ASA: Scheduling

Optimization problem

<u>Streaming intervals</u>

Given a node with a production rate R(v), we define its streaming interval accordingly

$$S^+(v) = S^-(v)/R(v)$$

Let $WCC(v)$ be the weakly connected component that contain the node v. Then we can show that

$$S^+(v) = \frac{\max\limits_{u \in WCC(v)} K^-(u)}{K^+(v)} \quad .$$

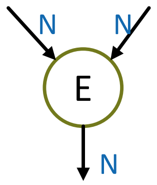
Where $K^-(u)$ is the number of elements read by a node u, while $K^+(u)$ is the number of element being produced

This tell us that we have to look at the maximum data volume to understand how the nodes may be slow-down by the streaming effect.

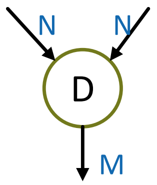
Optimization problem

Last output: the last-out time $LO\ (u)$ be the time the last element leaves node u

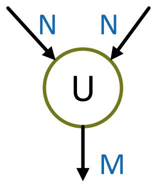
$$LO(v) = \max(R(v), 1) + \max_{(u,v)\in E(G)} LO(v)\ .$$



Element-Wise
(R(v) = 1)



Down-sampler
(R(v) < 1)



Up-sampler
(R(v)>1)

For source and buffer nodes, we can prove that this is:

$$(K^+(v) - 1)S^+(v) \leq LO(v) \leq K^+(v)S^+(v)$$

$$S^+(v) = \frac{\max_{u\in WCC(v)} K^-(u)}{K^+(v)}\ .$$

The LO relates with the maximum data volume that is in the weakly connected component

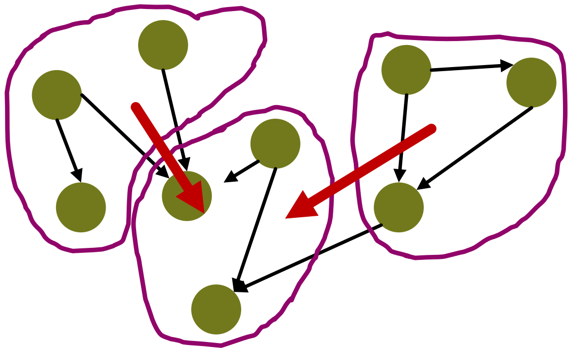
Optimization problem

When we have P < N, we need to partition the DAG into Streaming Blocks of at most P nodes.
Such partitioning must be done so that the overall execution time ($\max_{\{v \in V\}} LO(v)$) is minimized

The discussion in the previous slide show the last-out time relates to the maximum amount of data produced by the nodes in the graph. This result allows us to define the scheduling problem as an optimization problem.

Given a canonical task graph, we want to partition it into streaming blocks containing at most P computational nodes, such that:

- The sum of the max value of each Streaming Block component is maximized
- The dependencies among SB still form an acyclic DAG



Heuristic

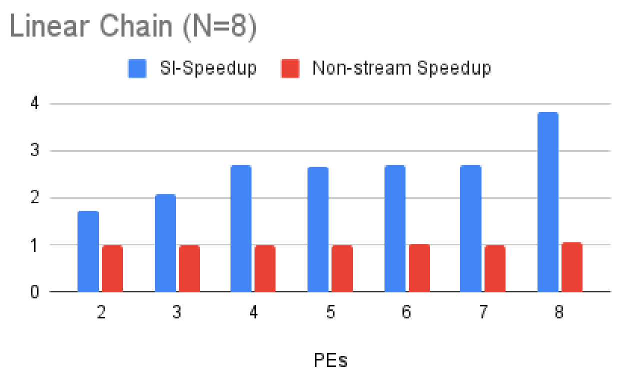
Streaming interval and new (guided) heuristics

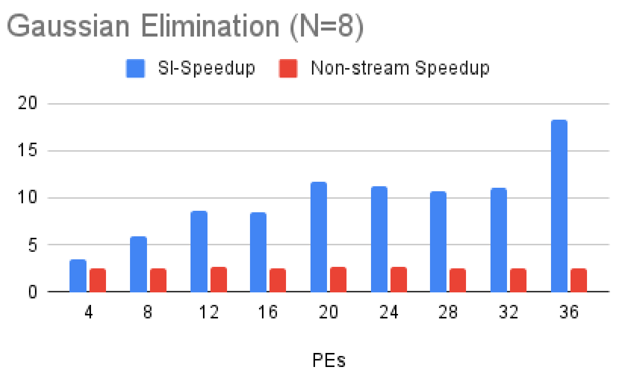
For a generic task graph, we build steaming blocks such that:

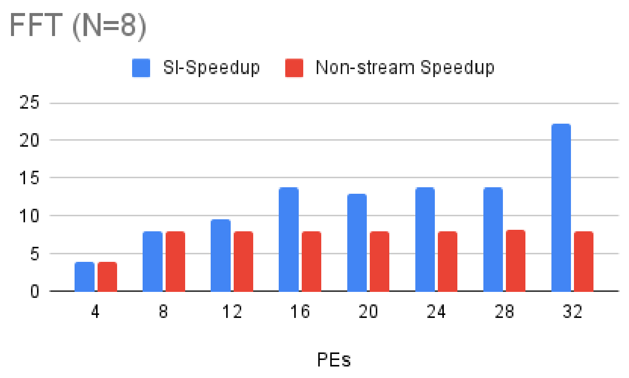
- We add a node to a streaming block if its produced data volume (work) is less than the data volume(work) produced by the block's source(s) from which it depends (if any).
- We continue adding to the same streaming block until such node exists or the block is full. Otherwise, we create a new streaming block, and we start filling it.

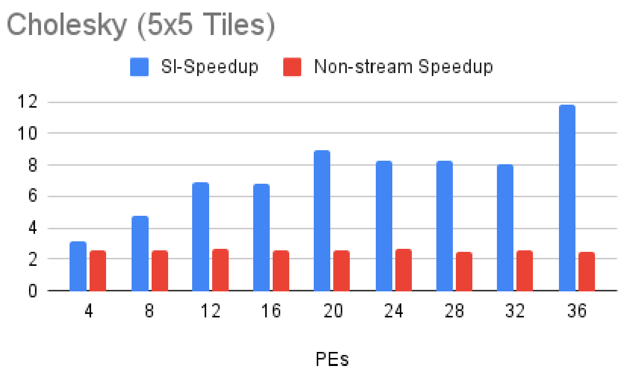
In this way, we are guaranteeing that the streaming interval of the block's sources is not increased by adding an upsampler node producing more data than the source itself, and no other node is slowed down by this. Note that in this case, a streaming block may have less than P tasks.

Results









- Being an heuristic there could be several trade-offs, and we want to investigate them
- Can we do something clever for special DAGs (e.g., composed only by certain types of nodes)?