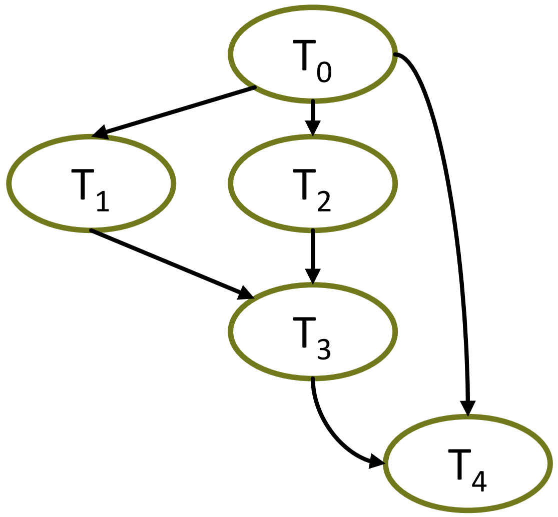
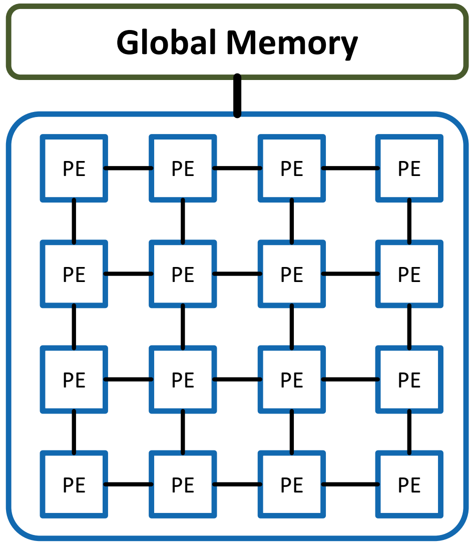
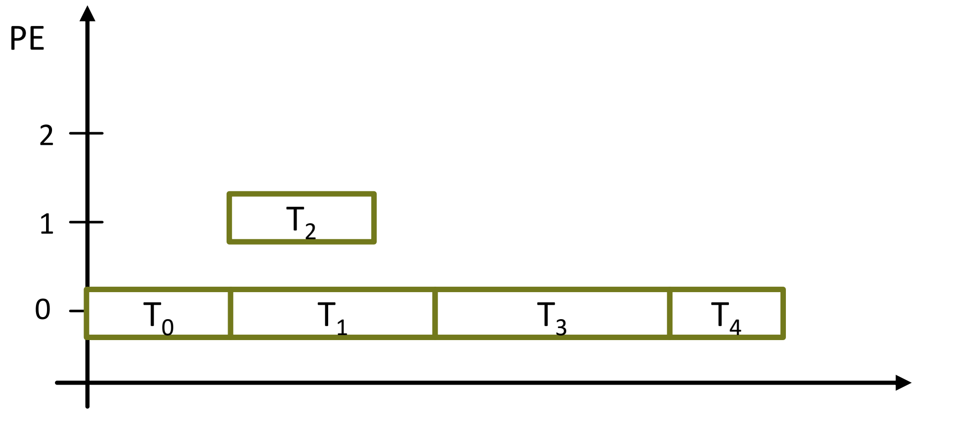
ASA: Scheduling

Scheduling





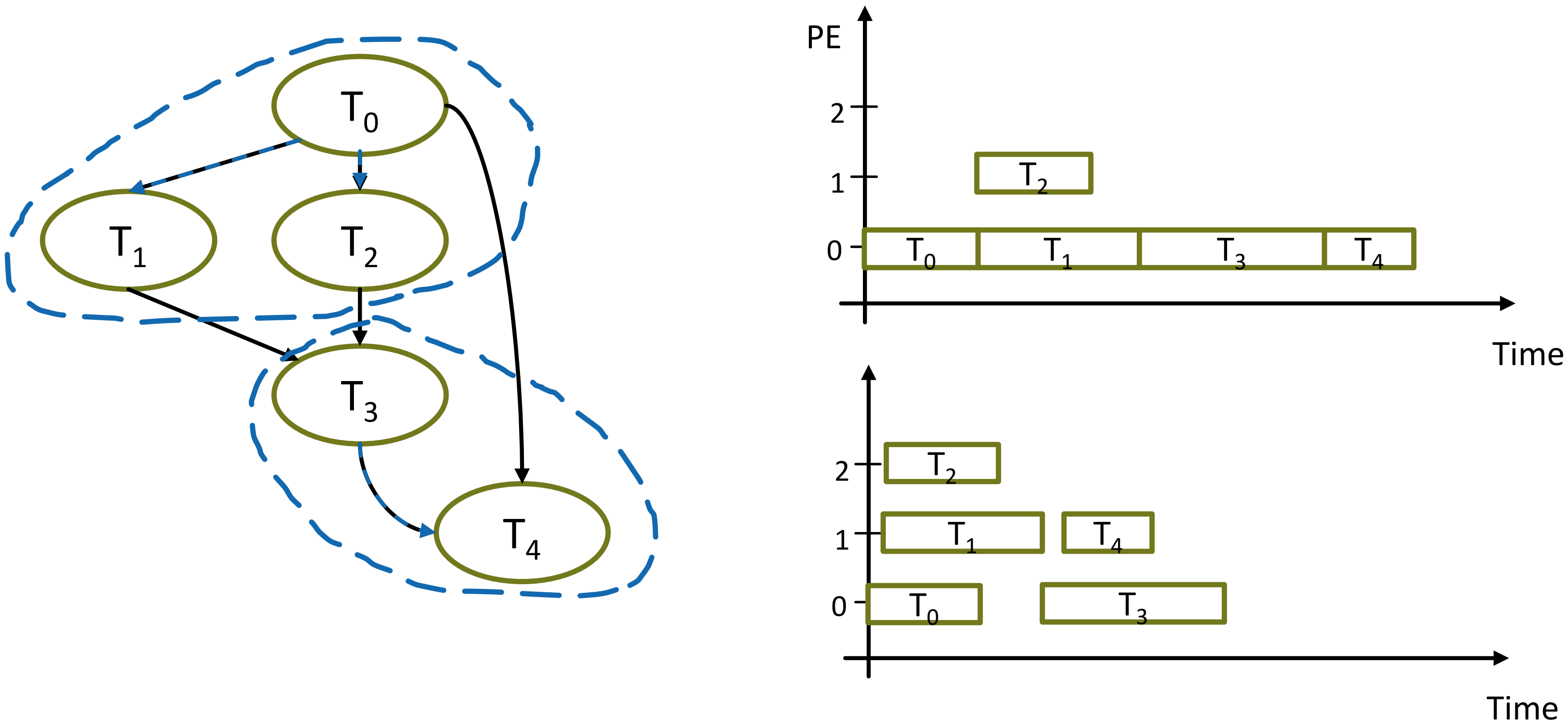
In traditional task scheduling, a task can start only when all the predecessors completed (compute-then-communicate)



Execution trace on 3 PEs

Streaming Scheduling

We want to enable streaming communications between tasks

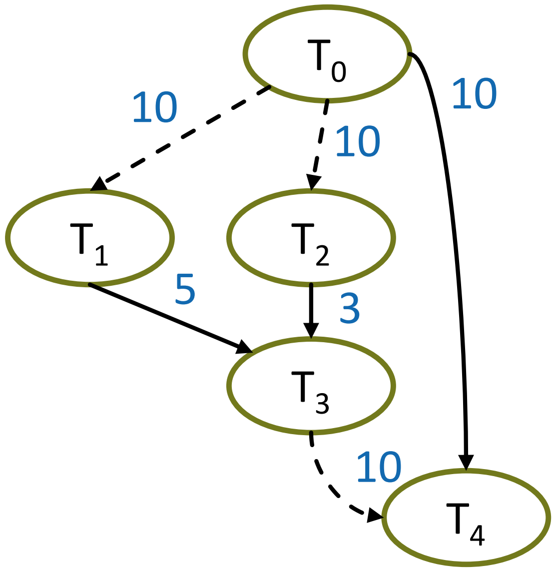


In this way we exploit spatial (pipeline) parallelism and reduce off-chip memory accesses

Solving this adds complexity to an already complicated problem:

- We need to understand whether it is better to stream or not, by building streaming blocks
- We need to understand how to schedule these streaming blocks

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

Streaming ⟷ **Scheduling**

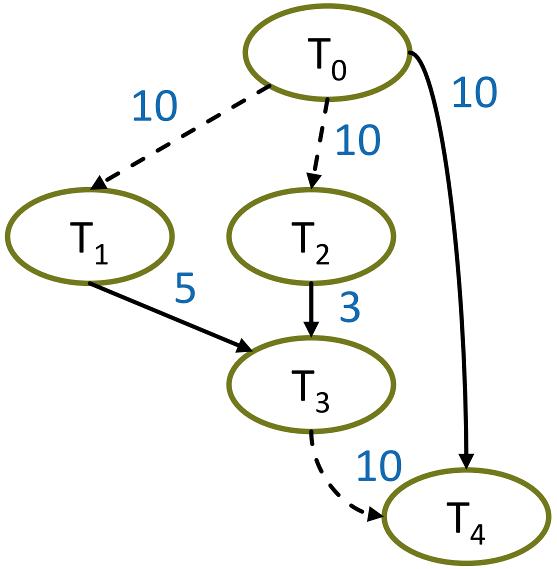
Streaming: decide which edge implements a pipelined communication and which not

Scheduling: schedule the tasks considering pipelined communication

We start by addressing the Scheduling problem

Scheduling

Let’s assume that we have the DAG with the type of communications



Algorithm 1. List-Scheduling

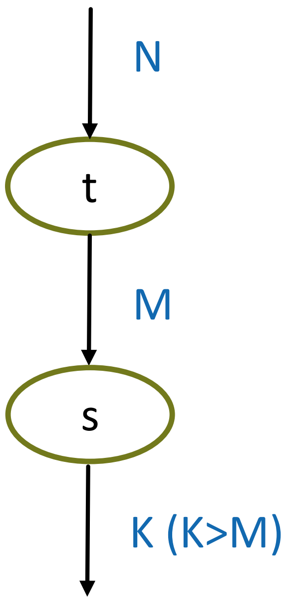
---

while *there are tasks to be scheduled* **do**
  Identify a highest priority task n (e.g., from a list);
  Choose a processor p for n;
  Schedule n on p at $est(n, p)$;
end

Candidates: HEFT, PEFT, … (something specific for homog. Computation TBD)

- take into account streaming (a children can start before its parent finishes)
- best-effort: streaming tasks are co-scheduled if it can do it, and it is useful, otherwise not
- Backpressure may be a problem

Tasks



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

$$T_t = \max\{N, M\}$$

Alternatively: we need additional input data to tell us how long does it takes to compute a task, what is its initiation interval, what is the latency …

A node may be slow down by a slower child. This affects its running time and the running time of the other children

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

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

TODO

- Find good heurstics and use them as reference
- Understand how to model backpressure
- Play with some toy example
- [Streaming]