**SpZIP: Architectural Support for Effective Data Compression in Irregular Applications**

Yifan Yang, Joel S. Emer, Daniel Sanchez

ISCA 2021

Session 12B (June 16, 2021 at 8 PM EDT)
Irregular applications are memory bound

- Irregular applications, such as graph analytics and sparse linear algebra, are an increasingly important workload domain

- Irregular applications are often *memory bound*

- Data compression is an attractive approach to accelerate irregular applications
Existing data compression solutions are not tailored for irregular applications.

Hardware compression units for **sequentially accessed long streams**
e.g., IBM z15 [ISCA’20]

This work is optimized for **indirect, data-dependent accesses to short streams**

Compressed memory hierarchies support **random accesses**
e.g., VSC [ISCA’04]
Understanding access patterns in irregular applications
Existing data compression solutions are not tailored for irregular applications

Hardware compression units for **sequentially accessed long streams**
- e.g., IBM z15 [ISCA’20]

This work is optimized for **indirect, data-dependent accesses to short streams**

Compressed memory hierarchies support **random accesses**
- e.g., VSC [ISCA’04]

- Limited compression gain on short streams
- Data decompression increases critical path latency
Compressing data structures in irregular applications is hard

- Challenge 1: Access and decompression are interleaved.
Compressing data structures in irregular applications is hard

- **Challenge 1:** Access and decompression are interleaved.

- **Insight 1:** Specialized hw to accelerate data access and decompression.
  - **Exploit** decoupled execution to hide memory access and decompression latencies.
Compressing data structures in irregular applications is hard

- **Challenge 2:** Need to support various access patterns and compression formats

- **Insight 2:** **Programmable** hardware
  - A pipeline consists of a set of **composable** operators expressing the traversal and decompression of data structures

- Dataflow Configuration Language (DCL)
Agenda

- Motivation
- SpZip Dataflow Configuration Language (DCL)
- SpZip Design
- Evaluation
A DCL program expresses the traversal, decompression and compression of data structures in irregular applications.

DCL program is an acyclic graph of composable operators.

Operators are connected by queues to exploit pipeline parallelism.
Traversing a sparse matrix in DCL

Compressed Sparse Row (CSR) format

{coordinate, value} pairs

for \( i \) in range(numRows):
  for \{col, val\} in rows[offsers[i]:offsers[i+1]]:
    visit({col, val})
Traversing a sparse matrix in DCL

```python
for i in range(numRows):
    for {col, val} in rows[offsets[i]:offsets[i+1]]:
        visit({col, val})
```

CSR format

```
<table>
<thead>
<tr>
<th>offsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>row0 row1 row2 row3</td>
</tr>
</tbody>
</table>
```

InputQ → Range → OffsetsQ → Range → RowsQ
Data decompression support in DCL

CSR with individually compressed rows

```
for i in range(numRows):
    for {col, val} in decompress(offsets[i], offsets[i+1]):
        visit({col, val})
```

InputQ → Range → OffsetsQ → Compressed RowsQ → Decompress
Using DCL in PageRank traversing multiple data structures

```python
def PageRankIter(Graph g, Array scores, Array contrs):
    for src in range(g.numVertices):
        for dst in decompress(g.neighs[g.offsets[src]:g.offsets[src+1]]):
            scores[dst] += contrs[src]
```

Graph adjacency matrix

Source Vertex Data

Compressed Adjacency Matrix

Prefetch Destination Vertex Data

InputQ

OffsetsQ

Compressed NeighsQ

Range

Decompress

NeighsQ

ContrbsQ

Core
Agenda

- Motivation
- SpZip Dataflow Configuration Language (DCL)
- SpZip Design
- Evaluation
SpZip Overview

- SpZip augments each CPU core with a programmable fetcher and compressor.
- The fetcher accelerates data structure traversal and decompression.
- The compressor compresses newly generated data before storing it off-chip.
- Fetcher and compressor issue conventional cache line requests.
SpZip exploits decoupled execution

- The fetcher and compressor communicate with core through queues to exploit decoupled execution.

- The fetcher runs ahead of the core to traverse and decompress data, hiding memory access and decompression latencies.
SpZip fetcher microarchitecture

- Access Unit and Decompression Unit implement DCL operators
- Scratchpad holds queues between operators
- Queues between operators allow pipeline parallelism
SpZip fetcher is programmable

- Scratchpad is configurable to support variable numbers and sizes of queues
- DCL operators are time-multiplexed on the same physical unit
- Scheduler holds operator contexts and chooses which operator to fire each cycle
SpZip compressor overview

- Compression uses a different set of DCL operators

- Similar decoupled and programmable design as the fetcher

- See paper for more details
Agenda

- Motivation
- SpZip Dataflow Configuration Language (DCL)
- SpZip Design
- Evaluation
Evaluation Methodology

- Event-driven simulation using ZSim
- SpZip system
  - 16 Haswell-like OOO cores
  - 32 MB L3 cache
  - 4 memory controllers (51.2GB/s)
  - SpZip adds 0.2% area overhead of per-core fetcher and compressor

- Irregular applications
  - PageRank, PageRank Delta, Connected Components, Radii Estimation, BFS, Degree Counting, SPMV

- Large real world inputs
  - Up to 100 million vertices
  - Up to 1 billion edges
SpZip improves performance and reduces traffic
See paper for

- DCL support for compressing data structures
- Programmable compressor design
- Additional evaluation results
  - Impact of preprocessing
  - Benefits over compressed memory hierarchies
  - Impact of decoupled fetching vs data compression
Conclusions

- Irregular applications have indirect, data-dependent memory access patterns that make compression challenging.

- SpZip makes data compression practical for irregular applications.
  - Decoupled execution hides memory access and decompression latencies.
  - DCL and programmable design support wide range of data structures and compression formats.

- SpZip achieves significant speedups and memory traffic reductions on irregular applications.
THANKS FOR YOUR ATTENTION!

ISCA 2021
Session 12B (June 16, 2021 at 8 PM EDT)