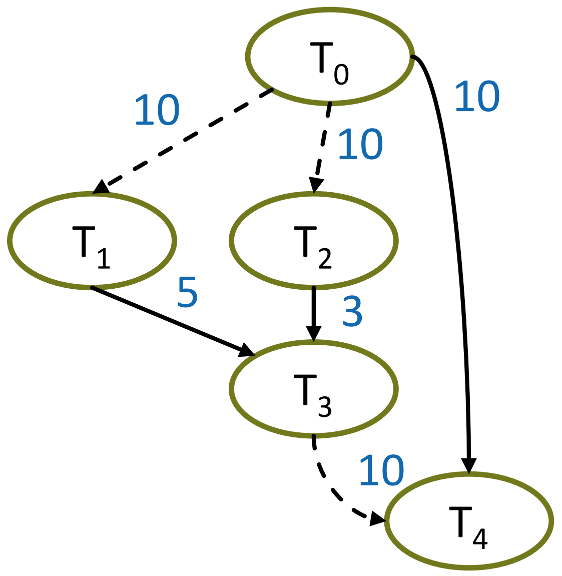
ETH zürich

spcl.inf.ethz.ch
@spcl_eth

D INFK

ASA: Scheduling

Input and assumptions



Input: a graph *G = (V,E)*, where:

- The nodes represent tasks (operations) in which the computation can be decomposed and that can be executed on a PE
- Edges represent data movements and dependencies. The labels indicate the number of data elements being transferred

Output: a schedule for *G* that minimizes its running time (makespan)

Target architecture:

- a spatial device composed by homogenous PEs
- PEs are fully connected
- Backing memory. PEs can always communicate through main memory

Communication: completely decoupled (solid edges) or pipelined (dashed)

Assumptions:

- Blocking read semantics
- Tasks are not preemptible
- Co-scheduling: weak or strict
- Communications occur w/o contention

Two subproblems

We need to solve two inter-related problems

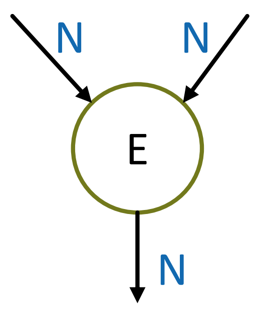
$$\begin{tikzcd}\text{Streaming} \arrow[r, leftrightarrow] & \text{Scheduling}\end{tikzcd}$$

Streaming: decide which edge implements a pipelined communication and which not

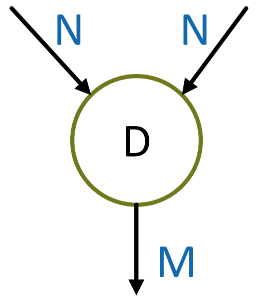
Scheduling: schedule the tasks considering pipelined communication

Type of tasks

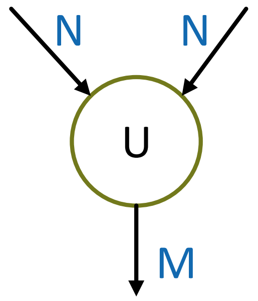
We begin by restricting ourselves to certain types of nodes



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



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



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

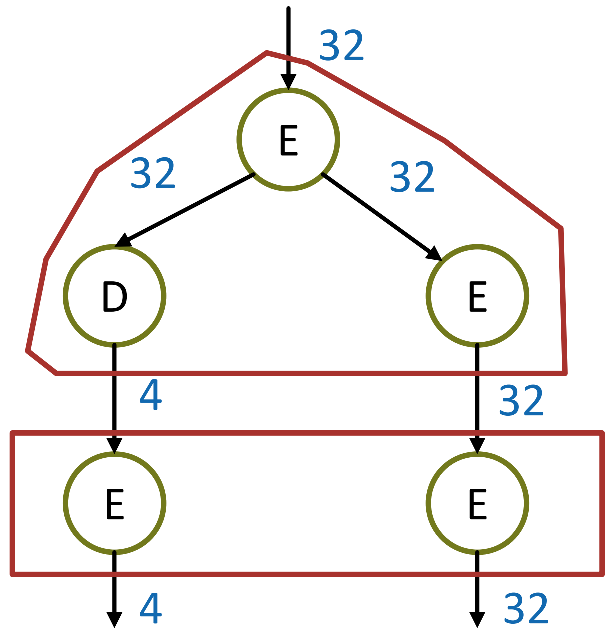
(we can also think to splitters/combiners node as special down-samplers/up-samplers)

For the moment being we assume that data volumes on incident edges is the same

For any node its running time (in isolation) is given by: $\max\{N, M\}$

Elwise-Downsampler nodes

In a DAG composed only by Elwise and Down-sampler nodes, we don't have any backpressure effect

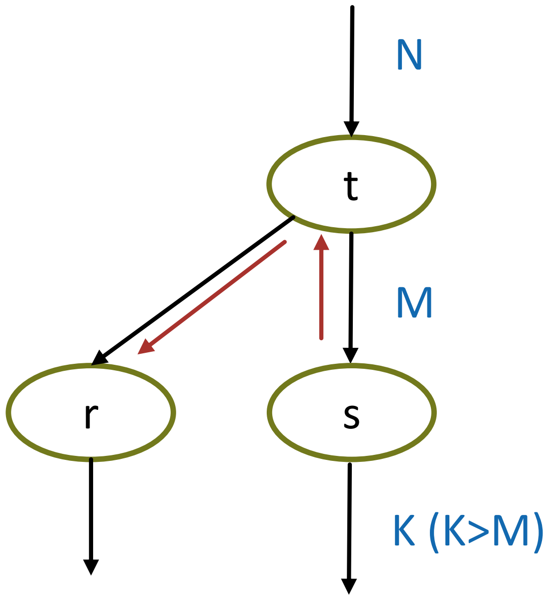


Going down level-by-level, the amount of work is non-increasing

We may want to partition this in streaming blocks where:

- Each block contains P nodes, where P is the number of PEs in the target architecture
- nodes have similar running time in order to increase the PE efficiency
- Communications within a streaming block are streaming
- Communications outside a streaming block are non-streaming

Upsampler



The running time is given by the volume of data being ingested/consumed

$$\overline{T}_t = \max\{N, M\}$$

Given an up-sampler with ratio *r* (K/M), it will ingest an input element every *r* unit of time, producing a new output element every unit of time

An up-sampler node may slow-down its parents. This affects its running time and the running time of other parent's descendants

$$T_t \geq \overline{T}_t$$

The backpressure effect must be taken into account while scheduling the tasks

We want to study this at the *steady-state* (synchronous nodes). No queuing theory or other, since it would never work…rather let's reason about at which time we should generate the data so that it is not backpressured?

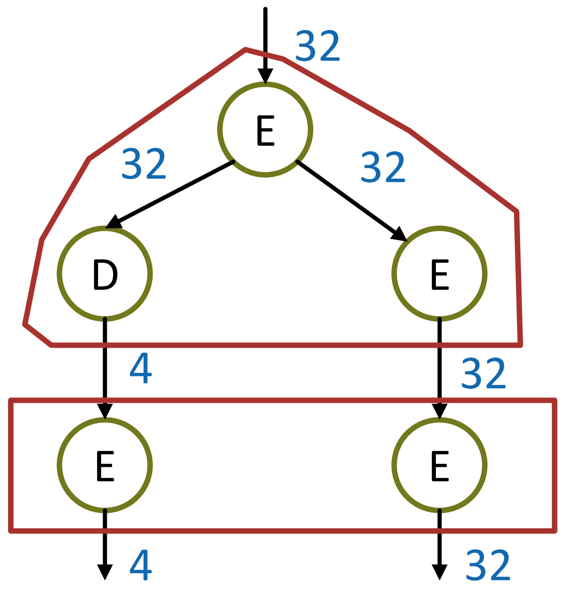
Alternatively: assume that we can always scale a task. This must be taken into account in the analysis

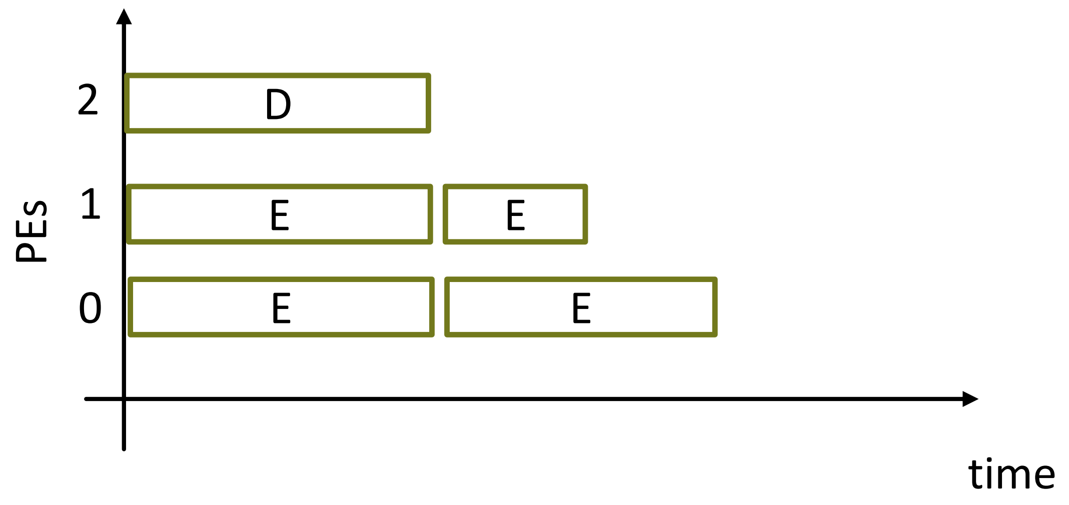
Scheduling

Streaming ⟷ **Scheduling**

Once we build our streaming blocks, we can perform the actual schedule. By means of analysis and heuristics for building the streaming blocks we can break this cyclic dependency

First approach: use a gang-scheduling heuristics: all nodes in a streaming block are scheduled together. When they finish, we move to the next streaming block (e.g., by reconfiguring the architecture)





TODO

- **Understand how to model backpressure**: we have some ideas there, need developments
- **Scheduling:** gang-scheduling seems the most straightforward solution, but maybe not the only one
- **Implementation:** need some prototype to validate/simulate our findings
- **[Study how to represent more generic computations]**