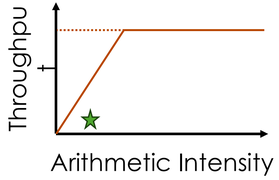
A Reconfigurable, Distributed Memory Accelerator for Sparse Applications

Courtney Golden (MIT CSAIL), Joel S. Emer (MIT CSAIL / NVIDIA), and Daniel Sanchez (MIT CSAIL)
{cgolden, emer, sanchez}@csail.mit.edu

Motivation & Background

Characteristics of Scientific and Graph Workloads

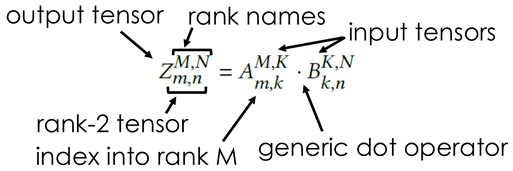


1. Iterative
2. Highly sparse
3. Static and dynamic sparsity
4. Low intra-iteration arithmetic intensity

Shortcomings of Prior Work

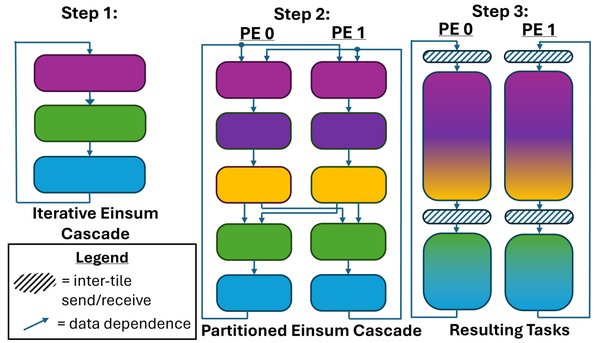
Prior all-SRAM architectures overcome the memory bottleneck but suffer from ***low programmability*** or ***low compute performance.***

Einsum Notation



Programming Model

We convert cascades of Einsums into tasks by encapsulating data partitioning across tiles and then grouping operations that occur on the same tile.



Ex: SpMV

$$Z^M_m = A^{M,N}_{m,n} \cdot B^N_n$$

strictly partitioned
shard partitioned
duplicate partitioned

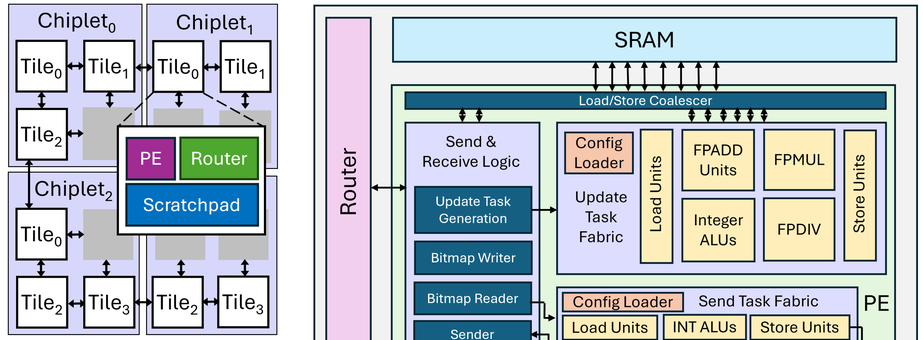
$$LB^{P,N}_{p,n} = T1^{P,N}_{q,n} \cdot B^{P,N}_{p,n}$$
$$LZ^{P,M}_{p,m} = A^{P,M,N}_{p,m,n} \cdot LB^{P,N}_{p,n}$$
$$Z^{P,M} = T2^{P,M}_{p,m} \cdot LZ^{P,M}_{q,m}$$

4 Task Types:

1. SendColumnVal
2. ScaleColumn
3. SendPartialSum
4. UpdateRowSum

Hardware Architecture

Our tiles are designed to efficiently execute short, dataflow tasks using reconfigurable logic. We exploit problem structure and data partitioning to support a few key sparse primitives needed for scientific and graph workloads.



Task Structure:

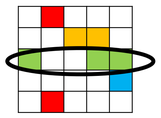
receive input from network and/or read local data from storage → compute on data → write data to local storage and/or send data over network

Data Partitioning

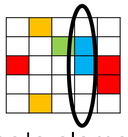
Partitioning data across distributed memories has a large performance impact because it directly determines network traffic and load balance.

Objectives for data partitioning:
1. Load balance work among tiles
2. Minimize inter-tile communication

We partition data at a *single-element granularity*, allowing any nonzero of any tensor to be placed at any tile.



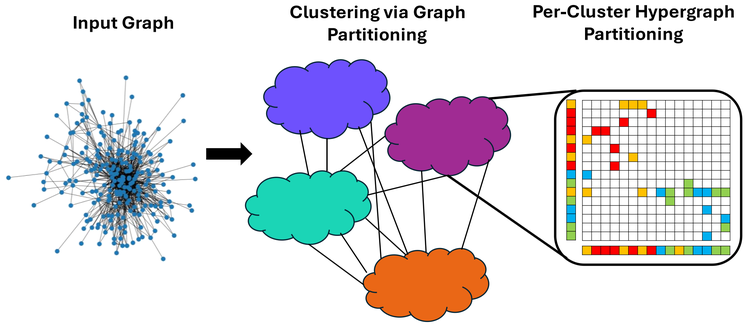
co-locate elements in the same **row**



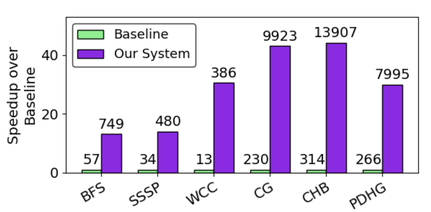
co-locate elements in the same **column**

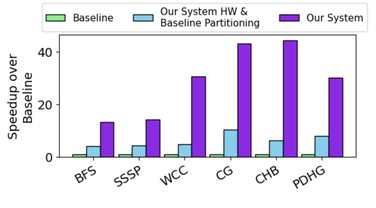
spread elements **equally** among tiles

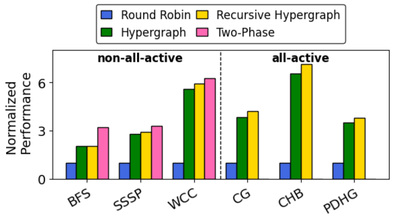
1. Identify data that are likely to be used in the same iteration and place into clusters.
2. Independently partition each cluster over tiles, minimizing intra-cluster communication.

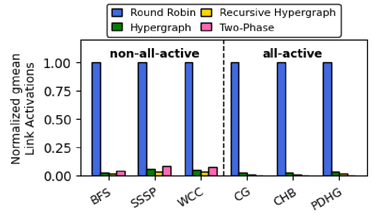


Results









Speedup (gmean): 26.2x

Area per chiplet: 450mm^2

Power per chiplet (gmean): 24.3 W