Accelerating SpMV on FPGAs by Lossless Nonzero Compression

Paul Grigoras

May 4, 2015
Motivation

Accelerate memory bound kernels – e.g. iterative SpMV

```c
A = ...  // read matrix once
for ( ... )  // many iterations
    q = ...
    p = A * q  // Sparse Matrix–Vector Multiplication
...
```

- common in scientific computing (e.g. Krylov iterative solvers)
Motivation

Accelerate memory bound kernels – e.g. iterative SpMV

```
A = ... // read matrix once
for ( ... ) // many iterations
  q = ... 
  p = A * q // Sparse Matrix-Vector Multiplication
...
```

- common in scientific computing (e.g. Krylov iterative solvers)

Memory bound – must increase effective DRAM bandwidth
Motivation

Accelerate memory bound kernels – e.g. iterative SpMV

A = ... // read matrix once
for ( ... ) // many iterations
    q = ...
    p = A * q // Sparse Matrix–Vector Multiplication
...

▶ common in scientific computing (e.g. Krylov iterative solvers)

Memory bound – must increase effective DRAM bandwidth

▶ use compression/decompression
Motivation

Accelerate memory bound kernels – e.g. iterative SpMV

\[
\begin{align*}
A = \ldots & \quad // \text{read matrix once} \\
\textbf{for} \ ( \ldots ) & \quad // \text{many iterations} \\
q = \ldots & \\
p = A \times q & \quad // \text{Sparse Matrix–Vector Multiplication} \\
\ldots
\end{align*}
\]

- common in scientific computing (e.g. Krylov iterative solvers)

Memory bound – must increase effective DRAM bandwidth

- use compression/decompression
- to improve overall performance (on FPGAs) must
  1. use only spare resources (BRAMs)
  2. decompress at processing pipeline rate
Approach

Overview

1. compress sparse matrix values on CPU
   - one-off operation – matrix reused for many iterations
   - use the Bounded CSRVI Format
2. store to FPGA accelerator DRAM
3. decompress at runtime
Approach

Overview

1. compress sparse matrix values on CPU
   ▶ one-off operation – matrix reused for many iterations
   ▶ use the Bounded CSRVI Format
2. store to FPGA accelerator DRAM
3. decompress at runtime

Bounded CSRVI

▶ Encode only $k$ most frequent values
  ▶ can control resource usage
▶ Store decoding table in BRAM
  ▶ use it at runtime for decompression
  ▶ decoding operation is BRAM look-up
  ▶ produces one value per clock cycle
Example

Figure 1: rajat30 – circuit simulation, 640K x 640K, 6M nnzs

- CSR
  - $k = 0$
- BCSRVI
  - $k \in [0, 2^{32}]$
- CSRVI
  - $k = 2^{32}$

Best

Insufficient Spare Resources

Insufficient Compression

Compression Ratio

Spare Resources: 8 BRAMs

Unique Values (BRAMS):

- 4(1)
- 8(1)
- 16(1)
- 32(1)
- 64(1)
- 128(1)
- 256(1)
- 512(1)
- 1024(2)
- 2048(4)
- 4096(8)
- 8192(16)
- 16384(32)
- 32768(64)
- 65536(128)
- 131072(256)
- 262144(512)
- 524288(1024)
Results

- **Test Systems**
  - Maxeler Maia (Stratix V) and Vectis (Virtex 6)

- **Benchmark**
  - 86 UoF matrices, \( \text{Order} \in [767..4M] \), \( \text{Nonzeros} \in [6027..77M] \)

- Low *resource usage* for up to 12 bits
  - enables use with multi-pipe SpMV kernels;
  - decoding tables R/O – use dual read-port to reduce BRAM

- With \( k = 2^{12} \) (4096 values, 12 bits)
  - Support 21 more matrices than CSRVI
  - Compression ratio over CSR: 1.16 – 1.79
  - Resource usage over CSRVI: 2.65 – 1139X less BRAMs
Conclusion

- Simple approach works well on some matrices
  - Can use spare resources for increased bandwidth
  - Supports more matrices than CSRVI
  - Often reduced storage over CSR (application specific)
  - High throughput (one value per cycle)
Conclusion

- Simple approach works well on some matrices
  - Can use spare resources for increased bandwidth
  - Supports more matrices than CSRVI
  - Often reduced storage over CSR (application specific)
  - High throughput (one value per cycle)

Future work

- Apply to other iterative streaming applications?
Conclusion

- Simple approach works well on some matrices
  - Can use spare resources for increased bandwidth
  - Supports more matrices than CSRVI
  - Often reduced storage over CSR (application specific)
  - High throughput (one value per cycle)

Future work

- Apply to other iterative streaming applications?

*Have a sparse matrix? Find me @poster session!*