Eliminates DRAM Bottleneck**

Flexible interconnect enables teaming ASSASIN cores for higher bandwidth and core utilization.
**Architecture Support for Stream computing to Accelerate computational Storage**

ASSASIN computes offloads on flash data directly. Data is not first staged in DRAM. **Eliminates DRAM Bottleneck**

Flexible interconnect enables teaming ASSASIN cores for higher bandwidth and core utilization.

ASSASIN core pulls/pushes data through streambuffer, computes on it directly

StreamISA (see paper) provides efficient stream pointers management.

Offload computation gets higher-performance data access.
Evaluation Methodology

Hybrid Simulation
- Gem5 RISCV model
  - Compute, DRAM, Stream Buffers
  - 8 RISCV cores
  - LPDDR5 8GB/s DRAM
- MQSim no interface Flash Model
  - Storage access
  - 8 channels of flashes
  - 1GB/s per channel

Workloads
- File system functions
- Database functions

Computational SSD Configurations

<table>
<thead>
<tr>
<th>Data Source</th>
<th>MemArch per Core. 32KB L1I omitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>DRAM (8GB/s)</td>
</tr>
<tr>
<td>L1D: 32KB, 8 way, 64B cache line</td>
<td></td>
</tr>
<tr>
<td>L2: 256KB, 16 way, 64B cache line</td>
<td></td>
</tr>
<tr>
<td>UDP [42]</td>
<td>DRAM (8GB/s) 256KB scratchpad</td>
</tr>
<tr>
<td>Prefetch</td>
<td>DRAM (8GB/s)</td>
</tr>
<tr>
<td>L1D: 32KB, 8 way, 64B cache line</td>
<td></td>
</tr>
<tr>
<td>L2: 256KB, 16 way, 64B cache line</td>
<td></td>
</tr>
<tr>
<td>DCPTPrefetcher [43] (best among Gem5 prefetchers)</td>
<td></td>
</tr>
<tr>
<td>AssasinSp</td>
<td>Scratchpad</td>
</tr>
<tr>
<td>64KB scratchpad</td>
<td></td>
</tr>
<tr>
<td>64KB I + 64KB O ping-pong scratchpads</td>
<td></td>
</tr>
<tr>
<td>AssasinSb</td>
<td>Streambuffer</td>
</tr>
<tr>
<td>64KB scratchpad</td>
<td></td>
</tr>
<tr>
<td>64KB I + 64KB O streambuffer (S=8, P=2)</td>
<td></td>
</tr>
</tbody>
</table>

Compute Acceleration

Memory Architecture Innovation
FileSystem Offloads: ASSASIN up to 2x Speedup

Prefetch aggravates the DRAM bottleneck. Minimal speedup.

ASSASIN benefits offload more when offloads are memory-intensive. Up to 2x speedup for eliminating the memory wall.

ASSASIN core features:
StreamISA and streambuffer brings 10% further speedup for AssasinSb over AssasinSp
Database offloads - combined operators (Parse, Select, Filter)

PSF Database offload is moderately compute-intensive

Compute acceleration (UDP) achieves 1.3x speedup over baseline (Geomean)

Memory-hierarchy innovation (ASSASIN) match and exceeds compute acceleration

- AssasinSp matches performance by avoiding DRAM bottleneck
- AssasinSb improves 18% further with streambuffer and StreamISA.
Achievable Clock Period

SB = StreamBuffer  SP = ScratchPad
Streambuffer precisely prefetches the head, so AssasinSb enjoys 11% shorter clock period.

Scratchpad limited by large MUXes. AssasinSp suffers 30% performance degradation.

SAED14nm technology
Achievable Clock Period

1.8x speedup, 3.0x power efficiency, 3.2x area efficiency

Shorter clock period further improves ASSASIN

SB=StreamBuffer SP=ScratchPad

Streambuffer precisely prefetches the head, so AssasinSb enjoys 11% shorter clock period.

Scratchpad limited by large MUXes. AssasinSp suffers 30% performance degradation.

SAED14nm technology
Related Work & Future Work

Landscape of Computational SSDs

Performance

ASIC
CIDR GList
GrafBoost Caribou
Lightstore DeepStore

FPGA
SmartSSD
Insider Access

Flexibility

Goal
Compute-oriented
programmable acc

ASSASIN

Summarizer
CPU
YourSQL BlockIF
IceClave Biscuit
Summary

Literature survey: Most computational storage offloads are dominated by streaming access.
The ASSASIN optimizes for streaming access.

- ASSASIN SSD architecture
  - integrates compute directly on flash data streams
  - eliminates the SSD DRAM bottleneck

- ASSASIN core architecture:
  - Streambuffers, StreamISA (integrated stream management)
  - Low-latency, energy, high bandwidth access

Benefits to offloads: 1.5-2.4x speedup. 3x power efficiency. 3.2x area efficiency.

See the paper for:
- ASSASIN programming model
- How ASSASIN enables independent FTL and thus generality
- ASSASIN’s performance scalability
- ASSASIN robust performance with data layout skew