ASA: Scheduling

Simulation

The focus of the last two weeks has been on implementation (simulation) and generalization

To schedule a DAG we go through an analysis that estimates backpressure (the computation of the streaming intervals, buffer space), and heuristics to derive streaming blocks and schedule them. We would like to:

- Assess the Quality of Result (is our schedule good?)
- Check for Correctness: we will need to compute buffer space (TODO). We want to be sure that it is computed correctly so that it prevent deadlocks

We want to use Discrete Event Simulation to do this

How this is implemented

To build the simulation we take into account:

- data communication volumes and dependencies, as expressed in the given DAG;
- communication type (streaming/non-streaming), as decided by our streaming blocks;
- PE assignments (and starting time) of each task, as decided by the scheduling heuristic;

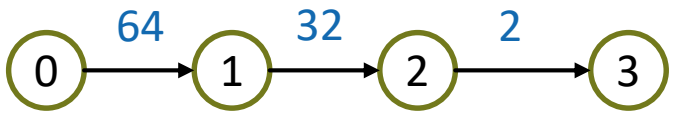
We create a Discretely Simulated Process for each Task:

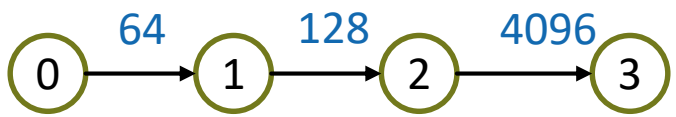
- Streaming communications between tasks are modeled using FIFO channels. They have finite size, *and are dimensioned according to the computed buffer size*.
- Non-streaming communications are modeled by means of events: when a task completes, it will notify the termination to all the waiting tasks.

Then we run the Simulation and we compare the simulated makespan with the computed one

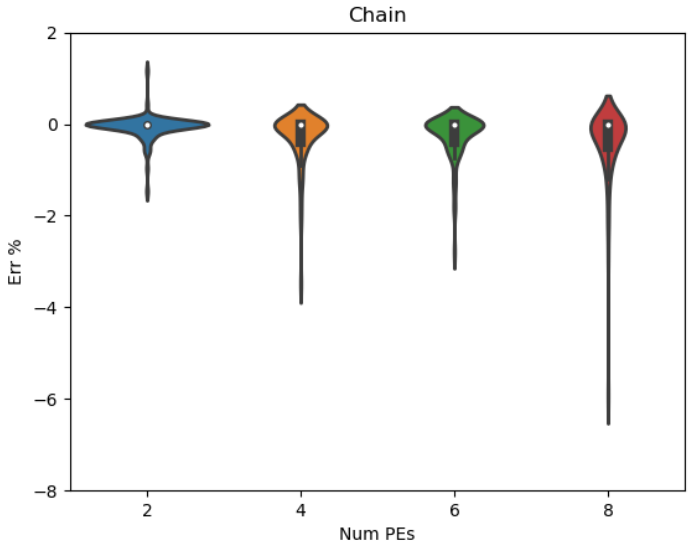
Results of simulation discrepancy

First results on DAGs that do not deadlock: we fix the topology and we randomly generate instances with different data volumes/nodes.

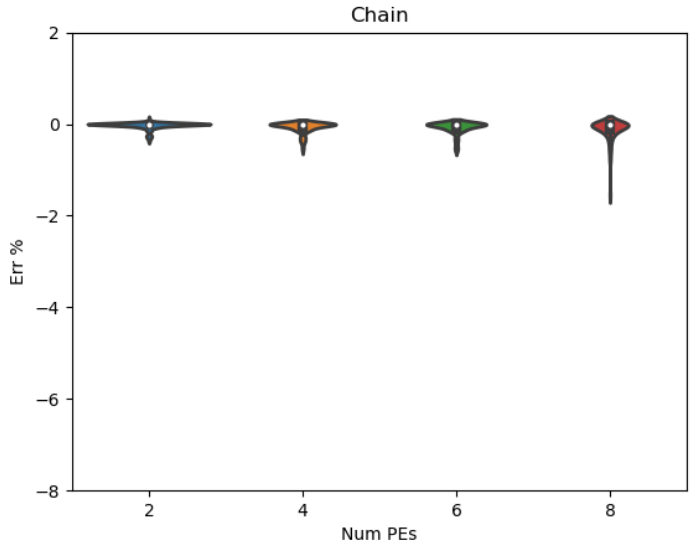




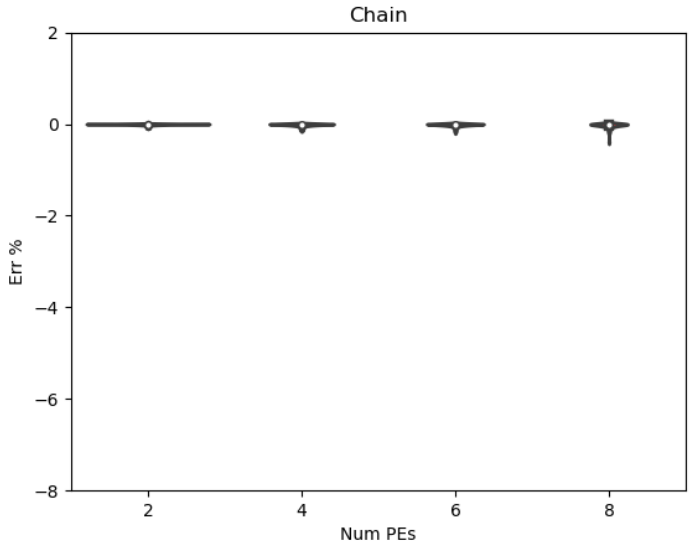
• • •



W = 128

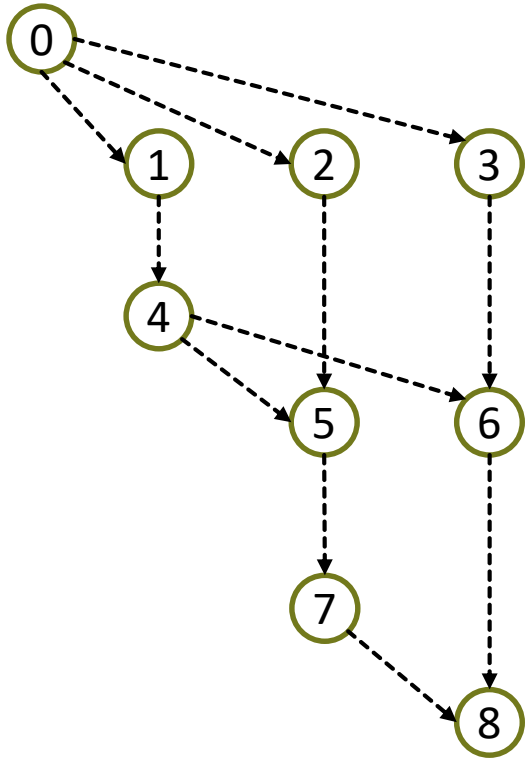


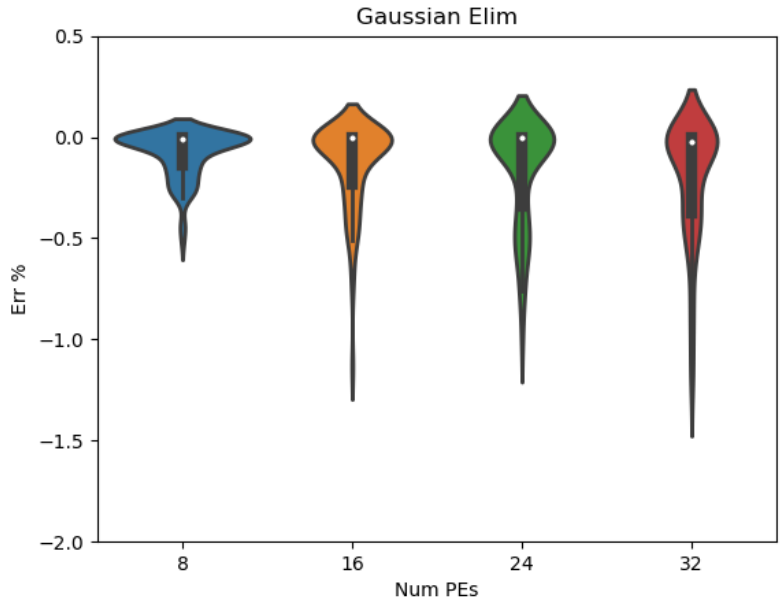
W = 512



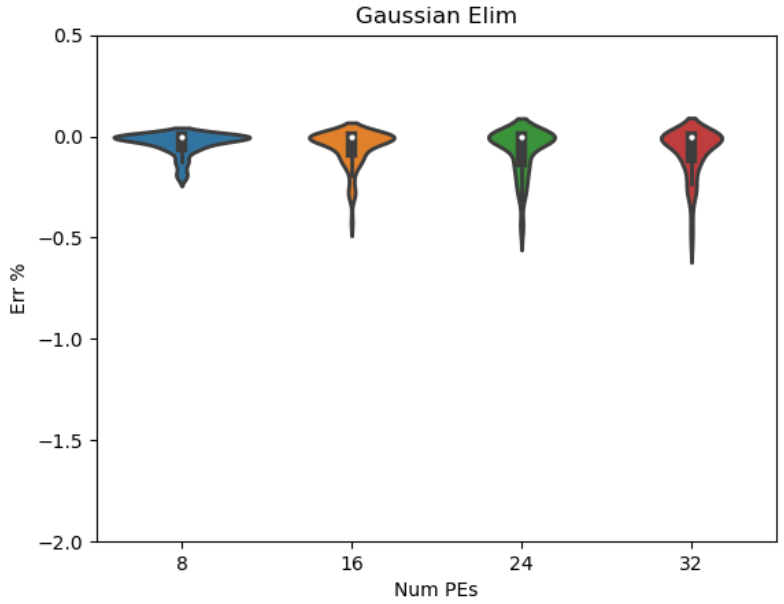
W = 4096

Results of simulation discrepancy



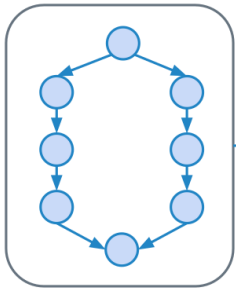


W = 128



W = 512

General toolchain

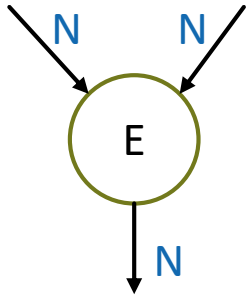


INPUT

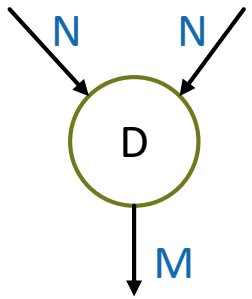
WIP

Type of tasks

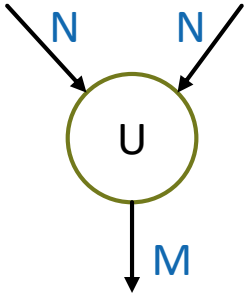
We begin by restricting ourselves to certain types of nodes



Element-Wise
(e.g., vect add)



Down-sampler
(M <N, e.g. dot product)



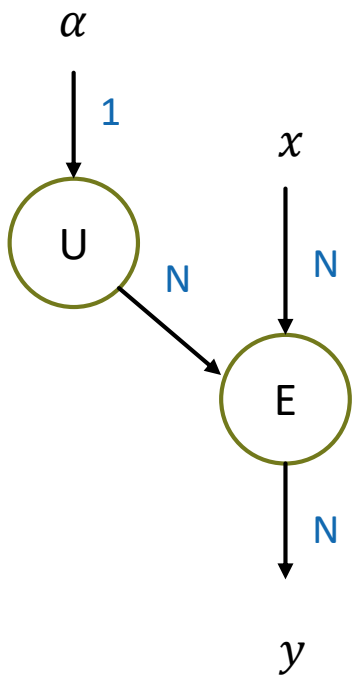
Up-sampler
(M>N, e.g., out.product)

Up-samplers may trigger backpressure effect

What about more generic computations?

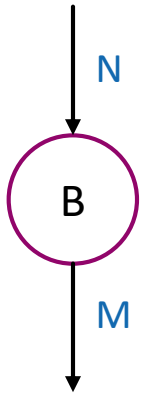
Vector Scaling

Suppose that you want to compute a vector scaling $y = \alpha x$, where α is a scalar and x is an N-element vector. How we can represent this with our restricted set of nodes?



Buffer Node

To model more complicated computations we introduce a new node: the *Buffer node.* A buffer node stores its inputs and once all stored, each element is output multiple times.

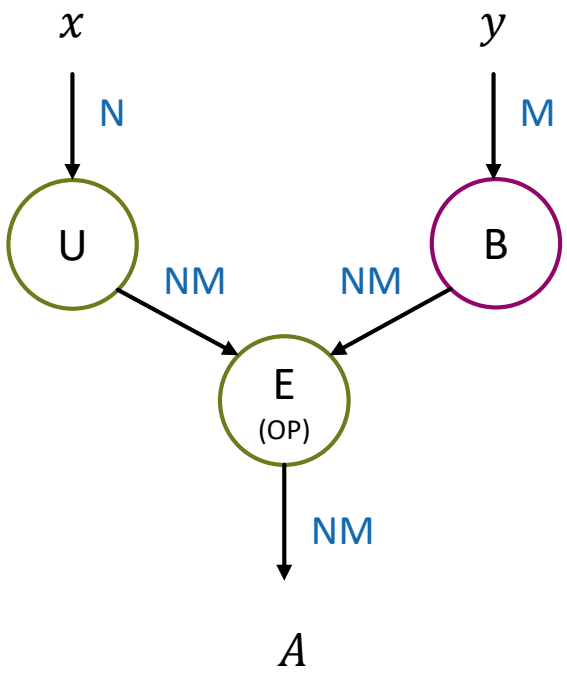


Given its behavior, we can not stream through the Buffer node: it acts as a barrier

Outer product

We want to compute $A = xy^T$, where *x* is an N-element vector and *y* is an M-element vector.

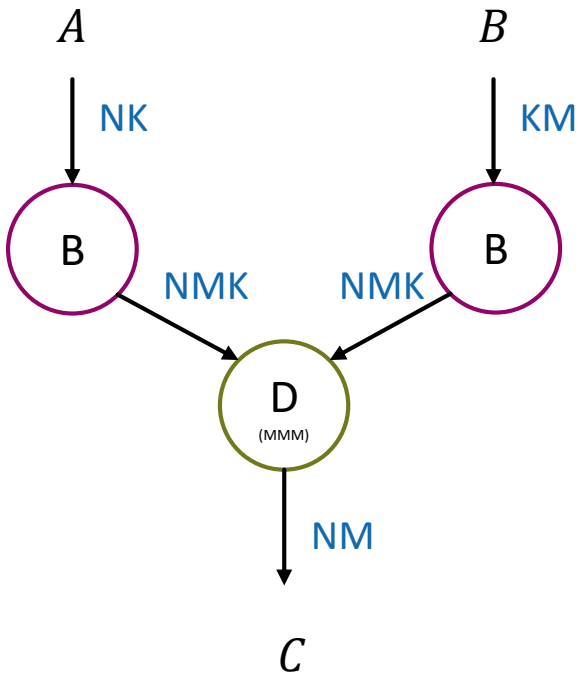
Let's assume that the outer product is computed as follows: every single element of *x* is multiplied by the entire *y* to compute a single row of *A*



Matrix-Matrix Multiplication

We want to compute $C = AB$, where A is a $N \times K$ matrix, B is a $K \times M$ matrix and C is a $N \times M$ matrix.

Let's assume it is implemented in the naïve way: we can exploit again the buffer node



It seems that we can express more general computation using the restricted and buffer nodes.

The representation is strictly dependent from the actual implementation.

We need to further study how much this is generally applicable

TODO

- Study and propose solution for the buffer space
- Continue with implementation/validation with Simulation
- [Study how to represent more generic computations and unify theory]