Indirection Stream Semantic Register Architecture for Efficient Sparse-Dense Linear Algebra

Paul Scheffler  IIS, ETH Zurich  <paulsc@iis.ee.ethz.ch>
Florian Zaruba  IIS, ETH Zurich  <zarubaf@iis.ee.ethz.ch>
Fabian Schuiki  IIS, ETH Zurich  <fschuiki@iis.ee.ethz.ch>
Torsten Hoefler  SPCL, ETH Zurich  <htor@inf.ethz.ch>
Luca Benini  IIS, ETH Zurich  <lbenini@iis.ee.ethz.ch>
Sparse tensors are *ubiquitous*
- ML, CS, physics, economics...
- Various formats and algorithms

Common: *CSR Matrix × Vector (CsrMV)*
- Large *control-to-compute* ratio
- Low data reuse
- *Memory indirection*

➢ SoA: low functional utilization
- Xeon Phi KNL (CVR format\(^1\)): 0.7% FP64
- AGX Xavier (CuSPARSE\(^2\)): 2.1% FP32

\[
\begin{pmatrix}
0 & 0 & 7 & 9 & 0 \\
3 & 0 & 4 & 1 & 2 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 7 & 0 & 9 \\
\end{pmatrix}
\]

\[
\vec{y} = \begin{pmatrix} 1 \\ 7 \\ 3 \\ 2 \\ 9 \end{pmatrix}
\]

A_vals[] = \{7, 9, 3, 4, 1, 2, 7\}
A_idcs[] = \{2, 3, 0, 2, 3, 4, 2\}
A_ptrs[] = \{0, 2, 3, 3, 4\}

\[
\text{for } i \text{ in } 0 \text{ to } \text{A_nrows:}
\]
\[
\text{for } j \text{ in } \text{A_ptrs}[i] \text{ to } \text{A_ptrs}[i+1]:
\]
\[
y[i] += \text{A_vals}[j] * x[\text{A_idcs}[j]]
\]
Stream Semantic Registers

- Lightweight RISC-V ISA extension
  - Dense LA: up to 100% FPU utilization
- Map registers to *memory streams*
  - Accesses become loads/stores
  - Fixed-stride address generators
  - Orthogonal to HW loops
- Sparse-dense LA: limited by *indirection*
  - With SSR: 11 → 14% max FPU utilization
  - Indirect inside SSR → issue only fmadds

![Diagram showing memory access and SSR](image)

---

3 February 2021

Paul Scheffler, Florian Zaruba, Fabian Schuiki, Torsten Hoeffer, Luca Benini, ETH Zurich
Our Contributions

1. *Indirection SSR (ISSR)* architecture
2. Programming model and sparse-dense product kernels
3. Significant performance and energy benefits
4. Comparison to SoA CPU, GPU, HW accelerator approaches
Address Generator Extension

- Idea: Index lookup inside address generator
  - Add read-only memory port
  - Stay backward-compatible

- Indirection:
  1. Fetch 64b index words
  2. Serialize to 16b or 32b indices
  3. Shift: 64b offset (FP64) + dense axis stride
  4. Add base address
Integration for Evaluation

- Evaluated in *Snitch Cluster*[^4]
  - 8 cores: DP-FPU with 2 SSRs, frep
  - 1 core: 512-bit DMA[^5]
  - 256 KiB banked scratchpad

- 1 ISSR + 1 SSR per core
  - +4.4 kGE in address generator
  - +24% SSR, +0.8% cluster area

- Combine ISSR data, index ports
  - Prefer 80% max. utilization over 1.5× interconnect area


Sparse-Dense Product Kernels

- **Dot product (SpVV)**: minimal ISSR showcase
  1. **Setup**: set up SSRs, accumulators
  2. **frep loop**: continuous *fmadd stream*
  3. **Teardown**: reduce, write back

- **frep** *decouples* core and FPU
  ➢ Core free to set up next job

- **CsrMV**: accelerated through *row unrolling*, streaming entire matrix in *one job*

- **CsrMM**: reuse CsrMV with $2^K$ *stride*
Results: Single-Core Performance

• Compare 16b / 32b index ISSR kernels to
  • BASE: optimized RISC-V baseline
  • SSR: regular SSRs only

• All data in cluster memory

• SpVV: up to 80% / 67% FPU utilization
  • Slower ISSR convergence due to high peak

• CsrMV: approach 7.2x / 6.0x speedups
  • Real-world matrices\(^6\): 2-3.2k cols, 1.3-680k nonzeros
  • Optimizations: decent speedup even for low \(\overline{n}_{nz}\)

Results: Cluster CsrMV

- Eight-core CsrMV with same kernels
  - Same real-world matrices
  - Distribute rows, stationary vector
  - Double-buffered DMA matrix transfer
- Despite parallelization: up to $5.8\times$ faster
  - $1.9\times$ for $\bar{n}_{nz} = 1$, over $5\times$ for $\bar{n}_{nz} \geq 50$
  - Reduced by: row imbalance, work sharing, bank conflicts, I$S$ misses, vector transfer
- Up to $2.7\times$ more energy-efficient
  - $142 \rightarrow 53 \text{ pJ}$ per sparse-dense MAC
Related Work

• **CPUs**: ISSR similar to *scatter-gather*
  • SIMD-based: Xeon Phi KNL\textsuperscript{7}, Arm SVE\textsuperscript{8}
  • CVR SpMV on KNL\textsuperscript{1}: 0.7% DP-FP (70\times)

• **GPUs**: sparsity tackled in SW and HW
  • CuSPARSE\textsuperscript{2} CsrMV (1080 Ti): 17% FP64 (2.8\times)
  • A100\textsuperscript{9}: only *structured* sparsity (2 in 4)

• **HW Accelerators**: hard to compare
  • Not capable of general-purpose compute
  • DL: various precisions $\rightarrow$ goal is *accuracy*
  • Can adopt some *high-level methods* in SW

---

---

---
Conclusion

• Extended SSR for indirection $\rightarrow$ ISSR

• Efficient sparse-dense product kernels

• Evaluated ISSR in Snitch Cluster
  • +4.4 kGE or +0.8% in eight-core cluster
  • Single core: up to 7.2× faster, 80% FPU util.
  • Cluster: up to 5.8× faster, 2.7× less energy

• Compared ISSR to SoA approaches
  • 2.8× peak FP64 util. of CuSPARSE$^2$ (1080 Ti)
  • More flexible than GPU, accelerator HW
Questions?


