VLSI 1 - Notes Week 8

Ruben Schenk, ruben.schenk@inf.ethz.ch

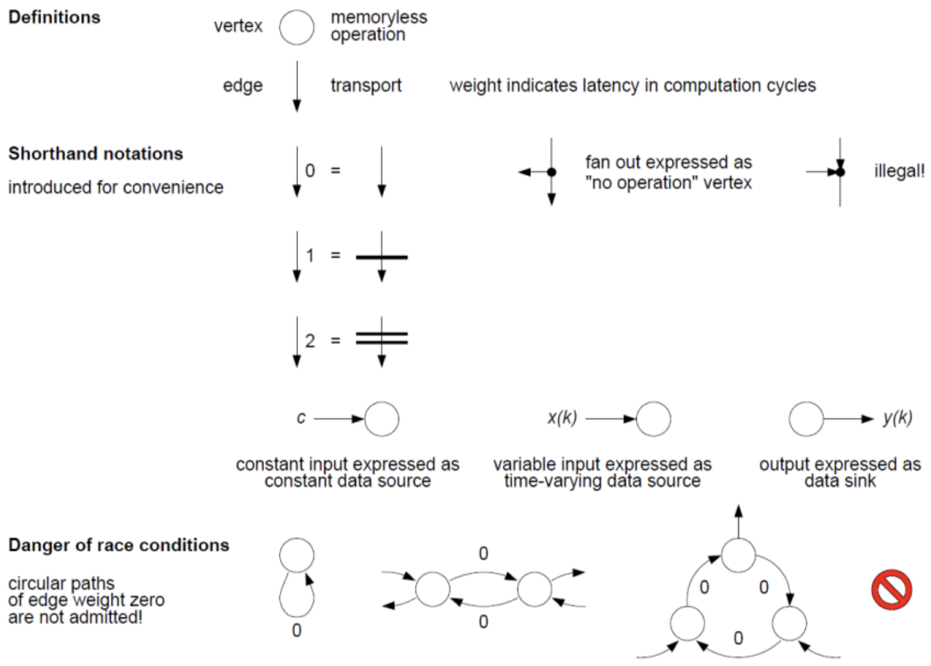
January 12, 2022

There is room for remodeling computations in two distinct domains:

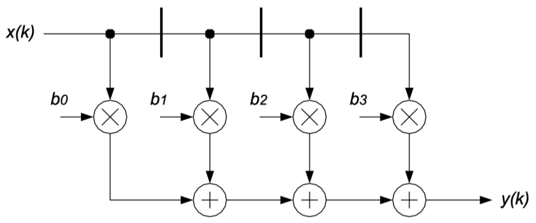
- Processing algorithm: Alternative choices in the algorithm domain. How to tailor an algorithm such as to cut the computational burden, to trim down memory requirements, and/or to speed up calculations without incurring unacceptable implementation losses?
- Hardware architecture: Equivalence transforms in the architectural domain. How to (re-)organize a computation such as to optimize throughput, circuit size, energy efficiency and overall costs while leaving the input-to-output relationship unchanged except, possibly, for latency?

In conclusion, it is always necessary to balance many contradicting requirements to arrive at a working and marketable embodiment of an algorithm.

Let us formalize how we describe an algorithm: with **data dependency graphs (DDG):**


Definitions
vertex
memoryless operation
edge
transport
weight indicates latency in computation cycles
Shorthand notations
introduced for convenience
0 =
fan out expressed as "no operation" vertex
illegal!
1 =
2 =
c
x(k)
y(k)
constant input expressed as constant data source
variable input expressed as time-varying data source
output expressed as data sink
Danger of race conditions
circular paths of edge weight zero are not admitted!
0
0
0
0
0
0

Example: Consider the following algorithm: $y(k) = \sum_{n=0}^{N=3} b_n x(k-n)$. Its representation as a DDG graph looks as follows:



We introduce different figures of merit for hardware architectures:

- *Cycles per data item* (Γ): number of computation cycles between releasing two subsequent data items.
- *Longest path delay* (t_{lp}): the laps of time required for data to propagate along the longest path. A circuit cannot function correctly unless $t_{lp} \leq T_{cp}$.
- *Time per data item* (T): the lapse time between releasing two subsequent data items. $T = \Gamma \cdot T_{cp} \geq \Gamma \cdot t_{lp}$
- *Data throughput* (Θ): $\Theta = \frac{1}{T} = \frac{f_{cp}}{\Gamma}$
- *Latency* (L): number of computation cycles from a data item entering a circuit until the pertaining result becomes available.
- *Circuit size* (A): expressed in mm^2 or GE (gate equivalent).
- *Size-time product* (AT): the hardware resources spent to obtain a given throughput. $AT = \frac{A}{\Theta}$.
- *Energy per data item* (E): the amount of energy dissipated for a given computation on a data item, e.g. $nJ/sample$. $E = PT$.

We can do some metrics for our previously shown DDG. We allow the following approximations: interconnected delays are neglected, delays of arithmetic operations are summed up, glitching is ignored. This leads to the following metrics:

- $A = 3A_{reg} + 4A_* + 3A_+$
- $\Gamma = 1$
- $t_{lp} = t_{reg} + t_* + 3t_+$
- $AT = (3A_{reg} + 4A_* + 3A_+)(t_{reg} + t_* + 3t_+)$
- $L = 0$
- $E = 3E_{reg} + 4E_* + 3R_+$

Let's quickly focus on the difference between *computation cycles* and *clock period:* A computation period T_{cp} is the time span that separates two consecutive computation cycles. During each computation cycle, fresh data emanates from a register, propagates through combinational circuitry before the result gets stored in the next analogous register. The *computation rate* denotes the inverse, i.e. $f_{cp} = \frac{1}{T_{cp}}$.

For all circuits that adhere to single-edge-triggered one-phase clocking, computation cycle and clock period are the *same:*

$$f_{cp} = f_{clk} \iff T_{cp} = T_{clk}$$

6.3 Equivalence Transforms For Combinational Computations

First, what do we mean by **combinational computation?** A computation is termed *combinational* if:

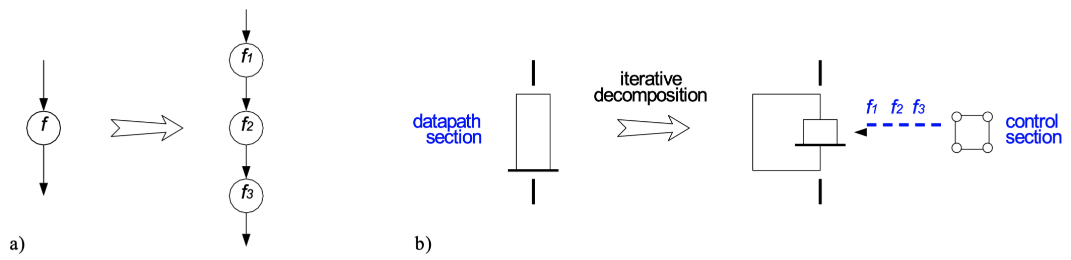
1. Result depends on the present arguments exclusively
2. All edges in the DDG have weight zero.
3. DDG is free of circular paths.

There are multiple options for implementing transforms at the architecture level:

- *Decomposing* function f into a sequence of subfunctions that get executed one after the other on the same hardware
- *Pipelining* of the functional unit for f to improve computation rate by cutting down combinational depth.
- *Replicating* the hardware for f and having all units work concurrently.

6.3.1 Iterative Decomposition

The idea of **iterative decomposition** is to perform a step-by-step execution by sharing one HW execution unit. The figure below shows the DDG (a) and a hardware configuration (b) for $d = 3$:



The performance and cost analysis of the iterative decomposition is as follows:

- $\frac{A_f}{d} + A_{reg} + A_{ctl} \leq A(d) \leq A_f + A_{reg} + A_{ctl}$
- $\Gamma(d) = d$
- $t_{lp}(d) \simeq \frac{t_f}{d} + t_{reg}$
- $L(d) = d$
- $E(d) \gtrless E_f + E_{reg}$

Iterative decomposition is attractive when a computation makes repetitive use of a single subfunction because a lot of area can be saved. It is unattractive when subfunctions are very disparate and, therefore, cannot be made to share much hardware resources.