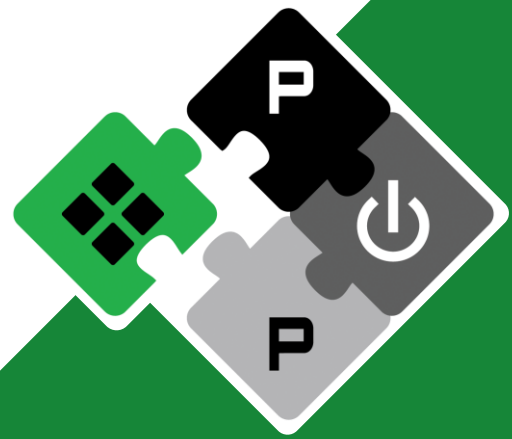
ETH zürich

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Exploring Outer Product and Gustavson Dataflows for SpMM acceleration

Integrated Systems Laboratory (ETH Zürich)

Marco Ferroni mferroni@ethz.ch



pulp-platform.org

@pulp_platform

company/pulp-platform

youtube.com/pulp_platform

PULP Platform

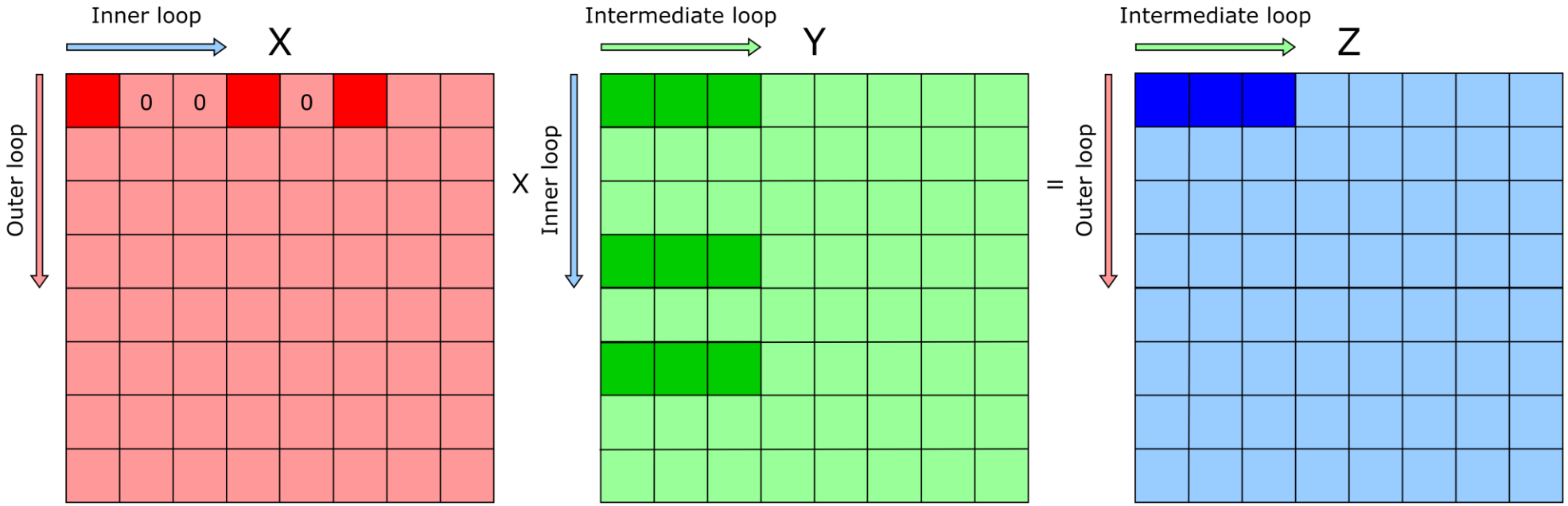
Open Source Hardware, the way it should be!

Motivation

- **GEMM is a fundamental operation for many workloads, especially ML**
  - Activations (ReLU, etc.) and pruning introduce many zeros.
  - Event-based sensors produce intrinsically sparse data.
- **Ignoring sparsity wastes cycles, bandwidth, and energy.**
  - Processing zeros ⇒ unnecessary latency and power
  - Making the core sparsity-aware is costly.
- **Idea: moving the "intelligence" inside the streamer**
  - No modifications to the engine which remains agnostic to the sparsity
  - Scalable
  - Less latency and memory traffic
- **Our study case:**
  - Matrix X is sparse and encoded in bitmap format, while matrix Y is dense

The idea

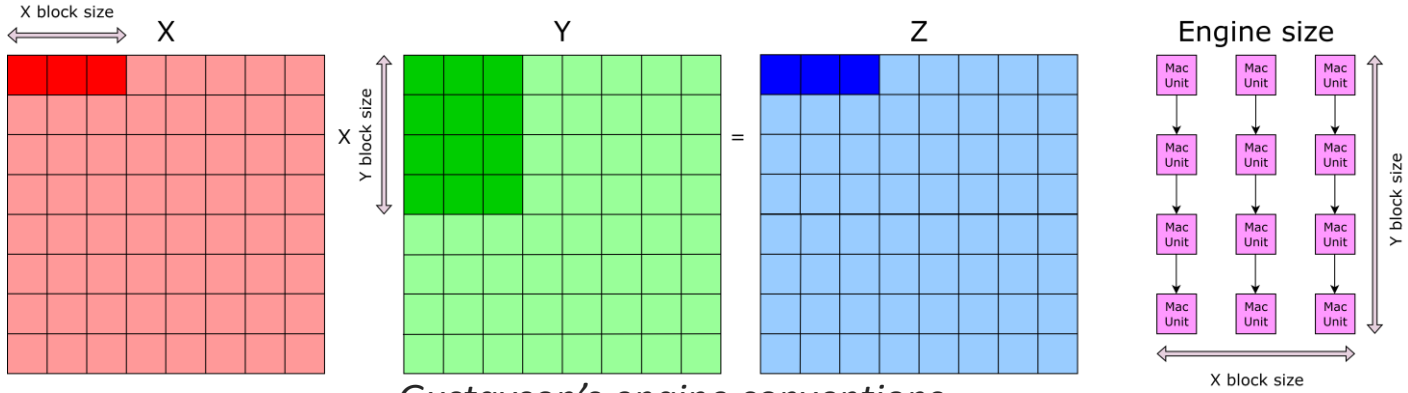
- **Avoid multiplications by zero**
  - If the element in the row of X is a zero, I can skip the correspondent row of Y
  - This way I avoid unnecessary loads and multiplications that consume BW, energy and cycles



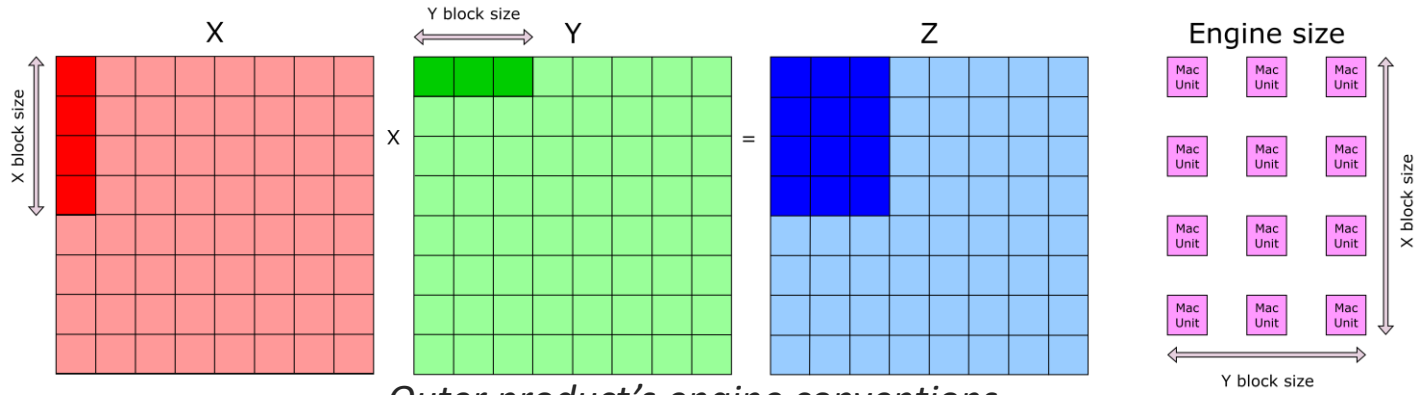
Example with Gustavson Z

Conventions used for the engine size

- **The engine size will depend on the chosen portions of matrices X and Y**
  - In Gustavson the same X block will be shared with every row of the Y block
  - In outer product the elements of the two blocks are combined to obtain a matrix chunk of partial results



Gustavson's engine conventions

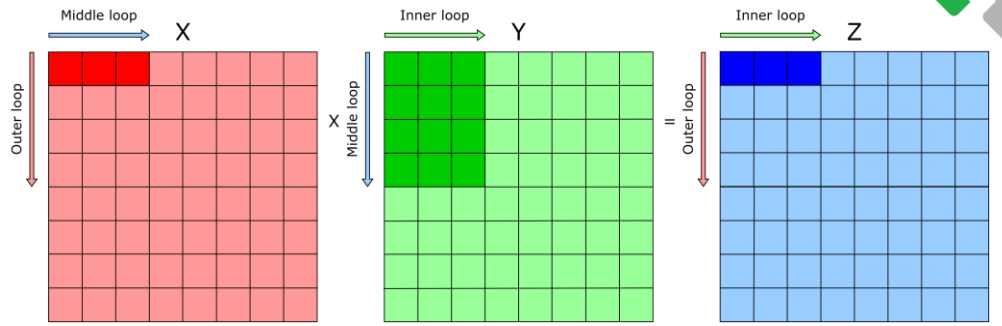


Outer product's engine conventions

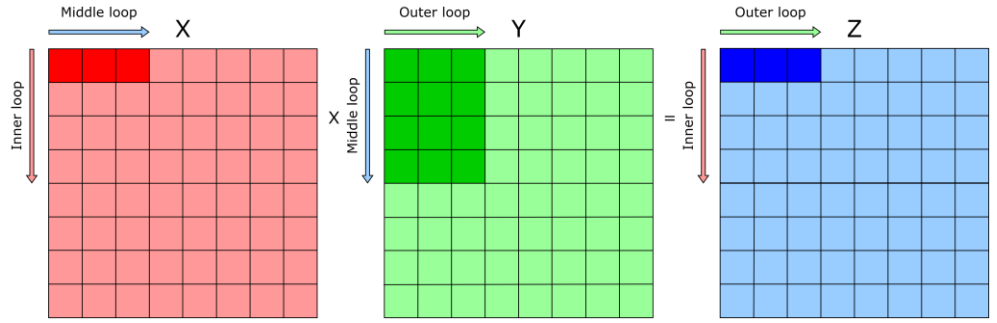
Gustavson options

- **There are three options**
  - X reuse: the elements of X are reused as much as possible and not changed in the inner loop.
  - Y reuse: the elements of Y are reused as much as possible and not changed in the inner loop.
  - Z reuse: the elements of Z are “reused” as much as possible meaning that, they are fully computed before being stored back

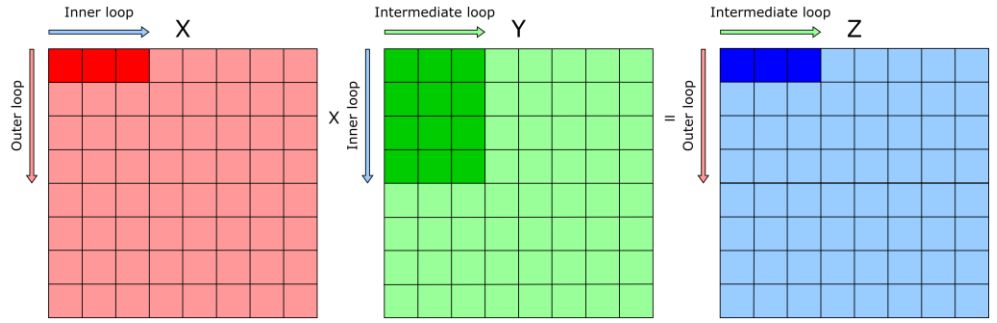
Gustavson X reuse:



Gustavson Y reuse:

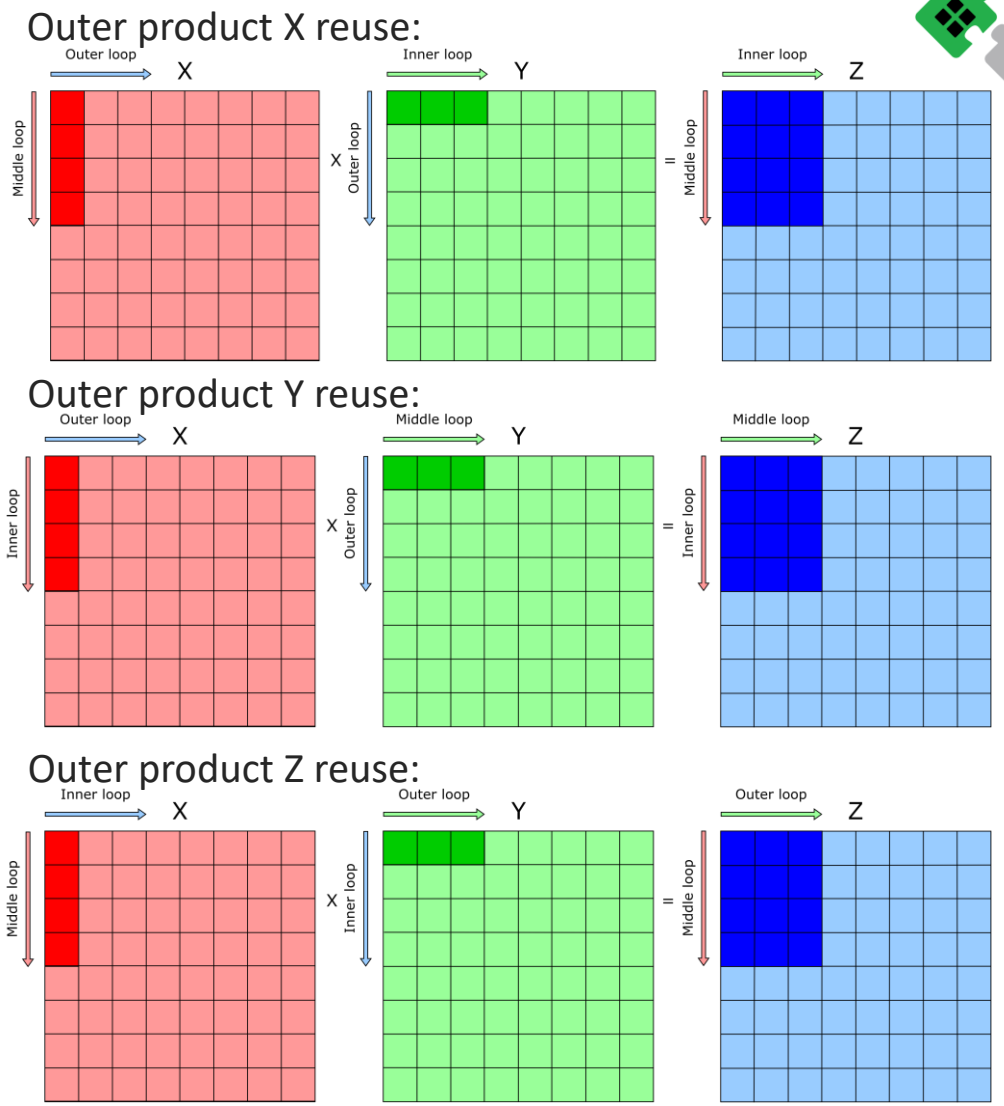


Gustavson Z reuse:

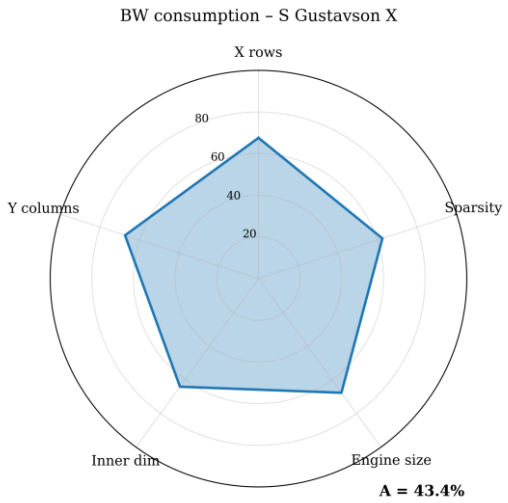


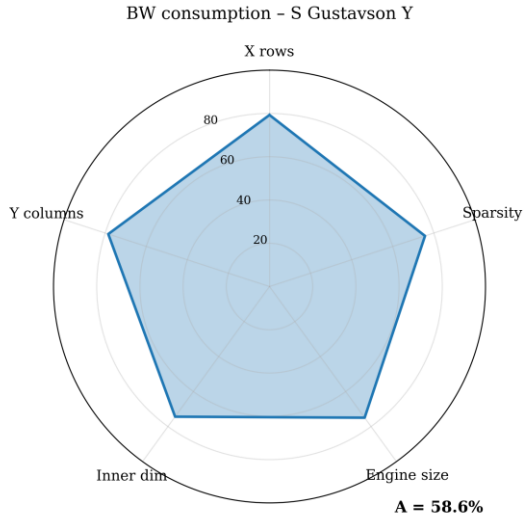
Outer product options

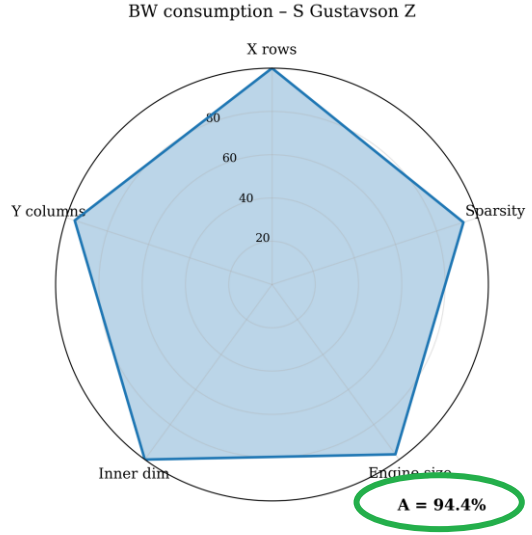
- **There are three options**
  - X reuse: the elements of X are reused as much as possible and not changed in the inner loop.
  - Y reuse: the elements of Y are reused as much as possible and not changed in the inner loop.
  - Z reuse: the elements of Z are “reused” as much as possible meaning that, they are fully computed before being stored back

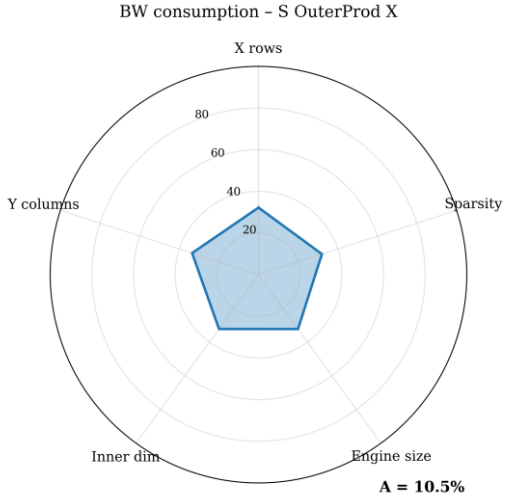


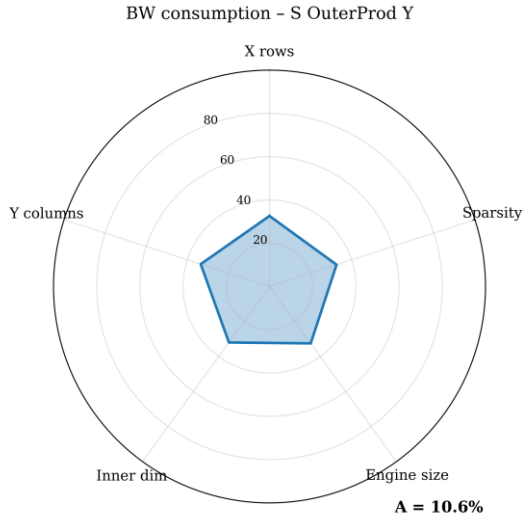
Dataflows comparison –BW consumption

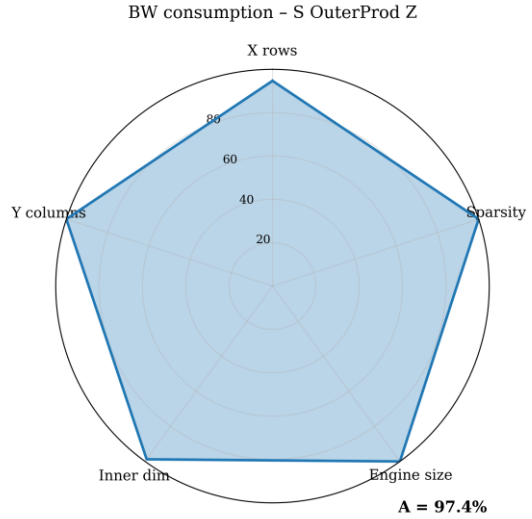




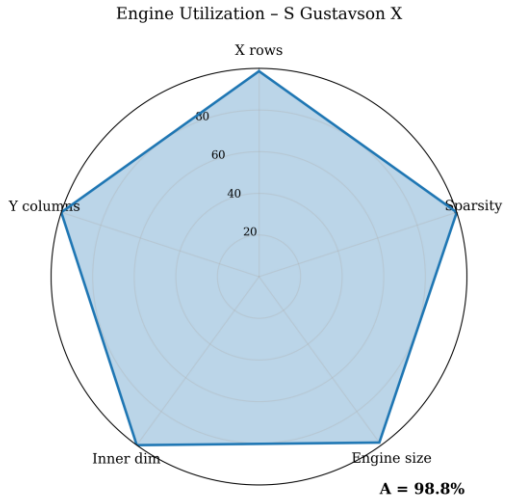


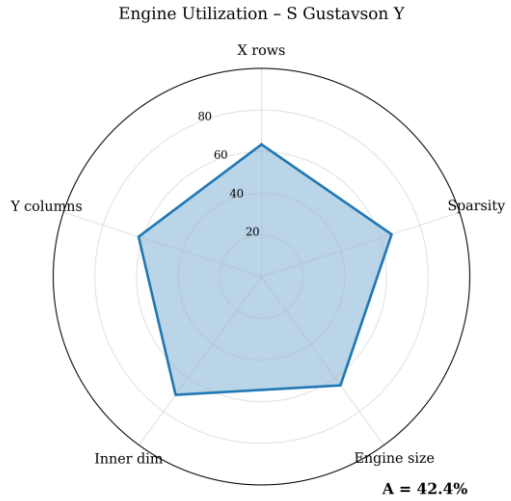


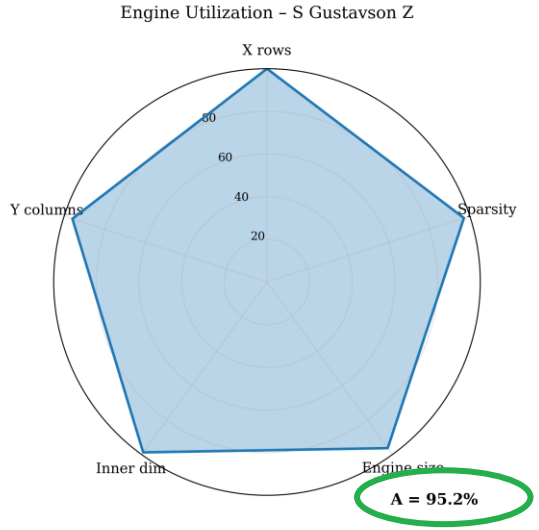


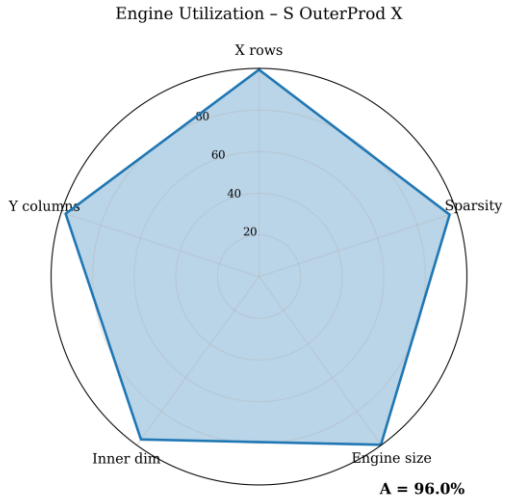


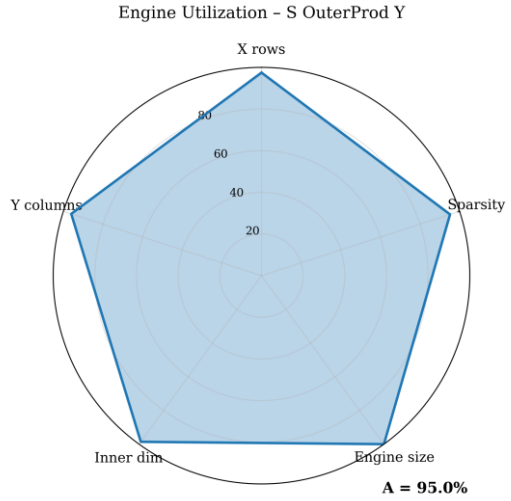
Dataflows comparison –Engine utilization

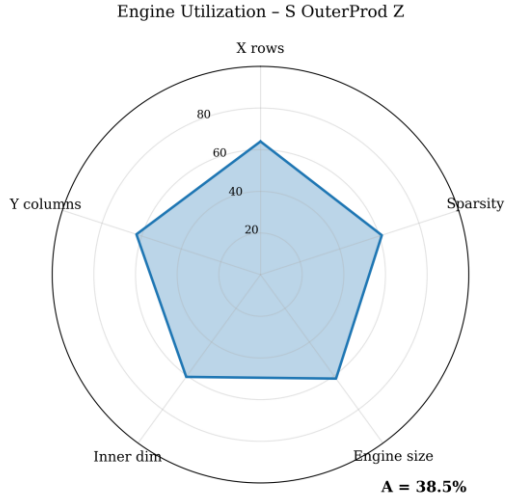




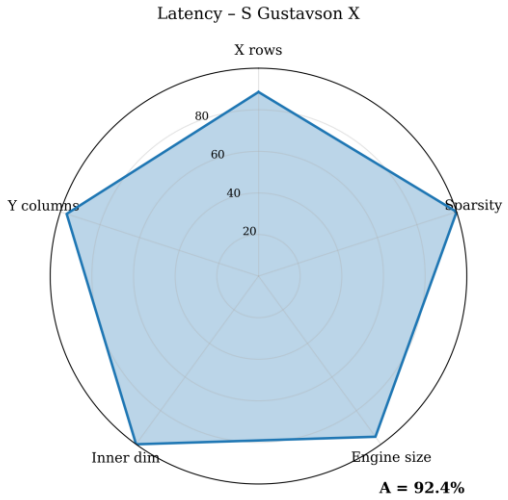


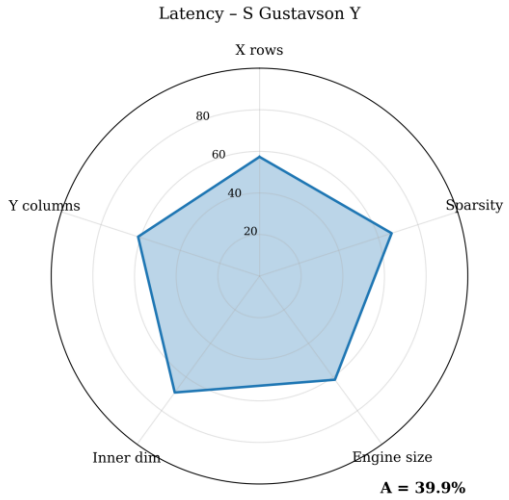


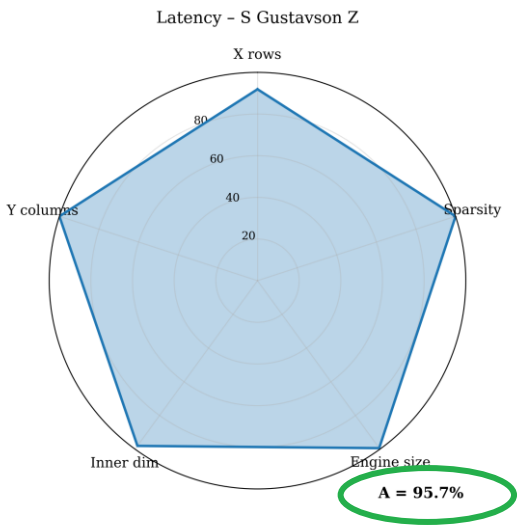


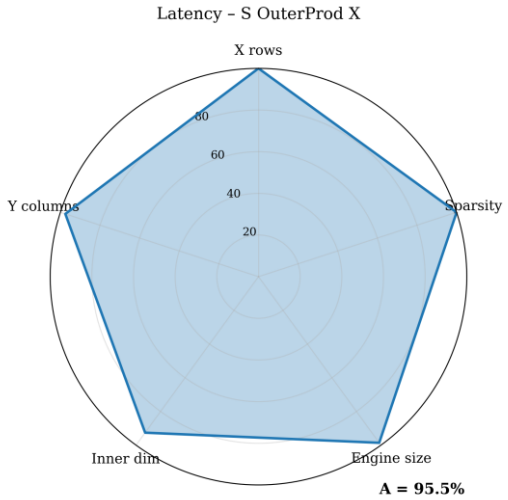


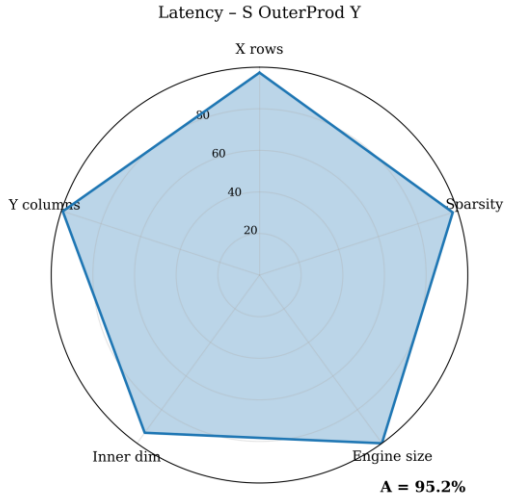
Dataflows comparison –Latency

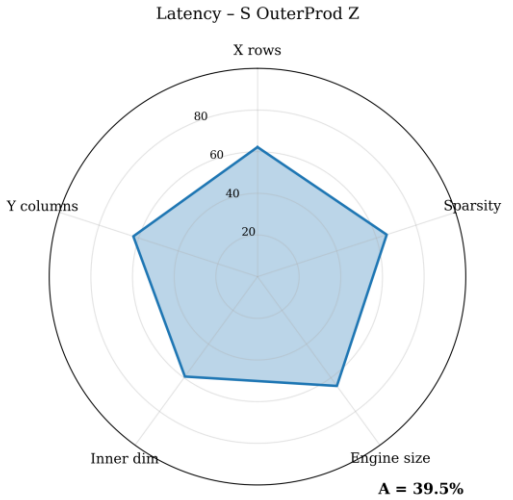






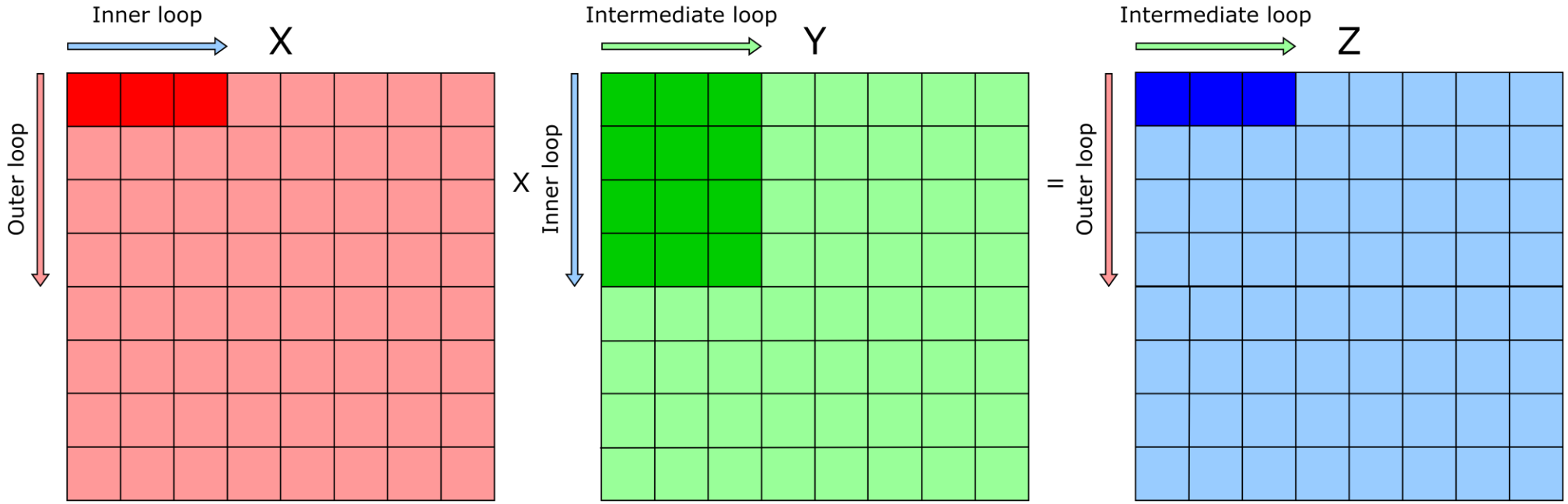






Final choice

- **Overall the gustavson dataflow seems to be the best**
  - It makes sense because you never load chunks of Z from memory

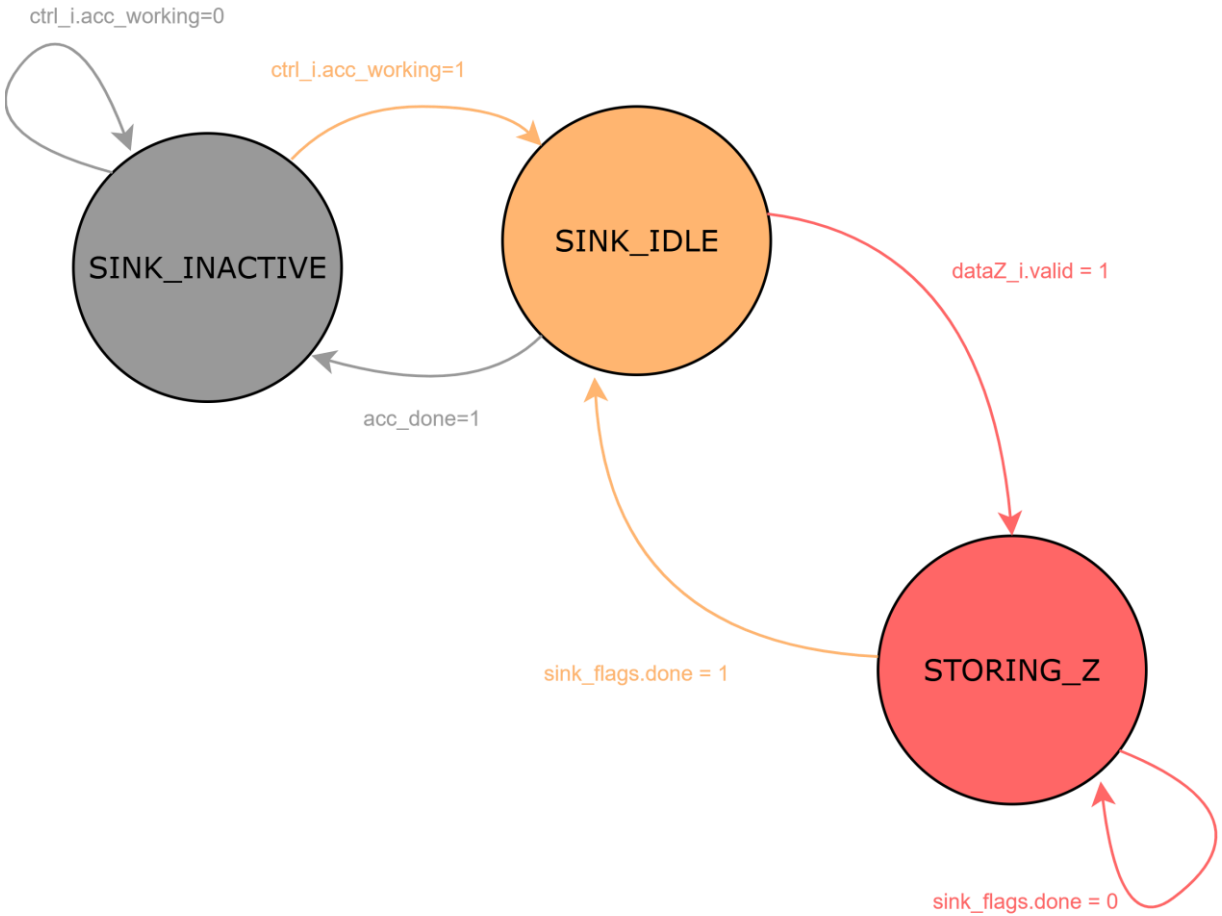


Gustavson dataflow with Z stationary

Streamer behavior –Loading part

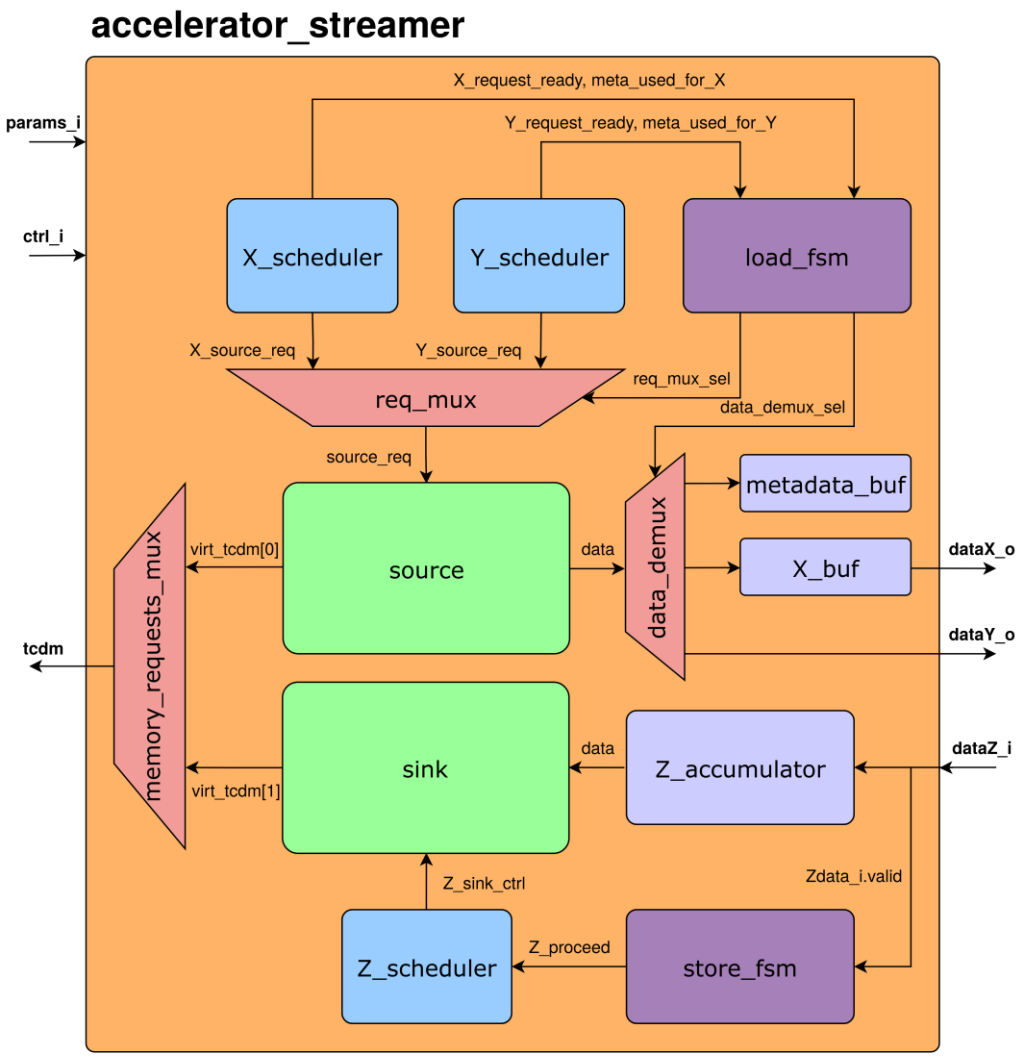
Loading FSM

Streamer behavior –Storing part



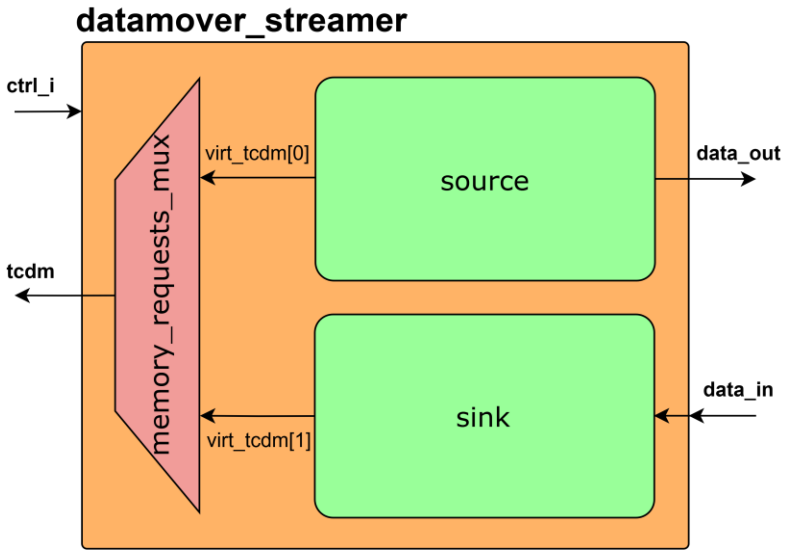
Storing FSM

Accelerator's streamer architecture VS datamover one



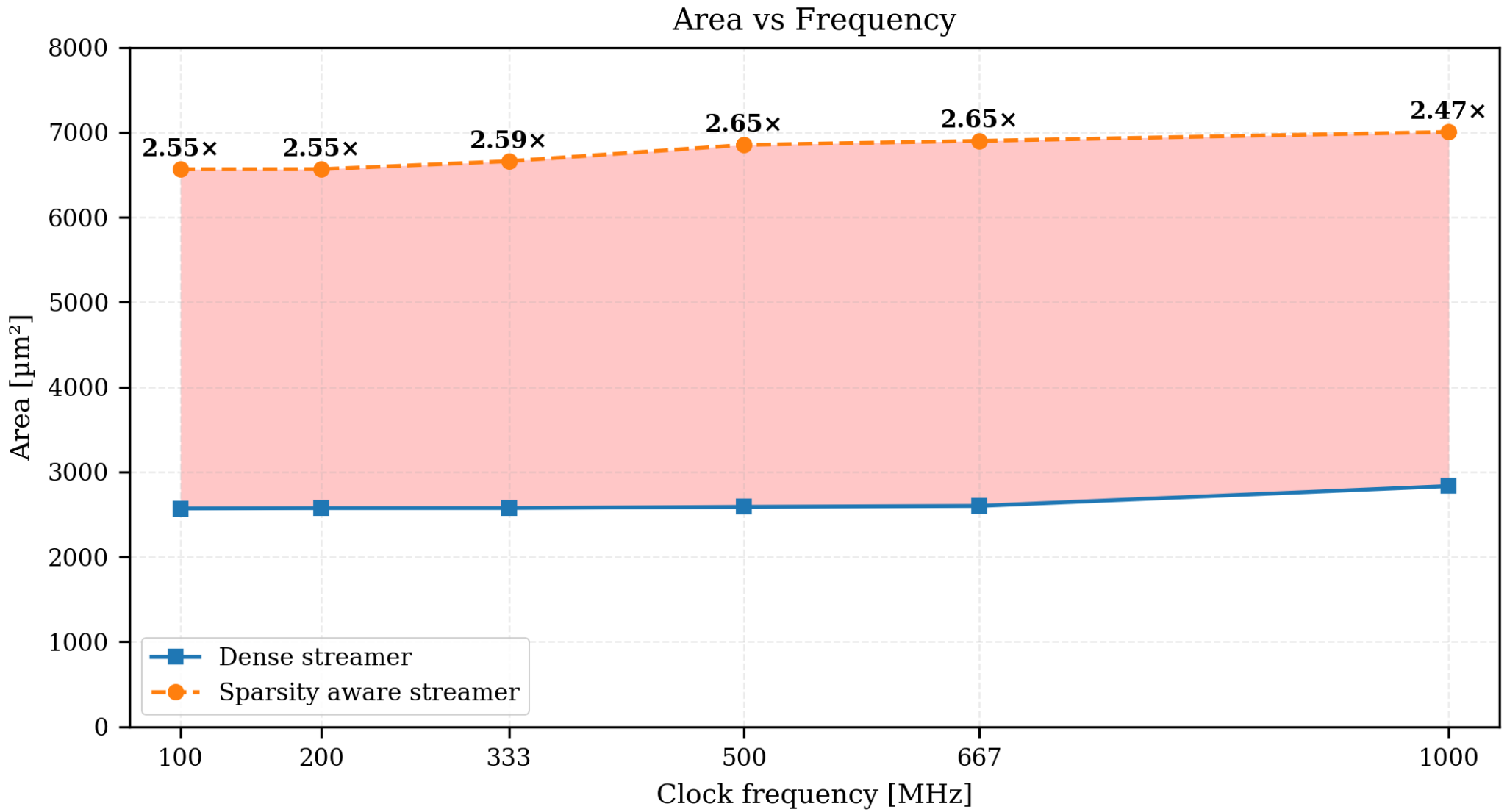
Sparsity-aware streamer architecture

VS



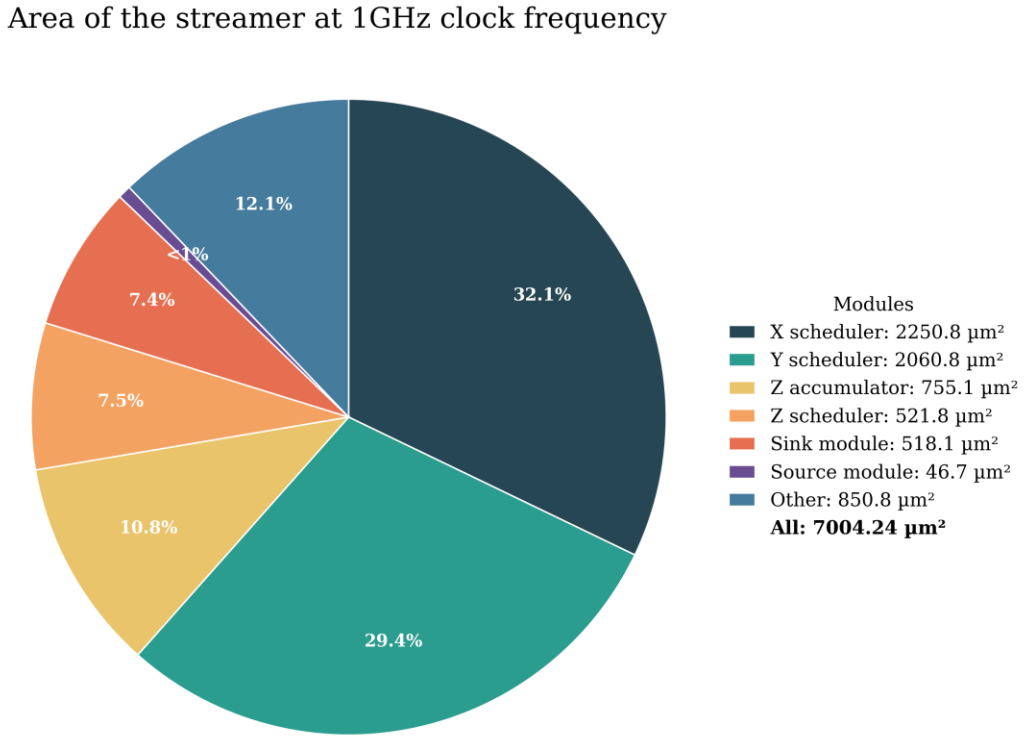
Non-sparsity-aware streamer architecture

Synthesis results- A VS f plot



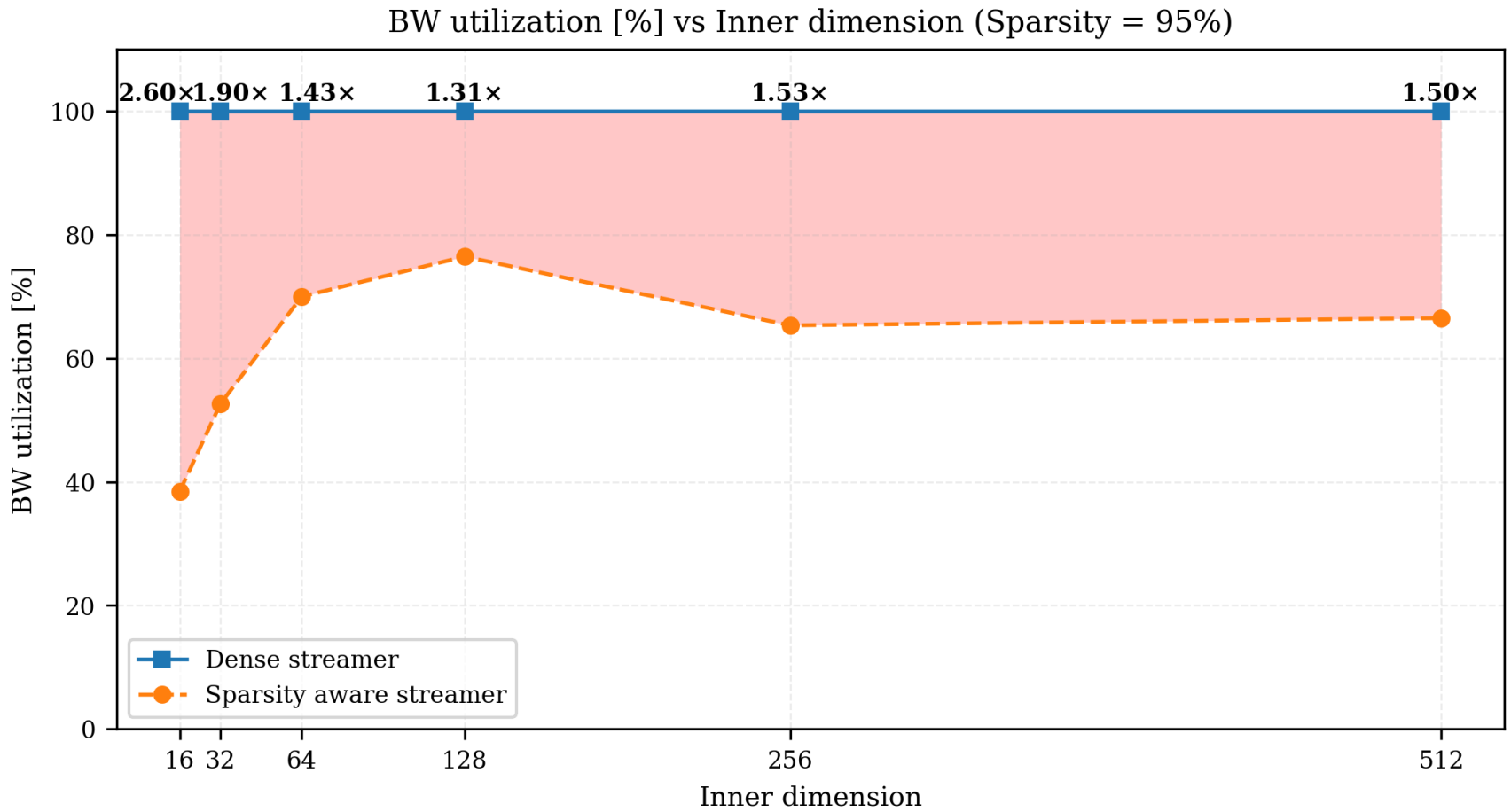
Synthesis results- Area contributions

- **Adding intelligence is expensive**
  - The biggest area contributions come from the schedulers
  - There is also an accumulator for Z that has a register buffer as wide as a whole block of Z; in this case 256bits.
- **Different function of the source module**
  - In the datamover streamer the source module was of comparable size with the sink module since they both did strided accesses. In my streamer this is no longer true.



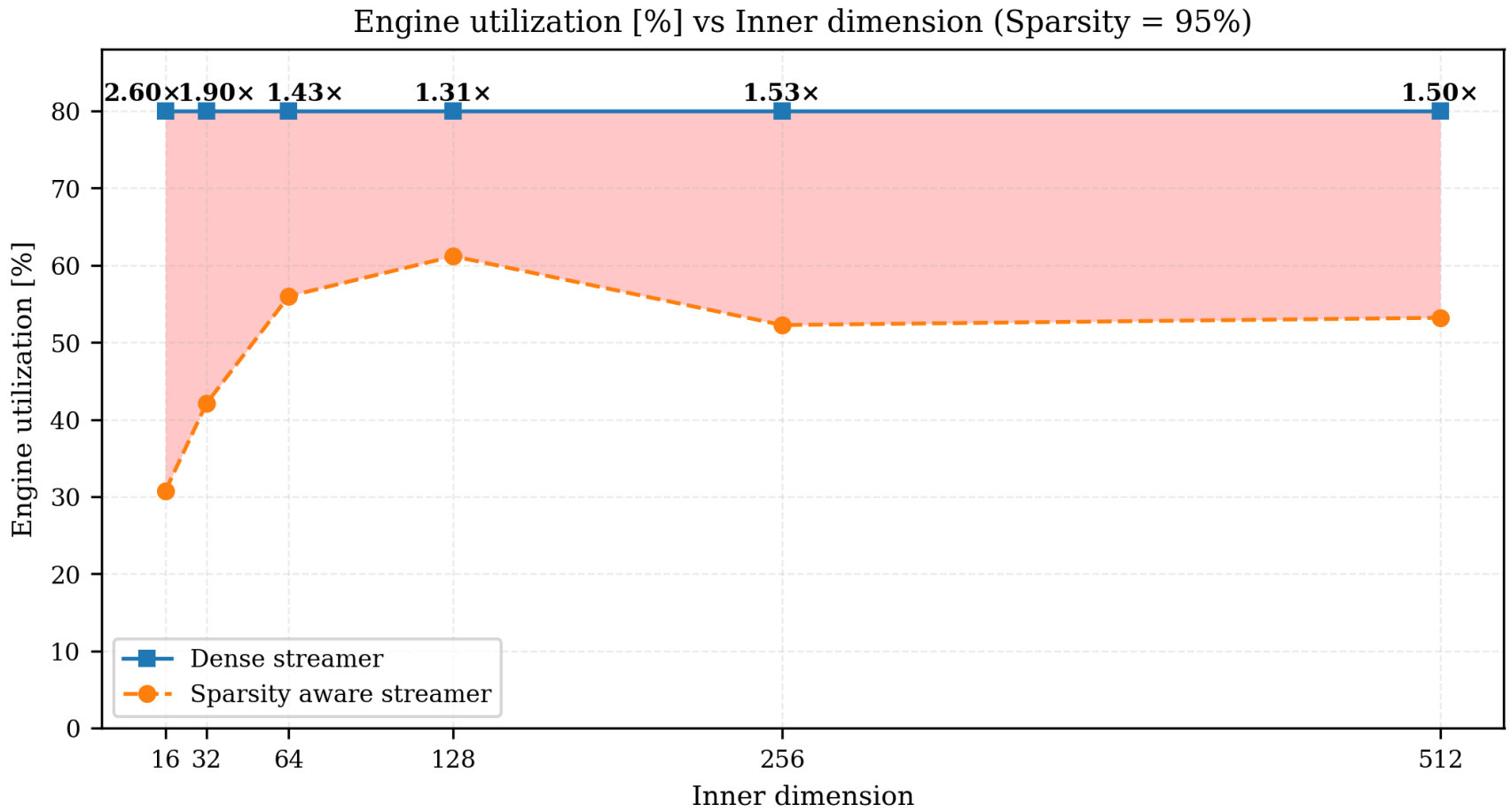
RTL results –Inner dimension (BW utilization)

All the results are acquired with an engine size of 4 MAC units, with an X matrix having two rows, a Y matrix having 16 columns and a metadata chunk of 128bits



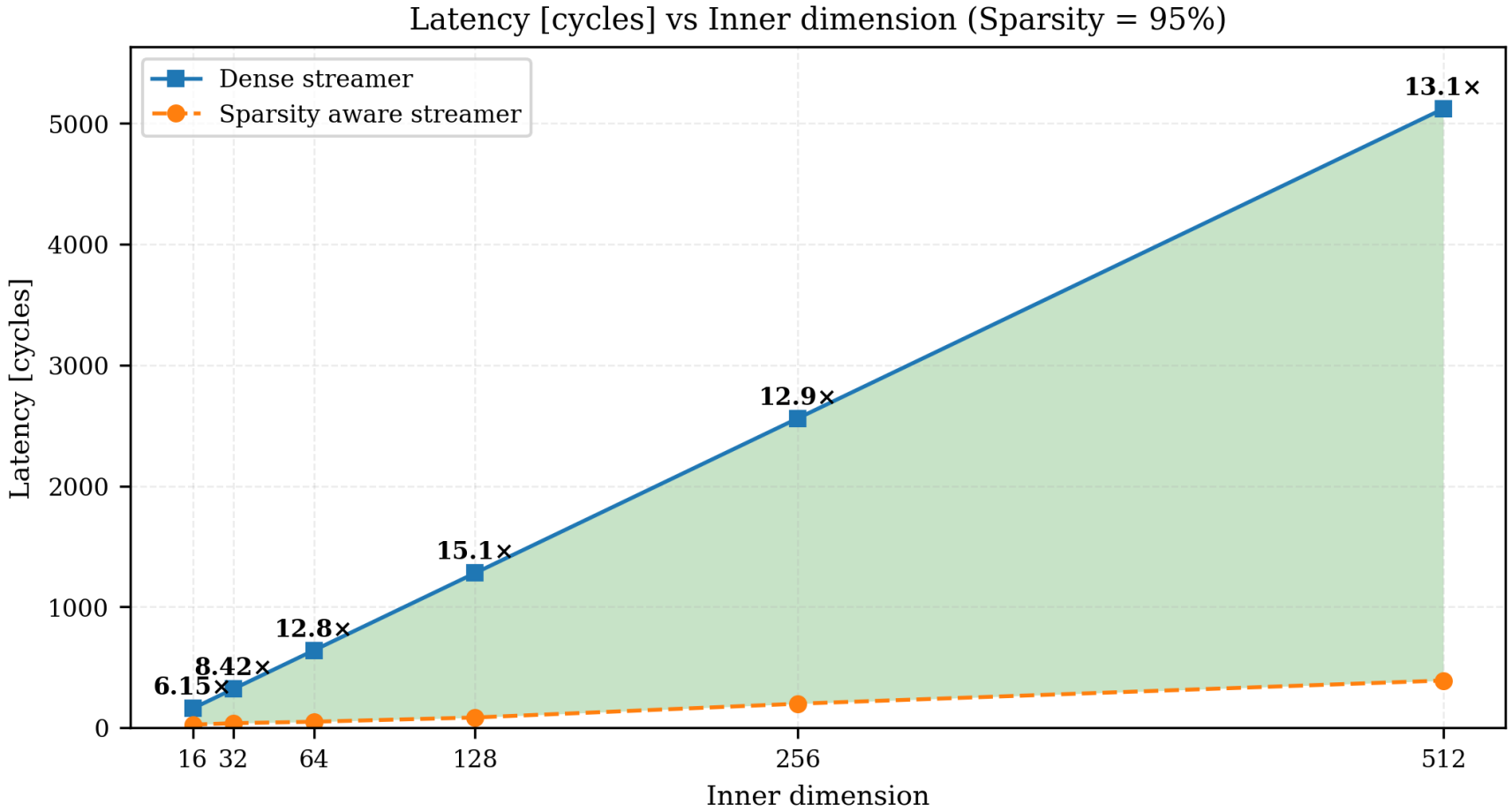
RTL results –Inner dimension (engine utilization)

All the results are acquired with an engine size of 4 MAC units, with an X matrix having two rows, a Y matrix having 16 columns and a metadata chunk of 128bits



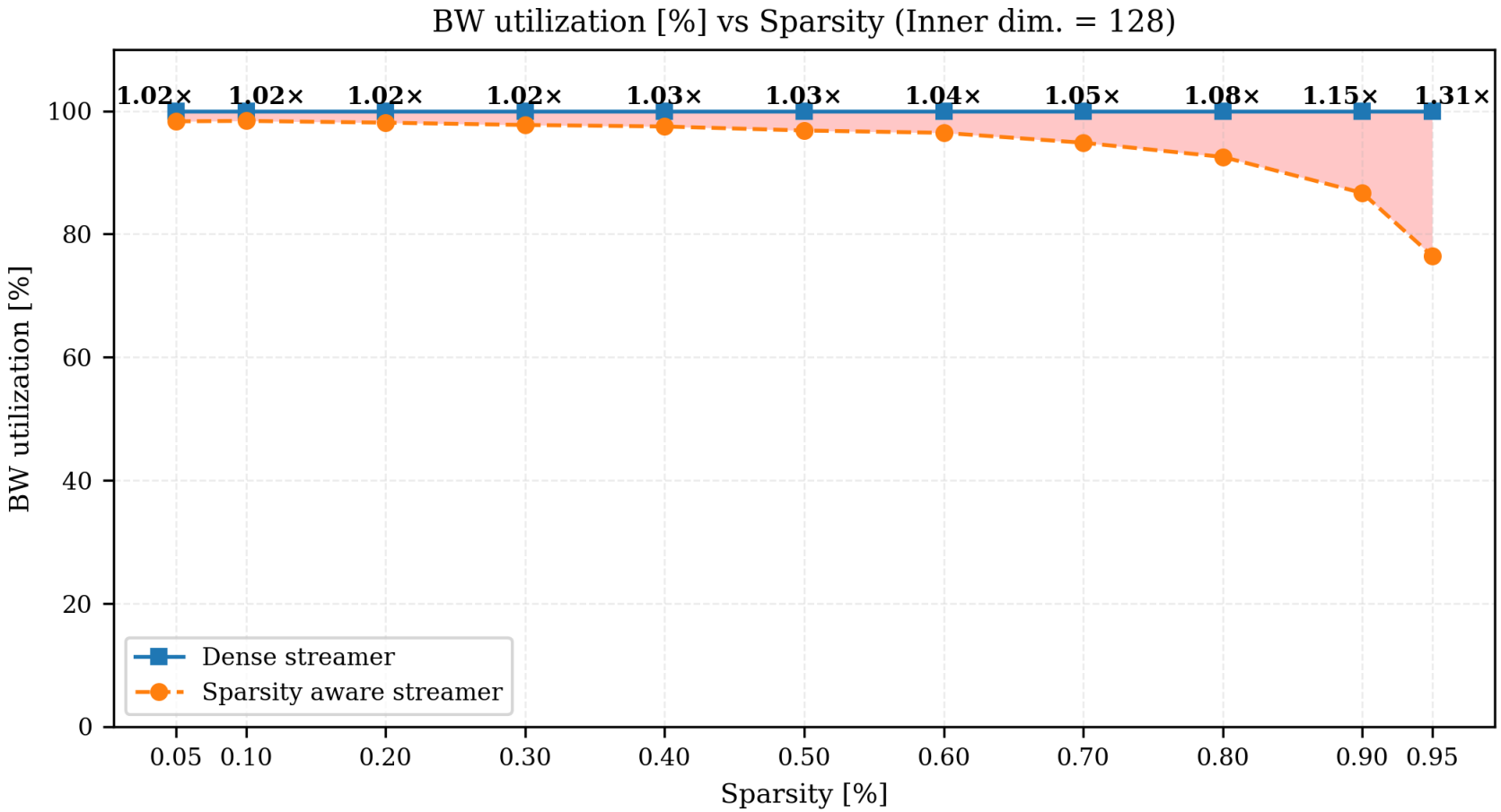
RTL results –Inner dimension (latency)

All the results are acquired with an engine size of 4 MAC units, with an X matrix having two rows, a Y matrix having 16 columns and a metadata chunk of 128bits



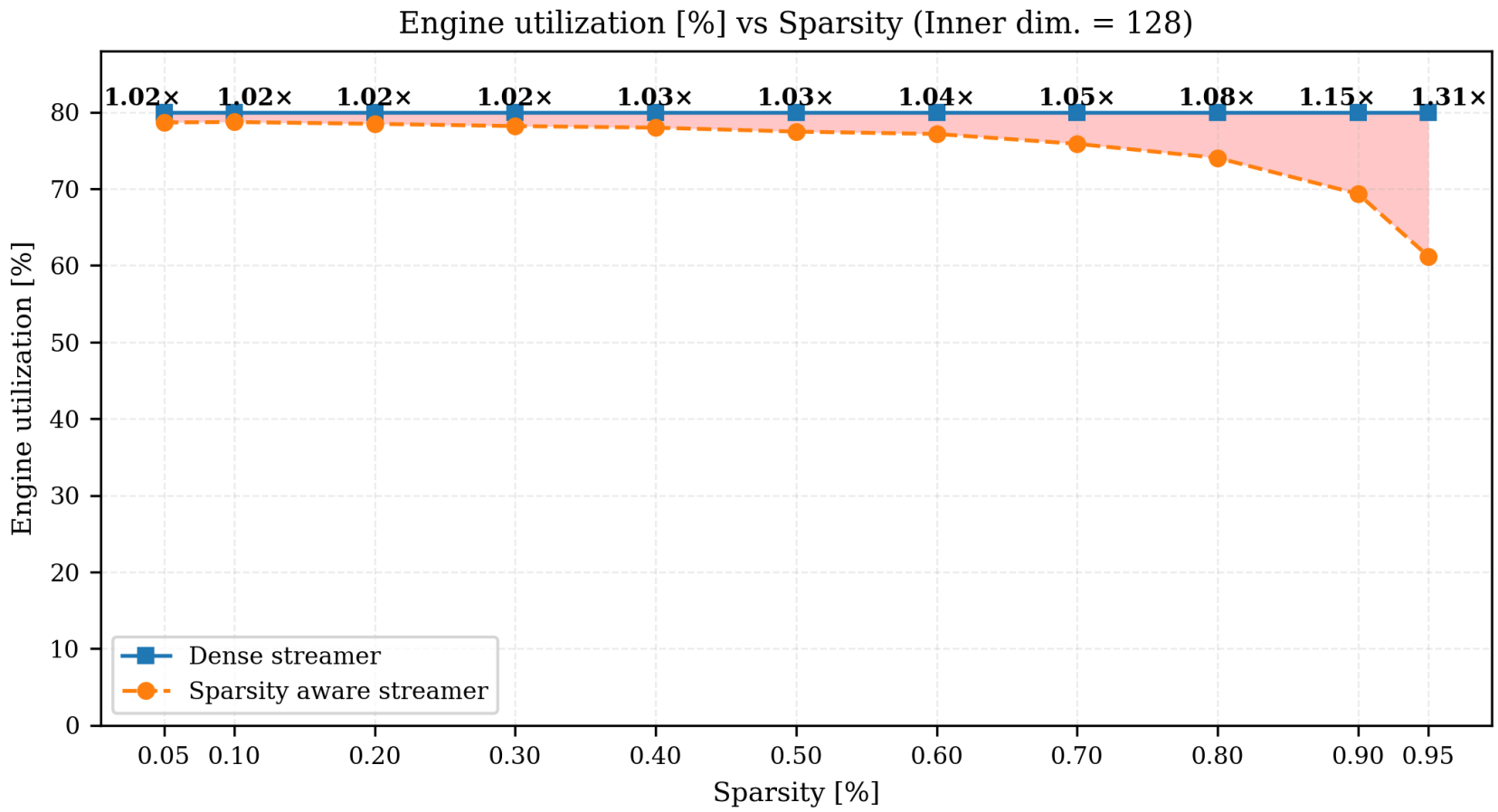
RTL results –Sparsity (BW utilization)

All the results are acquired with an engine size of 4 MAC units, with an X matrix having two rows, a Y matrix having 16 columns and a metadata chunk of 128bits



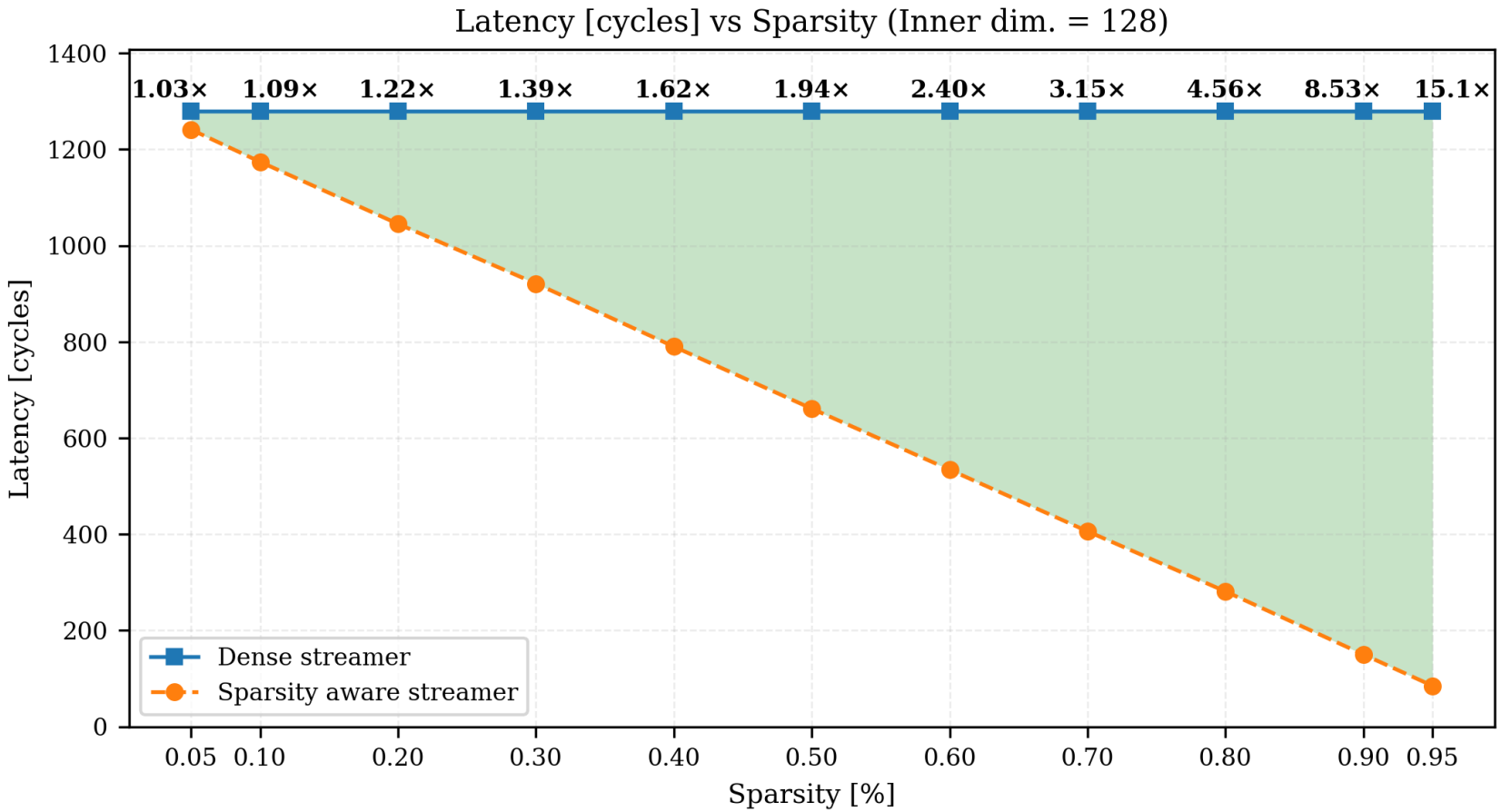
RTL results –Sparsity (engine utilization)

All the results are acquired with an engine size of 4 MAC units, with an X matrix having two rows, a Y matrix having 16 columns and a metadata chunk of 128bits



RTL results –Sparsity (latency)

All the results are acquired with an engine size of 4 MAC units, with an X matrix having two rows, a Y matrix having 16 columns and a metadata chunk of 128bits



- **GEMM is a fundamental operation**
- **We can exploit sparsity to accelerate it**
- **Problem: might be complicated to modify existing systems**
- **Solution: a flexible and portable accelerator whose streamer is smart enough to manage sparsity**
- **This costs in terms of area**

PULP Platform
Open Source Hardware, the way it should be!

pulp-platform.org
@pulp_platform
company/pulp-platform
youtube.com/pulp_platform

Institut für Integrierte Systeme, ETH Zürich
Gloriastrasse 35
Zürich, Switzerland

DEI, Università di Bologna
Viale del Risorgimento 2
Bologna, Italy

ETH zürich
ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA