PSACS: Highly-Parallel Shuffle Accelerator on Computational Storage

Chen Zou$^1$, Hui Zhang$^2$, Yang Seok Ki$^2$, Andrew A. Chien$^{1,3}$

{chenzou@, achien@cs.}uchicago.edu, {w.hzhang86, yangseok.ki}@samsung.com

$^1$Department of Computer Science, University of Chicago
$^2$Memory Solution Lab, Samsung Semiconductor Inc.
$^3$Mathematics and Computer Science, Argonne National Laboratory
Shuffle enables parallelism exploitation for OLAP
However, Shuffle may be a bottleneck

- **Fetch**
  - illustrative hash: \( h(\text{word}) = \text{word.length()} / 3 \)
- **Partition**
  - (pen, 1)
  - (pineapple, 1)
- **Group**
  - (pen, 1)
  - (apple, 1)
  - (pineapple, 1)
- **Shuffle**
  - 0: (pen, 1)
  - 1: (pen, 1)
  - 2: (pineapple, 1)

Most challenging!
Cache thrashing. Spills
Shuffle may take $\frac{1}{3}$ latency of an OLAP query

TPC-H with scaling factor 1000.

4 nodes running Spark 3.0.1 connected with 1Gbps ethernet.

CPU time collected with JVM-profiler[6].
PSACS: Highly-Parallel Shuffle Accelerator on Computational Storage

First shuffle acc on computational storage.
- Confines shuffle traffics in storage
- Liberates CPU and memory

PSACS microarchitecture exploits:
- Task, subtask and data-level parallelism
- Custom scratchpad for efficient gathering

PSACS achieves acceleration benefits:
- 5x kernel-level shuffle throughput
- 23% OLAP query speedup on average
PSACS partition approach: Hash

We opt for hash-based partition for its generality.

‘evenness’ rather than ‘collision resistance’ is the key here.

We apply a variant of fold hash that additionally zig-zag the input.
PSACS group approach

Bucketing (BypassMergeSortShuffleWriter, FPGAPart[12], Vitis[13]):

- Maintain a bucket for each destination partition to hold records going there.
- Hard to implement growable buckets without malloc-like dynamic memory allocation in hardware accelerators.
- Static bucket allocation presents linear scaling between capacity and #partitions.
- Parallelism exploitation leads to another shuffle problem.
PSACS group approach

Bucketing (BypassMergeSortShuffleWriter, FPGAPart[12], Vitis[13]):

- Maintain a bucket for each destination partition to hold records going there.
- Hard to implement growable buckets without malloc-like dynamic memory allocation in hardware accelerators.
- Static bucket allocation presents linear scaling between capacity and #partitions.
- Parallelism exploitation leads to another shuffle problem.

Sort (SortShuffleWriter):

- Sort the records by their assigned destination partition ids.
- Fixed resource requirement for different #partitions.
- Further optimization of only sorting record pointers rather than full records. Gather the records with sorted record pointers later using customized scratchpads.
Putting it together: PSACS uArch

- Partitioner (Hash)
- Sorter
- Reader
- Scratchpad
- FSM
- Gather
- Indexer
- Writer

Subtask parallelism
Random Access
Data-level Parallelism
Putting it together: PSACS uArch

**FSM**: Controls the shuffle process.
Putting it together: PSACS uArch

**FSM**: Controls the shuffle process.

**Reader**: Manages a customized scratchpad as a on-chip random access buffer for the table data.
Putting it together: PSACS uArch

**FSM**: Controls the shuffle process.

**Reader**: Manages a customized scratchpad as a on-chip random access buffer for the table data.

**Partitioner**: Accepts shuffle keys streamed from the reader. Map keys via hash to partition ID (PID).
Putting it together: PSACS uArch

**FSM**: Controls the shuffle process.
**Reader**: Manages a customized scratchpad as a on-chip random access buffer for the table data.
**Partitioner**: Accepts shuffle keys streamed from the reader. Map keys via hash to partition ID (PID).
**Sorter**: Sorts tuple (PID, RecPtr) from Partitioner by PID and stream the sorted tuples into Gather.
Putting it together: PSACS uArch

**FSM:** Controls the shuffle process.

**Reader:** Manages a customized scratchpad as a on-chip random access buffer for the table data.

**Partitioner:** Accepts shuffle keys streamed from the reader. Map keys via hash to partition ID (PID).

**Sorter:** Sorts tuple (PID, RecPtr) from Partitioner by PID and stream the sorted tuples into Gather.

**Gather:** Gathers records from Reader by RecPtrs in sorted tuple streams through random accessing.
Putting it together: PSACS uArch

**FSM**: Controls the shuffle process.

**Reader**: Manages a customized scratchpad as an on-chip random access buffer for the table data.

**Partitioner**: Accepts shuffle keys streamed from the reader. Map keys via hash to partition ID (PID).

**Sorter**: Sorts tuple (PID, RecPtr) from Partitioner by PID and stream the sorted tuples into Gather.

**Gather**: Gathers records from Reader by RecPtrs in sorted tuple streams through random accessing.

**Writer**: Stream records from Gather to DRAM, handling DRAM protocol
PSACS implementation

<table>
<thead>
<tr>
<th></th>
<th>LUT</th>
<th>FF</th>
<th>BRAM</th>
<th>URAM</th>
<th>DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avail</td>
<td>522720</td>
<td>1045440</td>
<td>984</td>
<td>128</td>
<td>1968</td>
</tr>
<tr>
<td>Used</td>
<td>61917</td>
<td>80885</td>
<td>433</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>Util</td>
<td>11.85%</td>
<td>7.74%</td>
<td>44.00%</td>
<td>50.00%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>
PSACS VS hand-optimized software

Single PSACS kernel VS single-thread software

5x kernel-level throughput
PSACS VS hand-optimized software

4-way-system pipeline with single PSACS VS 32-thread software (Data preparation and results writing back to storage included).

~20% higher system-level performance
CPU utilization reduction and overall query speedup

20x CPU utilization reduction

23% end-to-end query speedup on average
Related work

**Single-node partition acceleration:** Both academia[12] and industry[13]. Bucket-based grouping for single-node is used.


**Network fabric improvement:** SparkRDMA[10], SparkPMoF[11].

Summary

We proposed PSACS, the first shuffle accelerator addressing the shuffle bottlenecks of the OLAP systems.

- Employs rising computational storage paradigm
- Hardware acceleration through exploitation of multiple levels of parallelism
- Utilizes custom scratchpad for high-speed gathering

PSACS delivers 5x throughput at the kernel level and on average 23% end-to-end OLAP query speedup.

See our paper for more PSACS features

- Tiled shuffling tailoring for different levels inside the memarch
- Column-major output for higher compression ratio during fetch