Digital IC Design

Lecture 7: Energy-Efficient Architecture

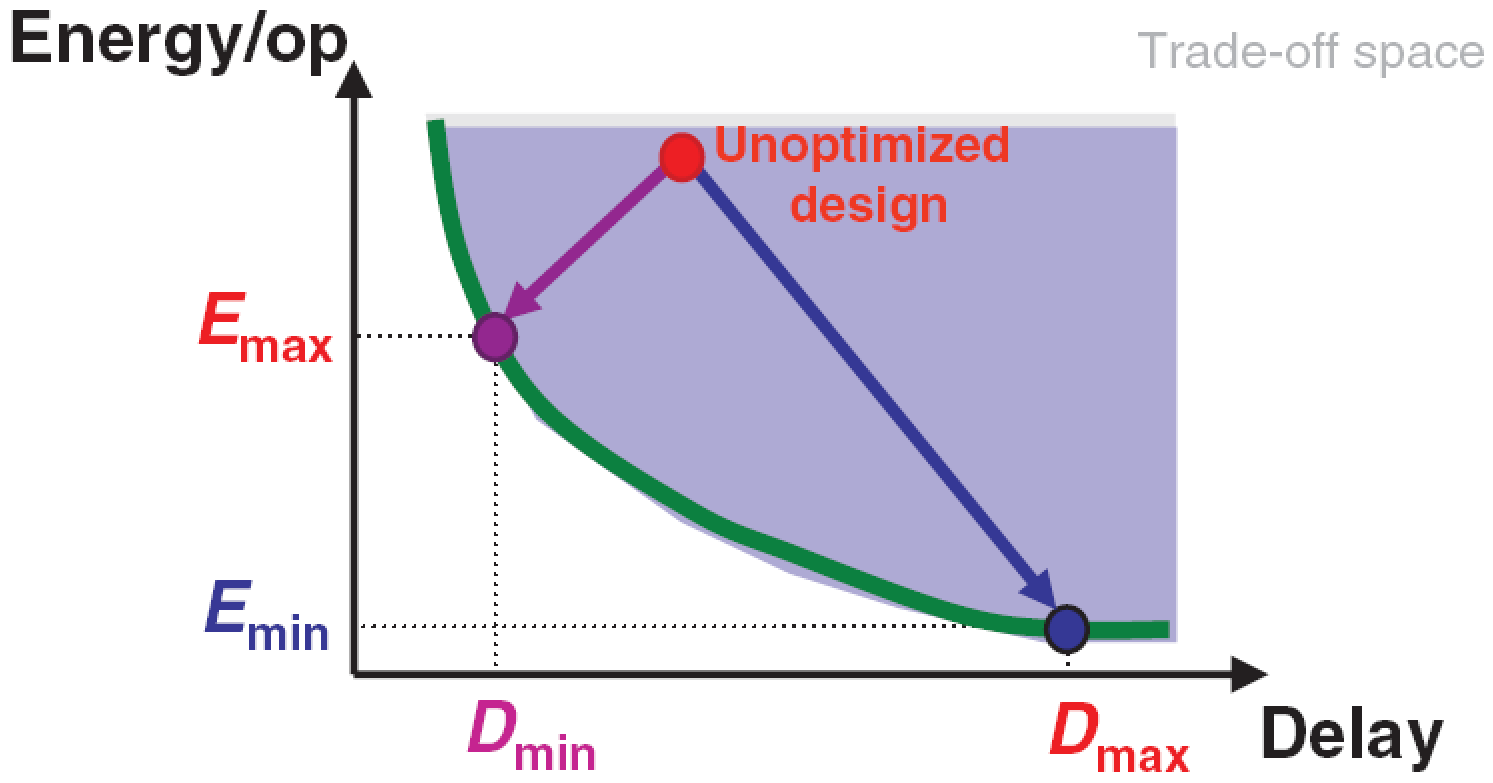
黃柏蒼 Po-Tsang (Bug) Huang
bughuang@nycu.edu.tw

International College of Semiconductor Technology
National Chiao Tung Yang Ming University

國立陽明交通大學
NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Energy–Delay Optimization

- Maximize throughput for given energy or Minimize energy for given throughput



Other important metrics: Area, Reliability, Reusability

Design Abstraction Stack Layers

- A very rich set of design parameters to consider! It helps to consider options in relation to their abstraction layer.

System/Application	Choice of algorithm
Software	Concurrency, Power Control
(Micro-)Architecture	Parallel, Pipeline, Configurable
Logic	logic family, standard cell, custom
Circuit	Sizing, supply, thresholds
Device	Bulk, SOI

Circuit Optimization Framework

Minimize **Energy** (V_{DD}, V_{TH}, W)

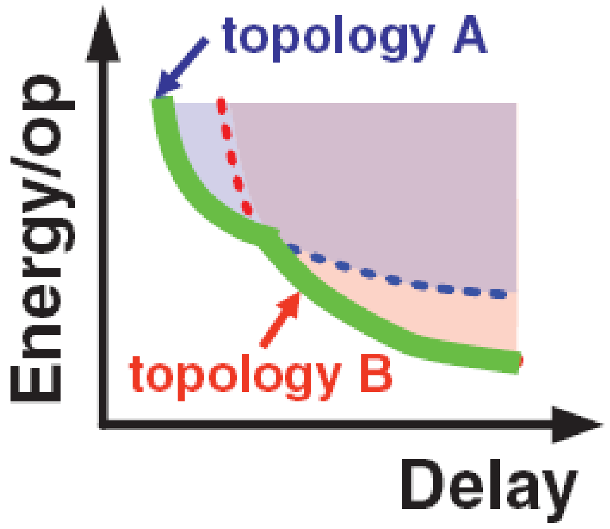
subject to **Delay** $(V_{DD}, V_{TH}, W) \leq D_{con}$

Constraints

$V_{DD\_min} < V_{DD} < V_{DD\_max}$

$V_{TH\_min} < V_{TH} < V_{TH\_max}$

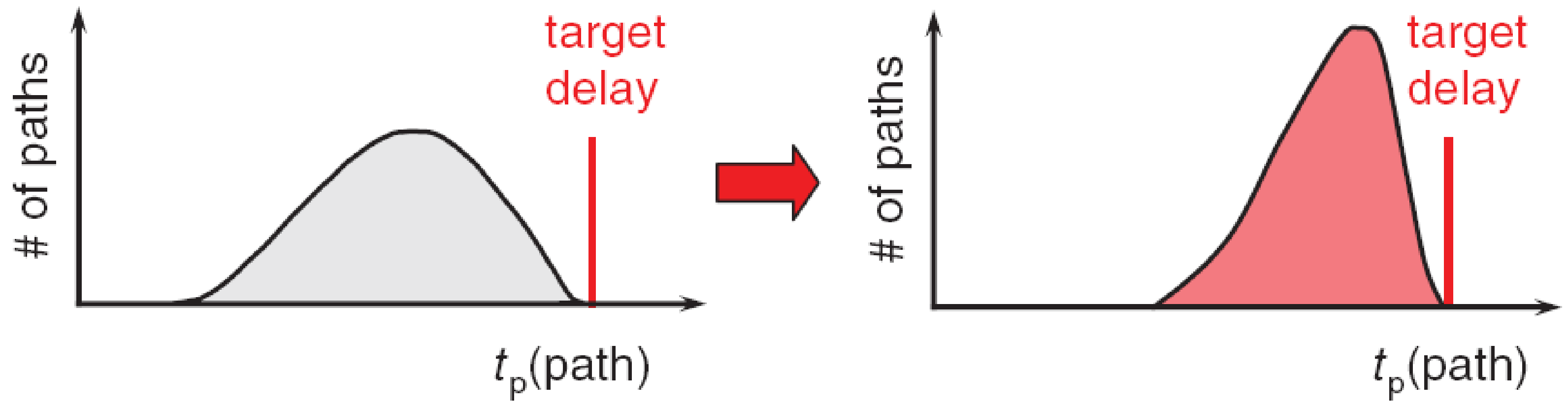
$W_{\_min} < W$



- Reference case
  - D_{min} Sizing @ (V_{DD_max}, V_{TH_ref})

Impact of Variations

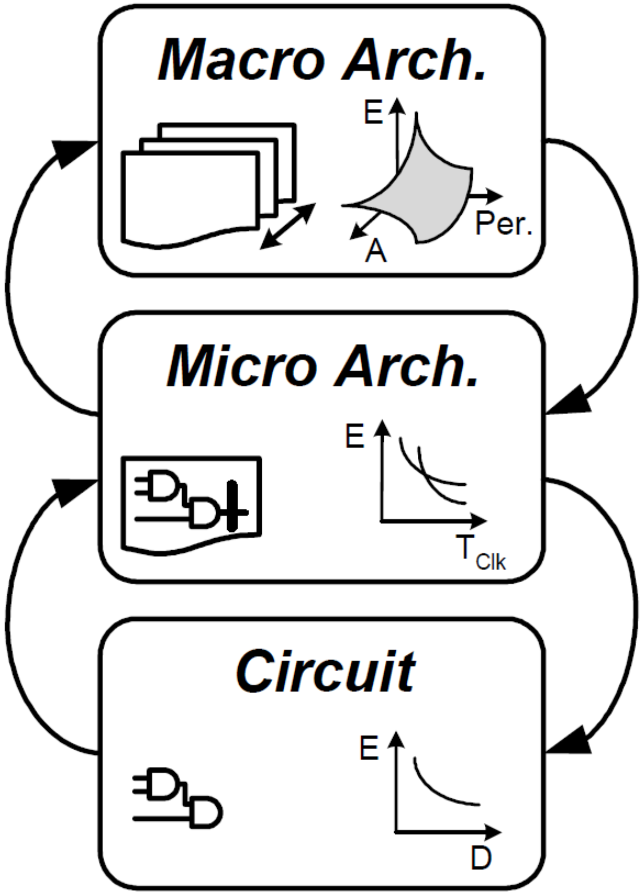
- Downsizing and/or lowering the supply on the critical path lowers the operating frequency
- Downsizing non-critical paths reduces energy for free, but
  - Narrows down the path–delay distribution
  - Increases impact of variations, impacts robustness

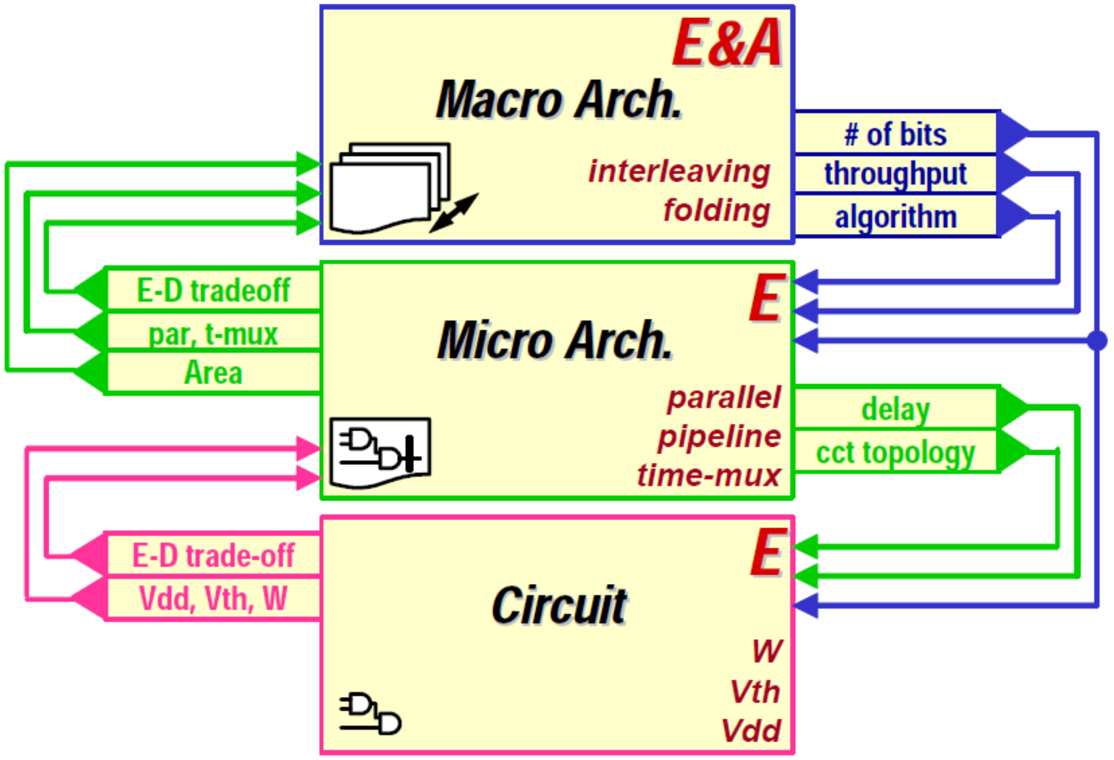


Optimizing Power at Architecture/System

- Optimizations at the architecture or system level can enable more effective power minimization at the circuit level (while maintaining performance), such as
  - Enabling a reduction in supply voltage
  - Reducing the effective switching capacitance for a given function (physical capacitance, activity)
  - Reducing the switching rates
  - Reducing leakage
- Optimizations at higher abstraction levels tend to have greater potential impact
  - While circuit techniques may yield improvements in the 10–50% range, architecture and algorithm optimizations have reported power reduction by orders of magnitude

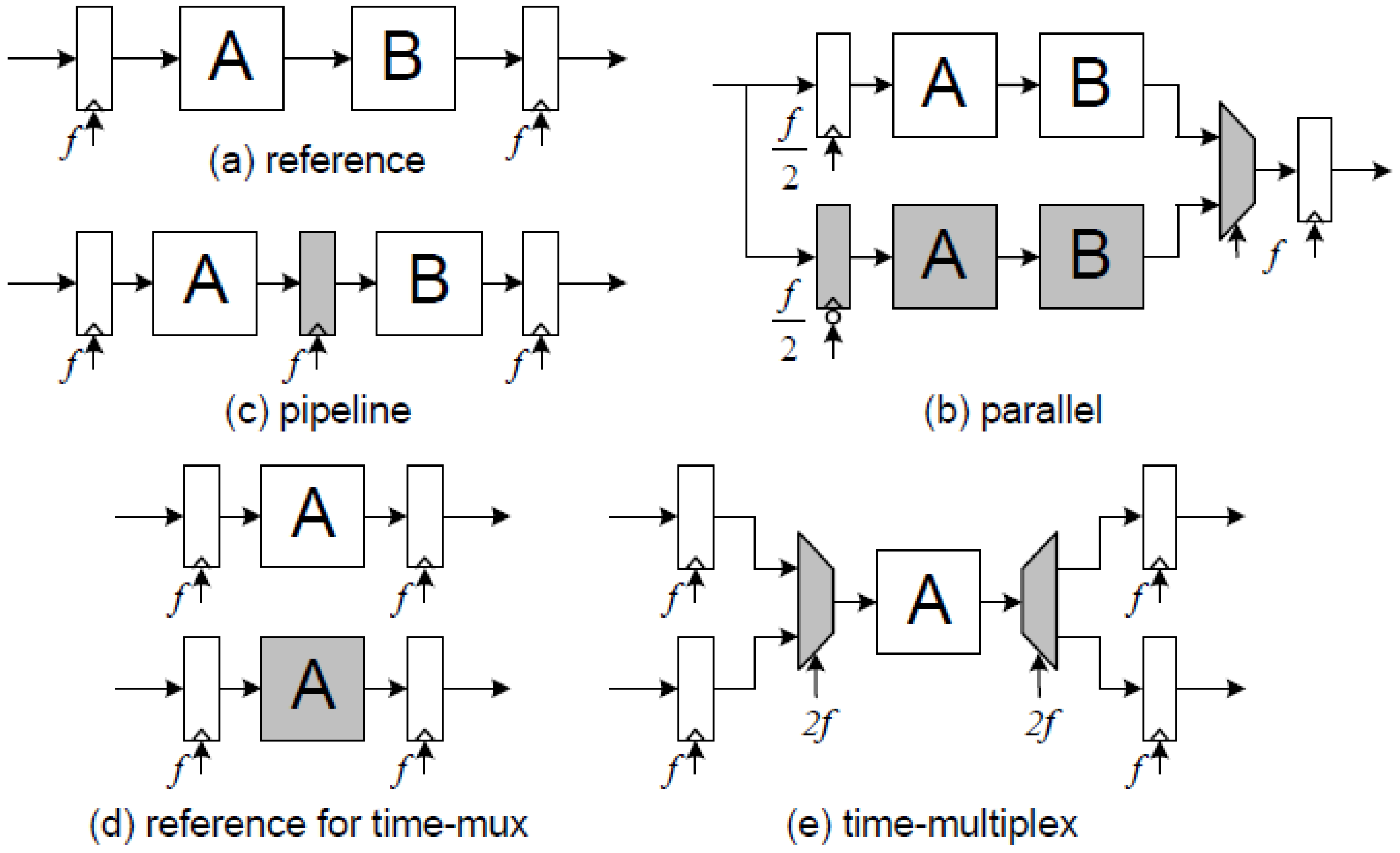
Cross-Layer Optimization





Basic Micro-Architecture Techniques

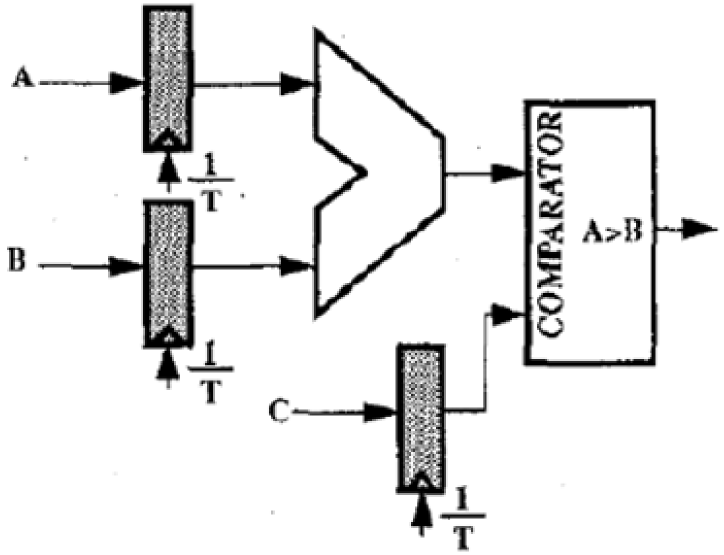
- Parallelism, pipeline, time-multiplexing

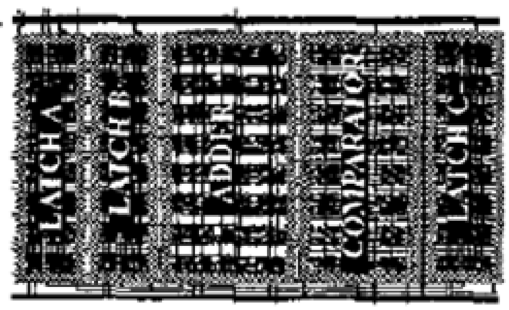


Reference Datapath

- Critical-path delay
  - Total capacitance being switching = C_{ref}
  - VDD = V_{DD_ref}
  - Power for reference datapath

$$P_{ref} = f_{ref} \cdot C_{ref} \cdot V_{DD\_ref}{}^2$$





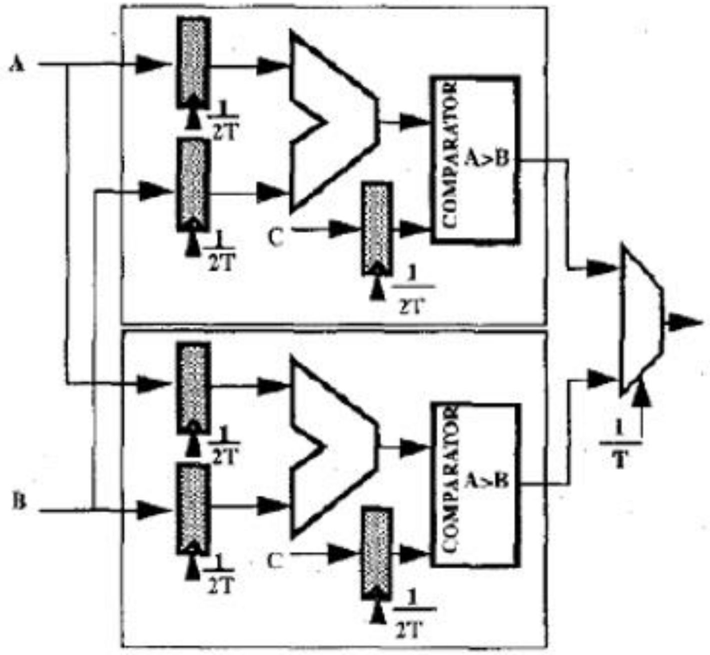
Area = 636 x 833 μ^2

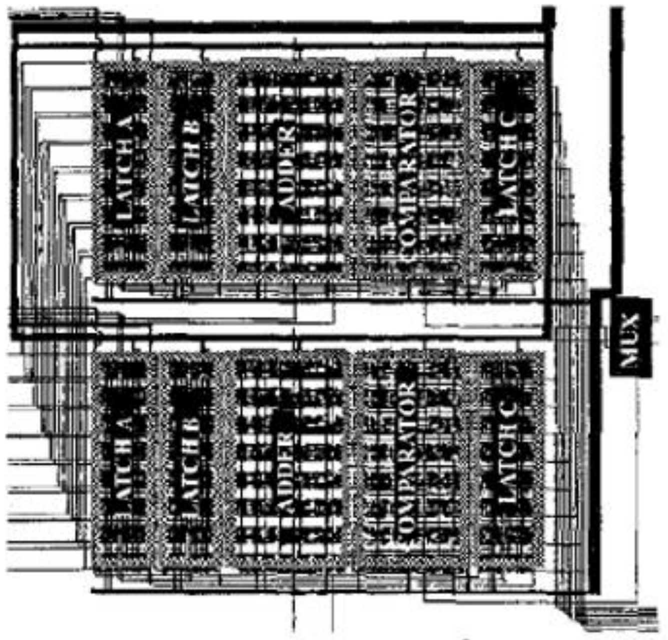
Concurrency: trading off frequency for area to reduce power

Parallel Datapath Architecture

- The clock rate of a parallel datapath can be reduced by half with the same throughput
  - $f_{par} = f_{ref}/2$
  - $V_{DD\_par} = V_{DD\_ref}/1.7$
  - $C_{par} = 2.15 * C_{ref}$

$\Rightarrow$ $P_{par} \approx 0.36 P_{ref}$

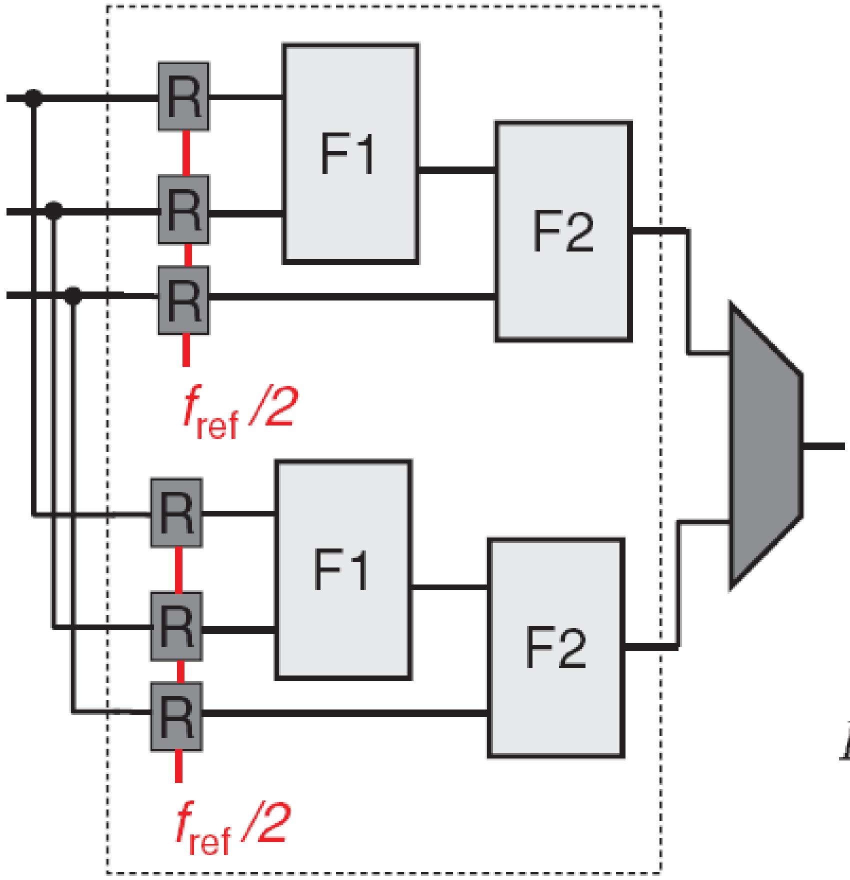




Area = 1476 x 1219 μ^2

A Parallel Implementation

- Running slower reduces required supply voltage yields quadratic reduction in power

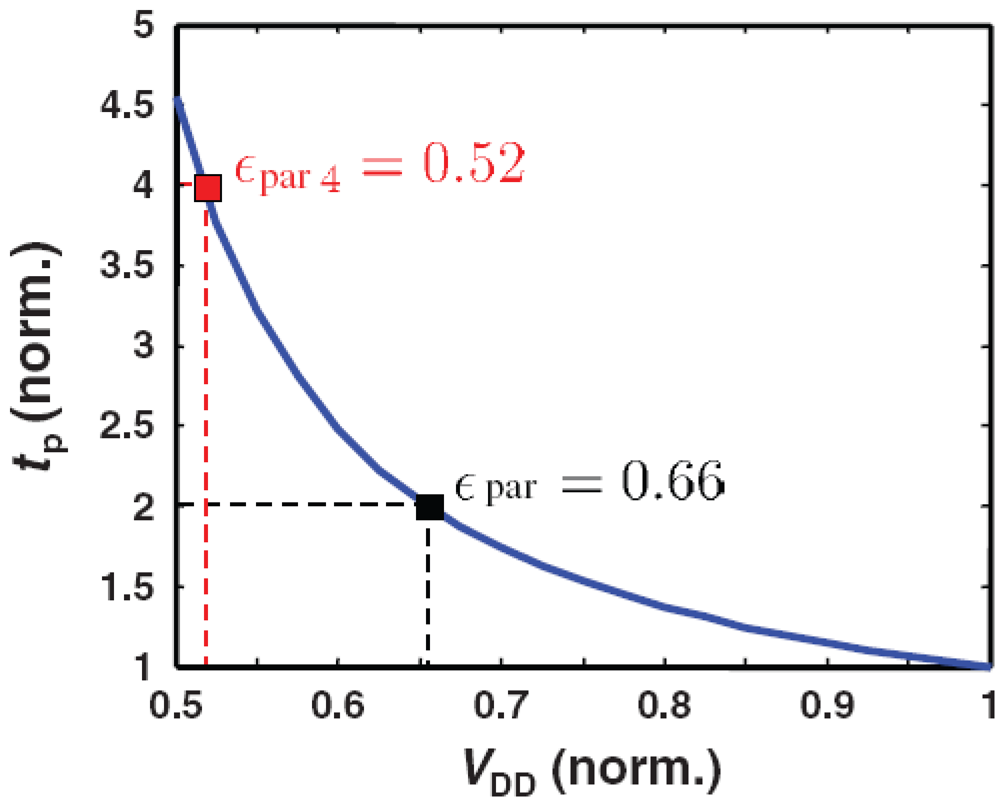


$$f_{par} = f_{ref}/2$$
$$C_{par} = (2 + ov_{par}) \cdot C_{ref}$$
$$V_{DD\,par} = \epsilon_{par} / . V_{DD\,ref}$$

Almost cancels

$$P_{par} = \epsilon^2_{par} \cdot \left(\frac{2 + ov_{par}}{2}\right) \cdot P_{ref}$$

Example for Parallelism in 90nm

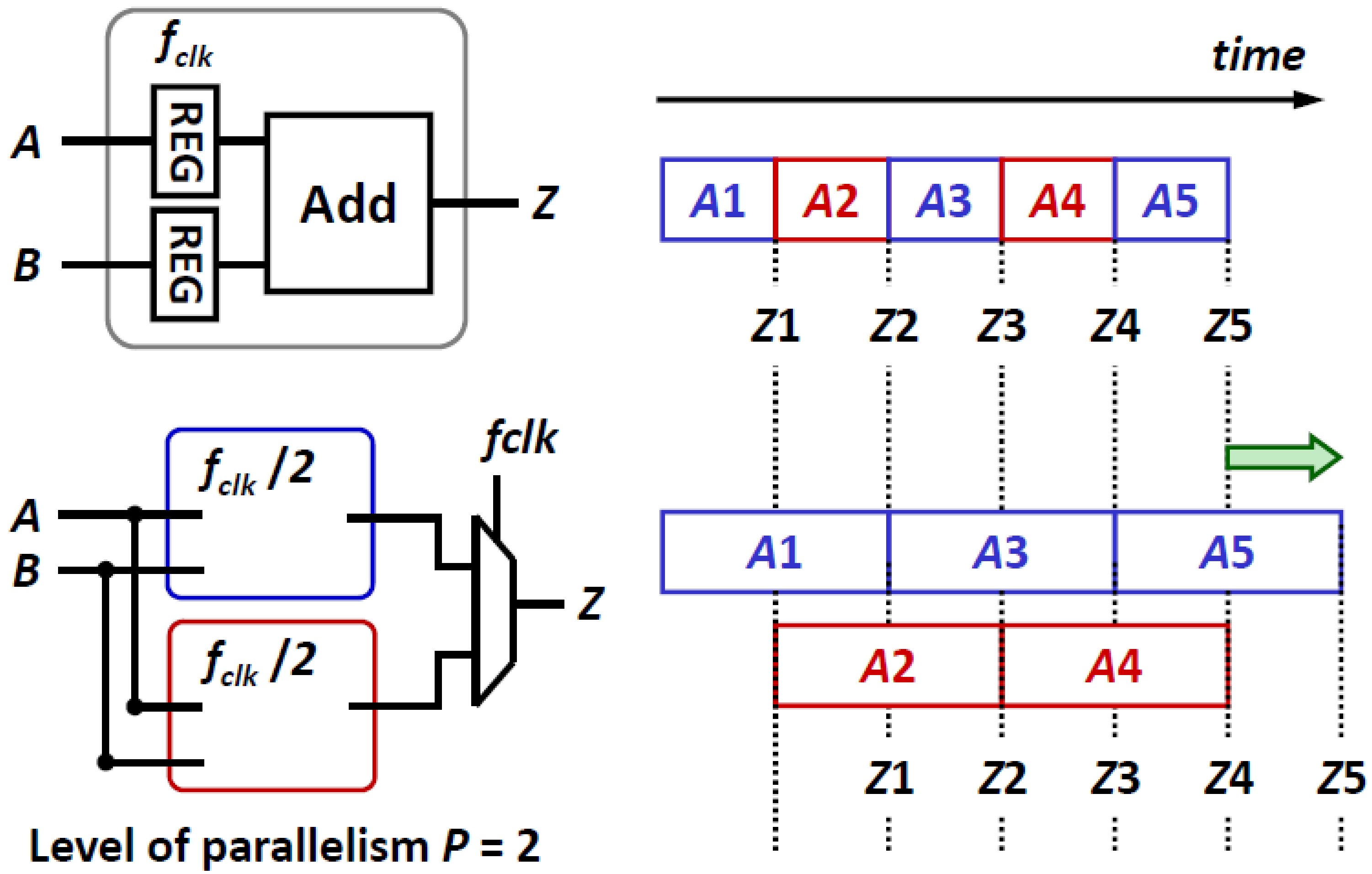


Assuming
$ov_{\text{par}} = 7.5\%$

$$P_{\text{par}} = 0.66^2 \cdot \frac{2.15}{2} \cdot P_{\text{ref}} = 0.47 P_{\text{ref}}$$

$$P_{\text{par 4}} = 0.52^2 \cdot \frac{4.3}{4} \cdot P_{\text{ref}} = 0.29 P_{\text{ref}}$$

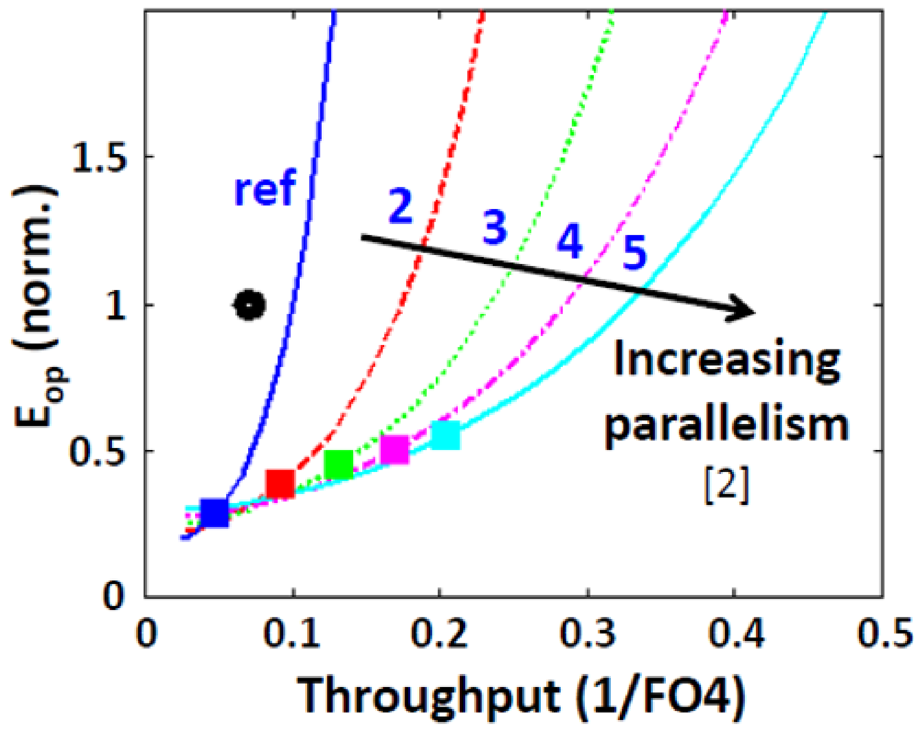
Parallelism Adds Latency



Level of parallelism $P = 2$

Increasing Level of Parallelism

- Improve throughput for the same energy
- Improve energy for the same throughput

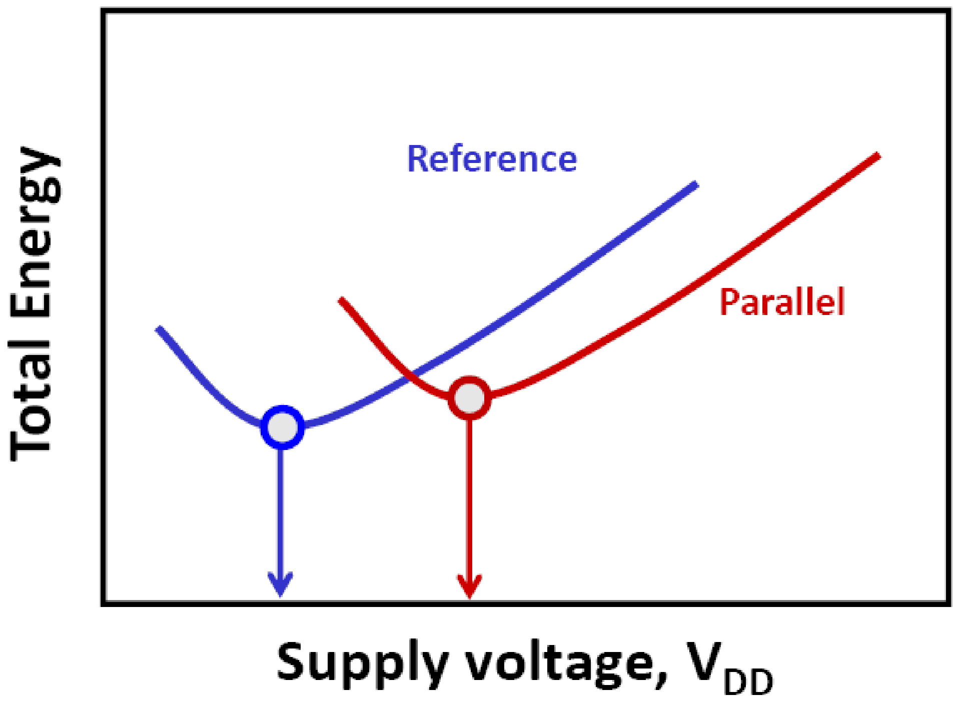


Area: $A_{par} \approx N * A_{ref}$

Cost: increased area

More Parallelism is NOT Always Better

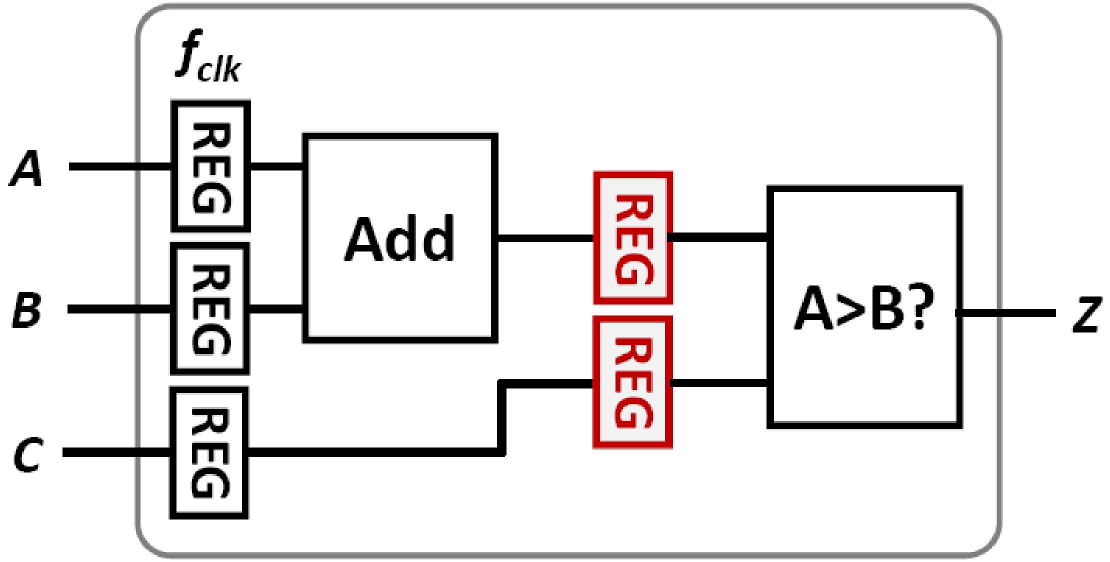
- Leakage and overhead start to dominate at high levels of parallelism, causing minimum energy (dot) to increase
- Optimum voltage also increases with parallelism



$$E_{total} = E_{SW} + N \times E_{leak} + E_{overhead}$$

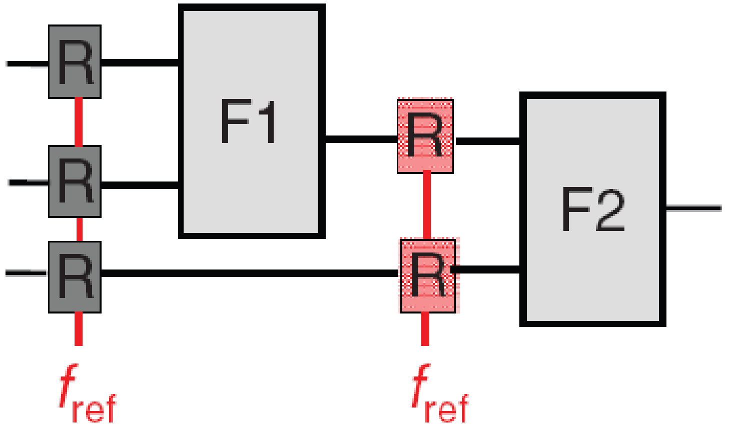
Pipelined Datapath Architecure

- Critical-path delay → max (t_{adder} , $t_{comparator}$)
  - ◆ Keeping clock rate constant: $f_{pipe} = f_{ref}$
  - ◆ Voltage can be dropped → $V_{DD\_pipe} = V_{DD\_ref} / 1.7$
  - ◆ Capacitance slightly higher: $C_{pipe} = 1.15 \cdot C_{ref}$
  - ◆ $P_{pipe} = f_{ref} \cdot (1.15 \cdot C_{ref}) \cdot (V_{DD\_ref} / 1.7)^2 \approx 0.39 \cdot P_{ref}$



A Pipelined Implementation

- Shallower logic reduces required supply voltage



$$f_{pipe} = f_{ref}$$
$$C_{pipe} = (1+ov_{pipe}) \cdot C_{ref}$$
$$V_{DDpipe} = \varepsilon_{pipe} \cdot V_{DD\,ref}$$

$$P_{pipe} = \varepsilon^2_{pipe} \cdot (1+ov_{pipe}) \cdot P_{ref}$$

Assuming
ov_{pipe} = 10%

$$P_{pipe} = 0.66^2 .1.1.\ P_{ref} = 0.48 P_{ref}$$
$$P_{pipe4} = 0.52^2 .1.1.\ P_{ref} = 0.29 P_{ref}$$

Architecture Summary (Simple Datapath)

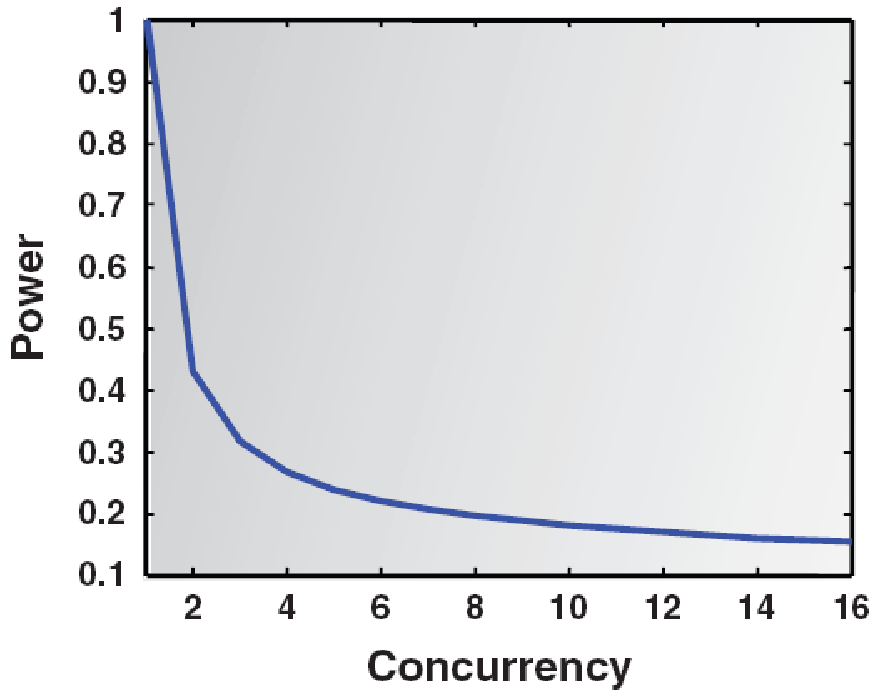
- Pipelining and parallelism relax performance of a datapath, which allows voltage reduction and results in power savings
- Pipelining has less are overhead than parallelism, but is generally harder to implement (involves finding convenient logic cut-sets)

Architecture type	Voltage	Area	Power
Reference datapath (no pipelining of parallelism)	5 V	1	1
Pipelined datapath	2.9 V	1.3	0.39
Parallel datapath	2.9 V	3.4	0.36
Pipeline-Parallel	2.0 V	3.7	0.2

Increasing Use of Concurrency

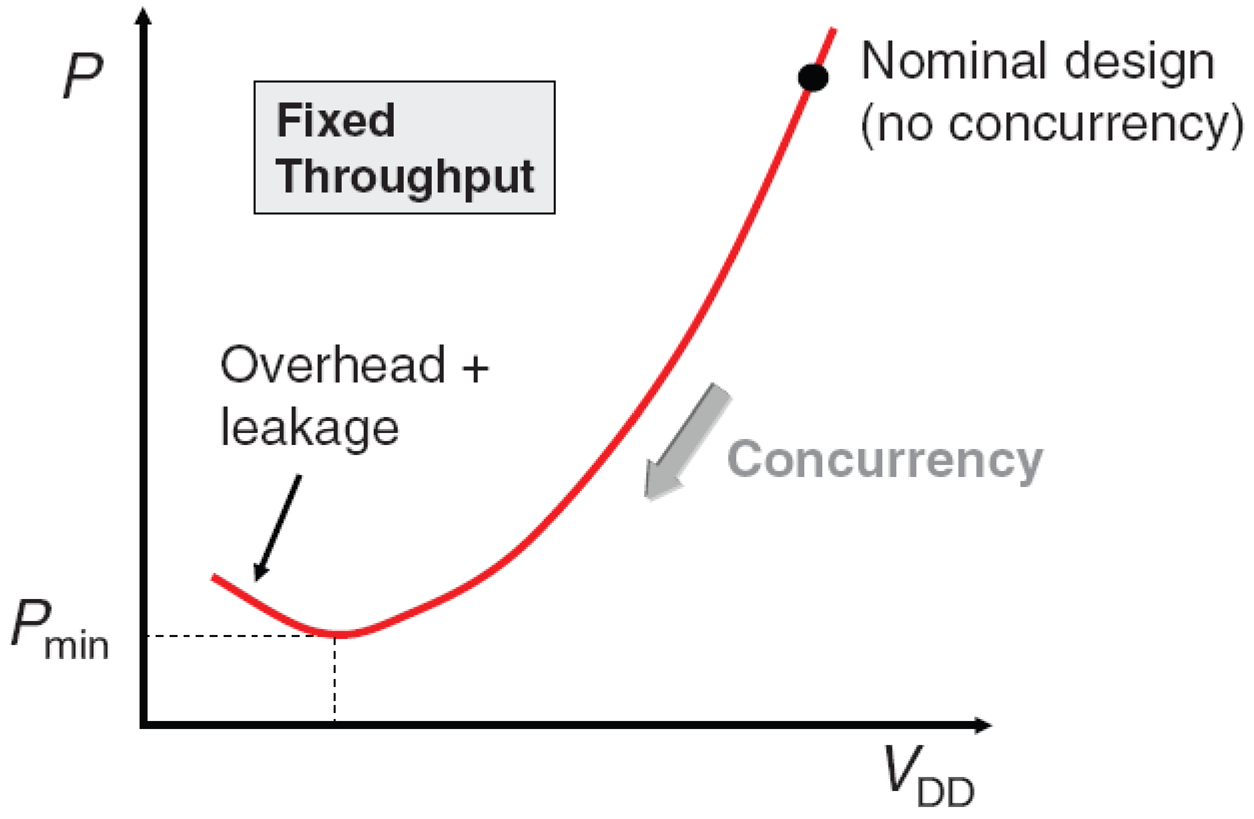
- Can combine parallelism and pipelining to drive *supply voltage* down
- But close to process threshold, overhead of excessive concurrency starts to dominate

Assuming constant % overhead

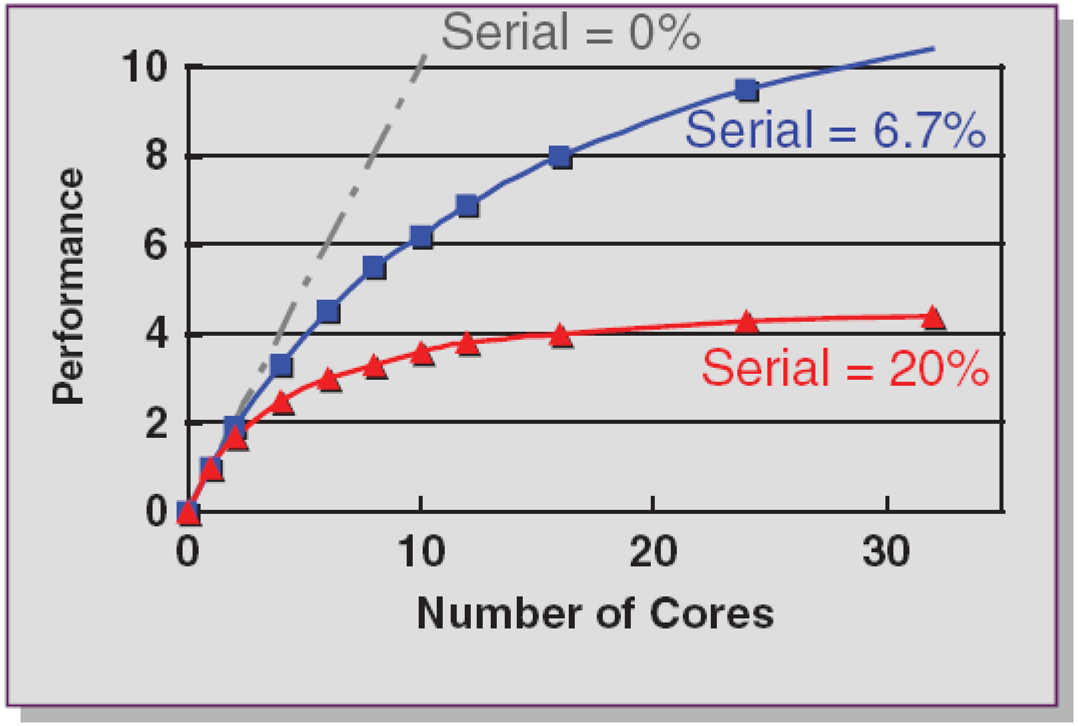


Increasing Use of Concurrency

- Must consider leakage



The Quest for Concurrency

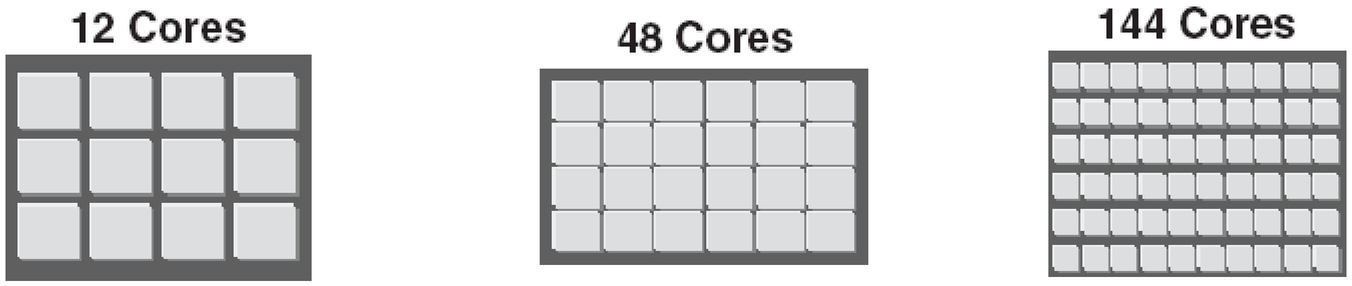


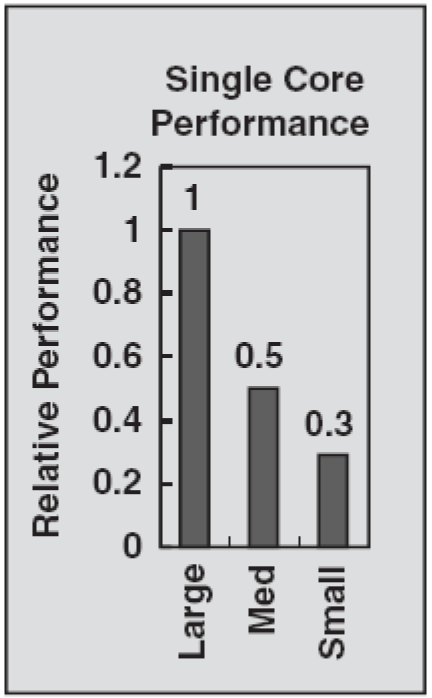
Amdahl's Law:

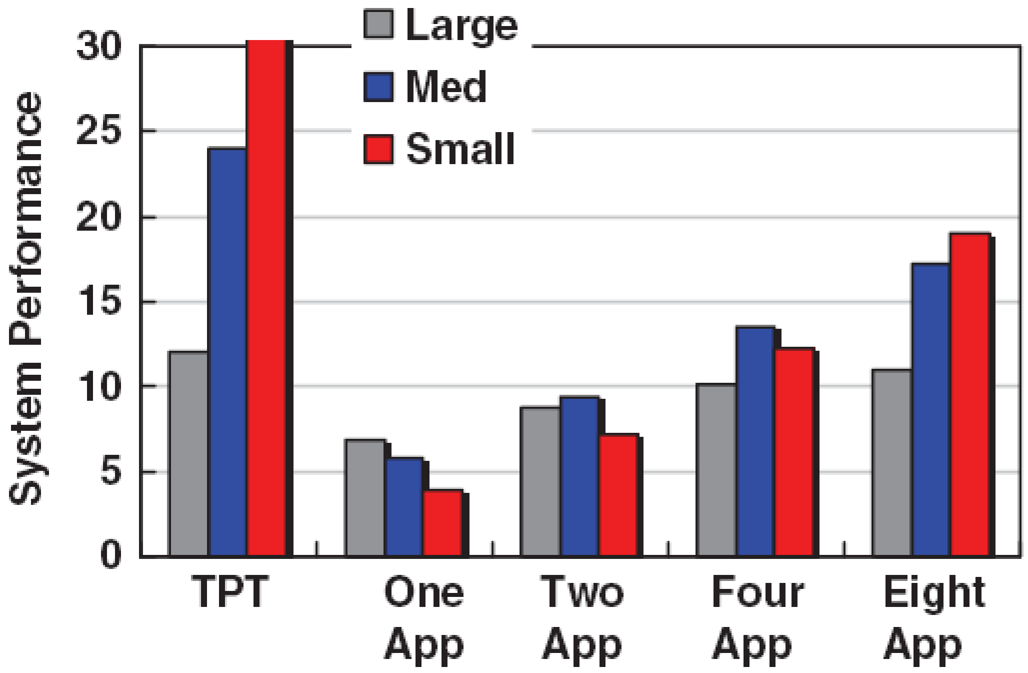
$$Speed\text{-}up = \frac{1}{Serial + \frac{1 - Serial}{N}}$$

Peroforamance/Area in Multi-Core

13 mm,**100 W**, 48 MB Cache, 4B Transistors, in 22 nm

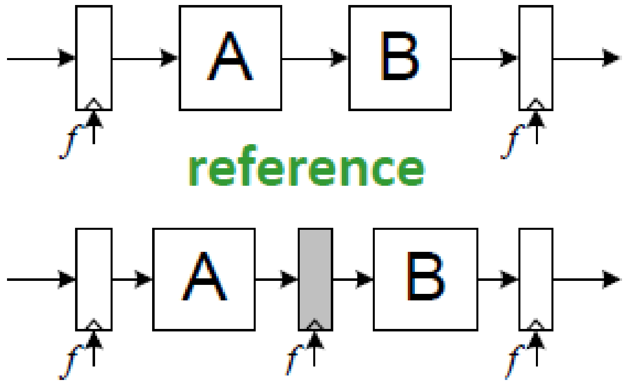


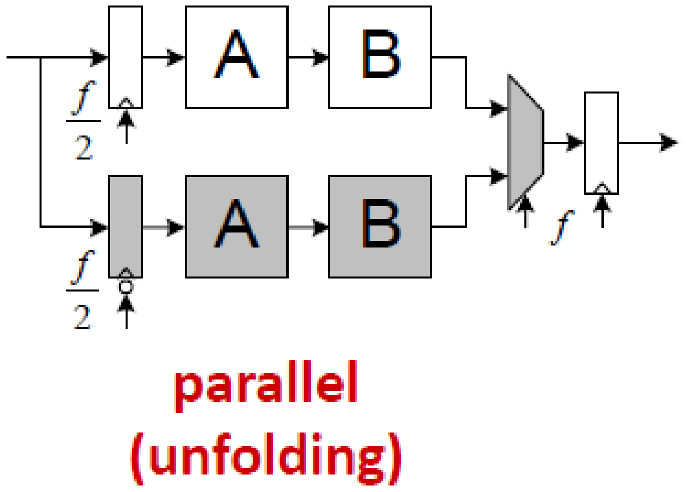


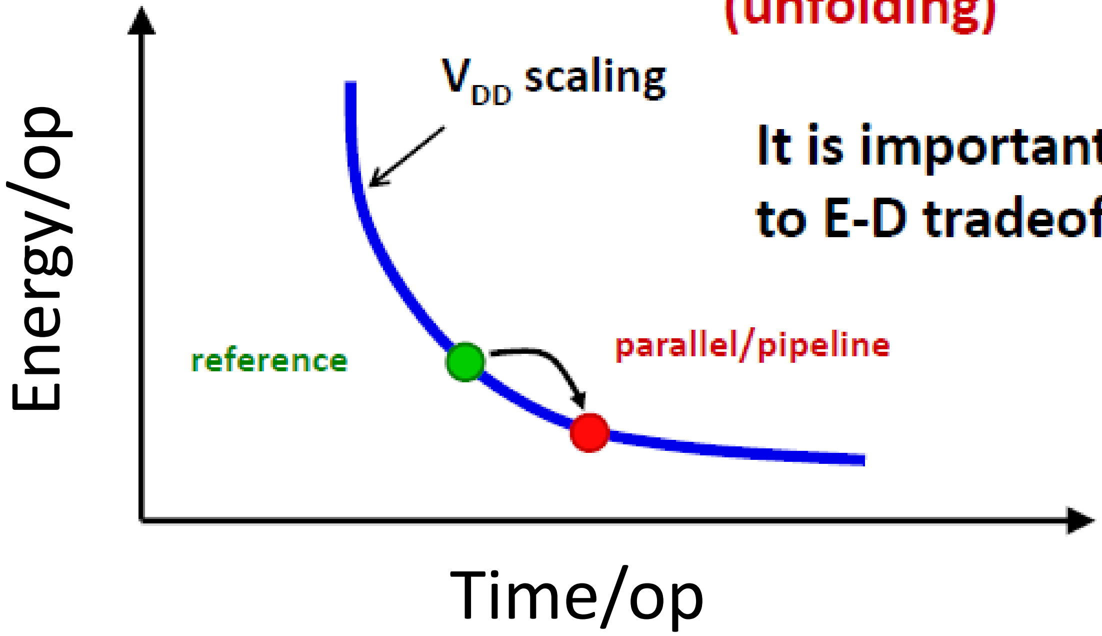


[Courtesy : S. Borkar, Intel, 2006]

Parallelism and Pipelining

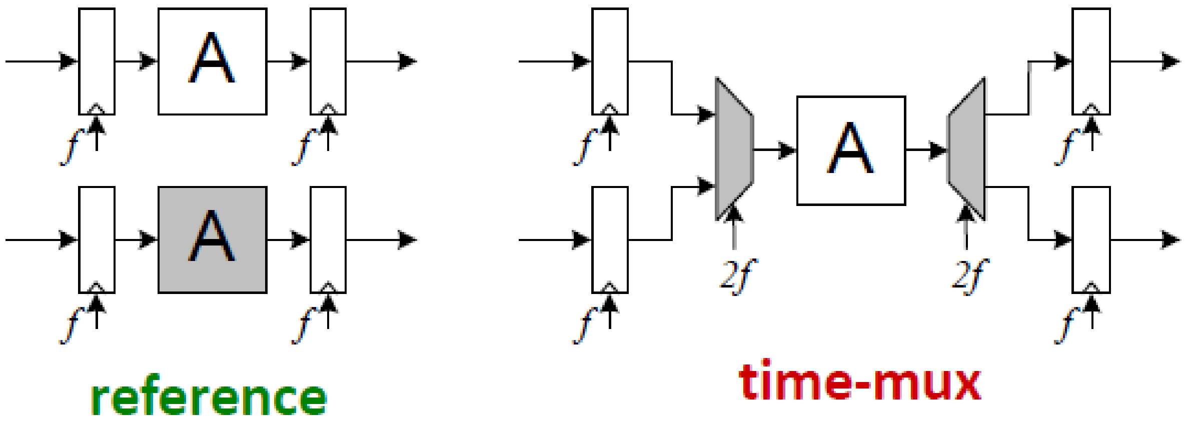


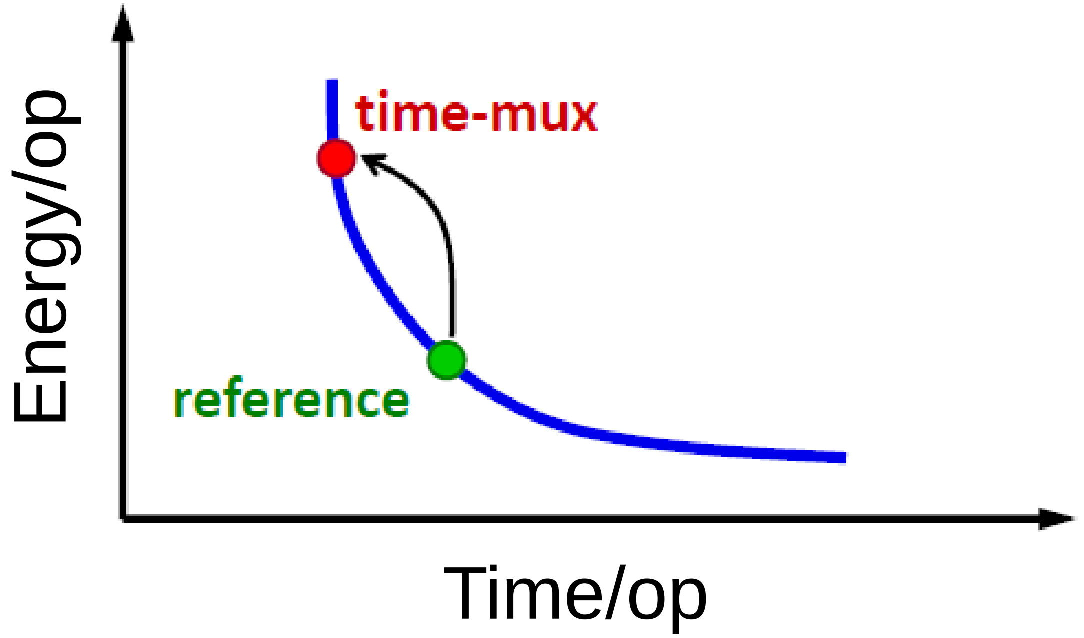




It is important to link to E-D tradeoff

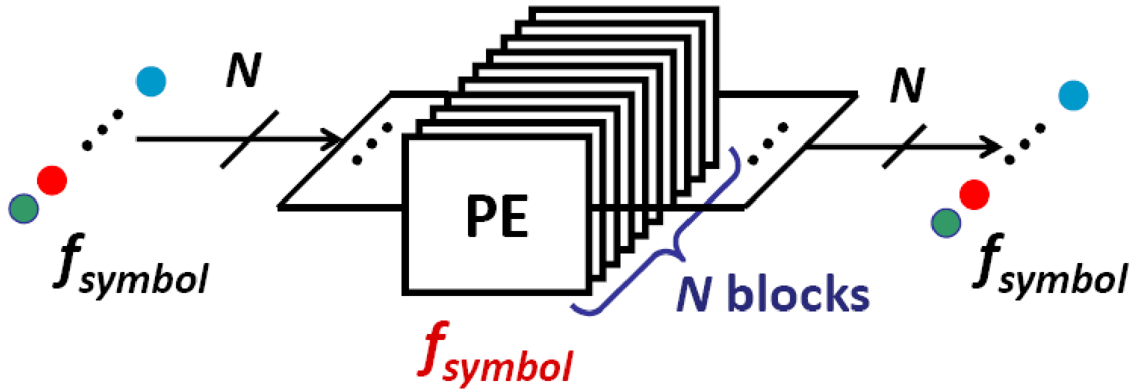
Time Multiplexing





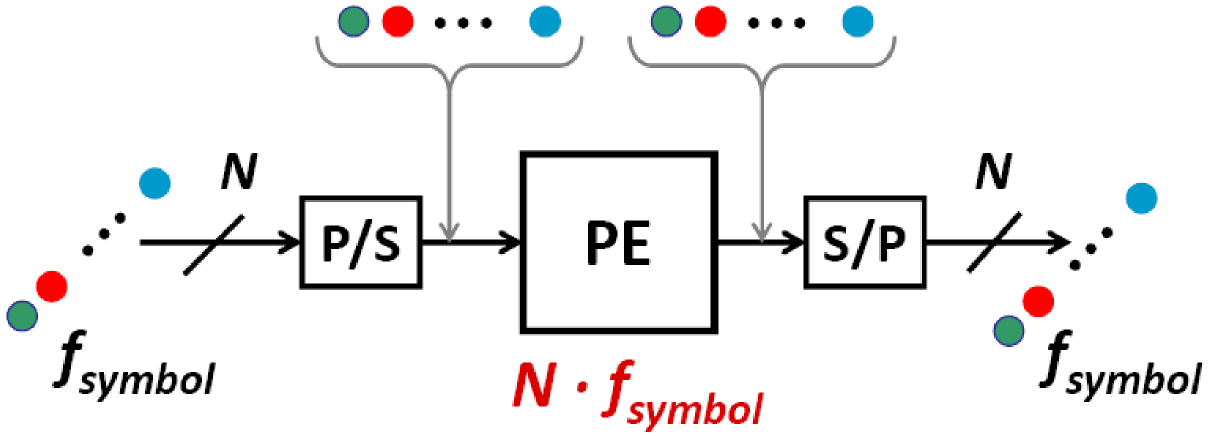
Data-Stream Interleaving

PE = recursive operation (feedback)



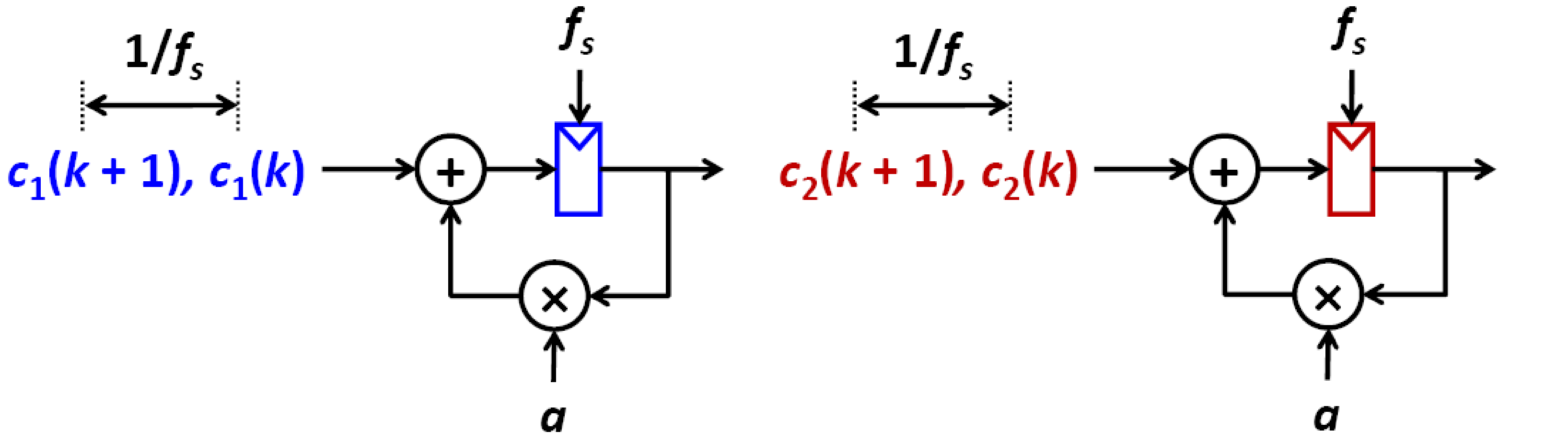
Large area

Interleaving Architecture

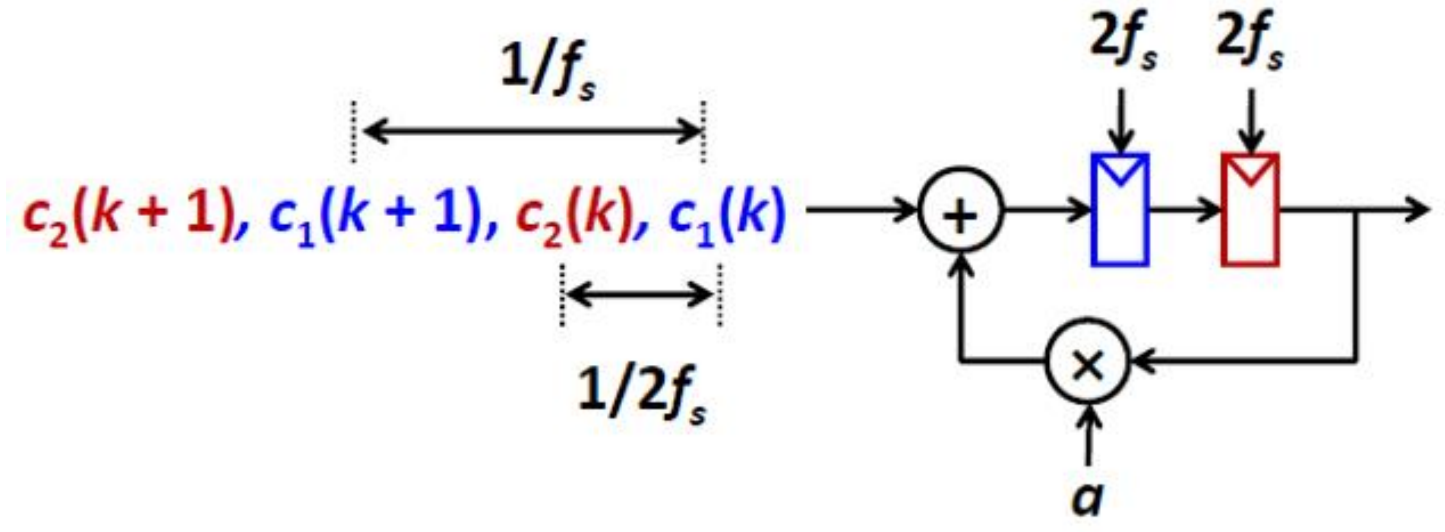


- ➢ Reduced area
- ➢ P/S overhead
- ➢ Pipelined

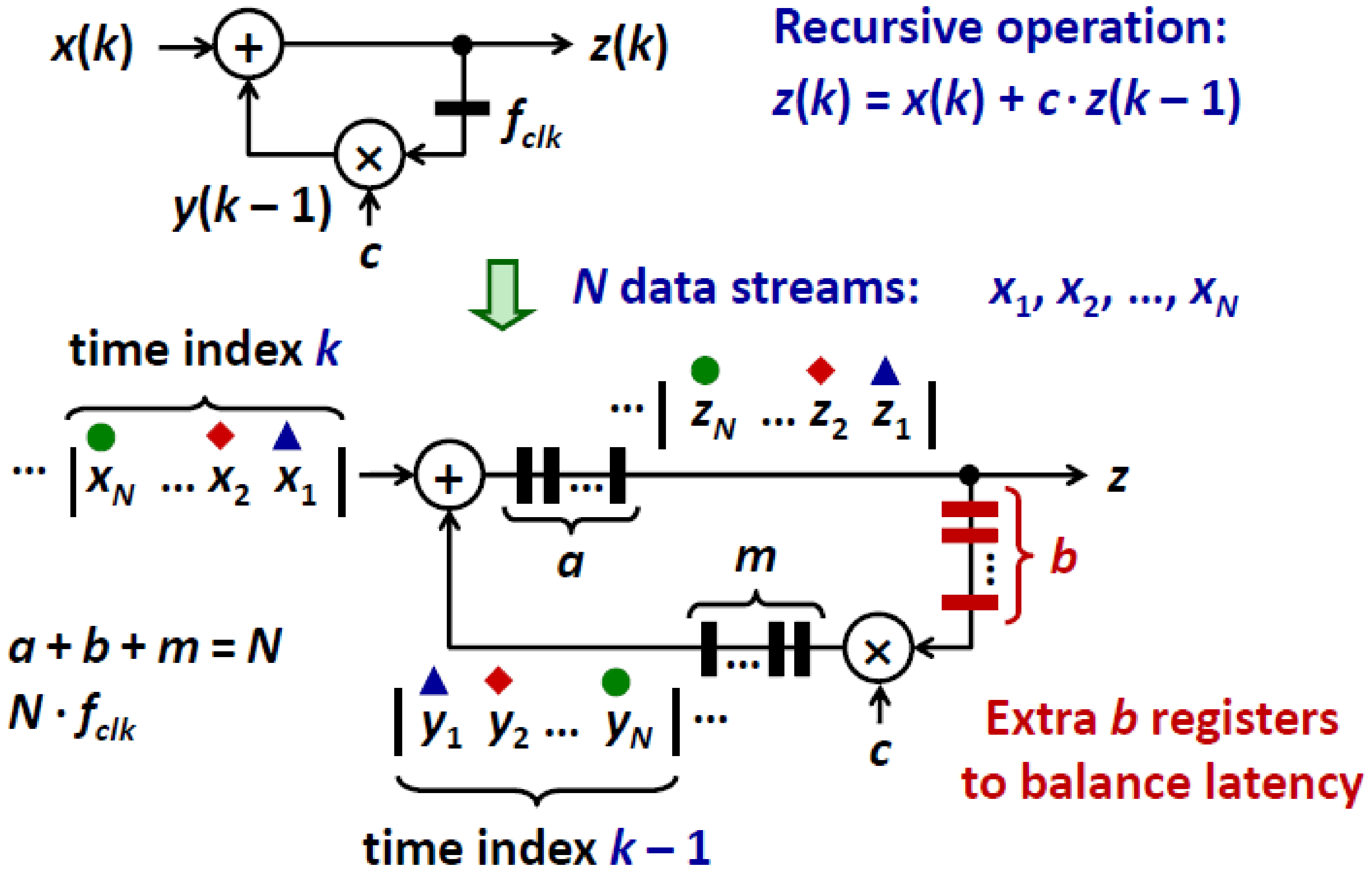
PE Performs Recursive Operation



- **Interleave = up-sample & pipeline**



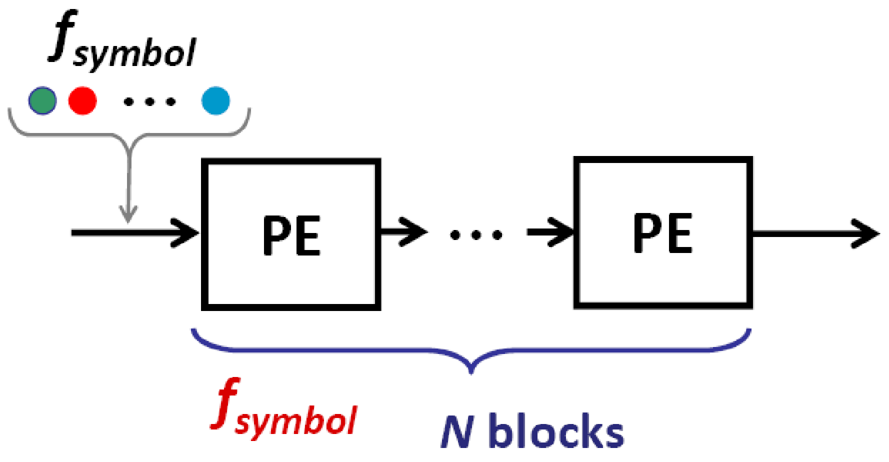
Data-Stream Interleaving Example



$x(k)$
$z(k)$
f_{clk}
$y(k-1)$
c
Recursive operation:
$z(k) = x(k) + c \cdot z(k-1)$
N data streams: $x_1, x_2, \ldots, x_N$
time index k
$\ldots \mid x_N \ \ldots \ x_2 \ x_1 \mid$
$\ldots \mid z_N \ \ldots \ z_2 \ z_1 \mid$
z
a
m
b
$\mid y_1 \ y_2 \ \ldots \ y_N \mid \ldots$
c
time index $k-1$
$a + b + m = N$
$N \cdot f_{clk}$
Extra b registers to balance latency

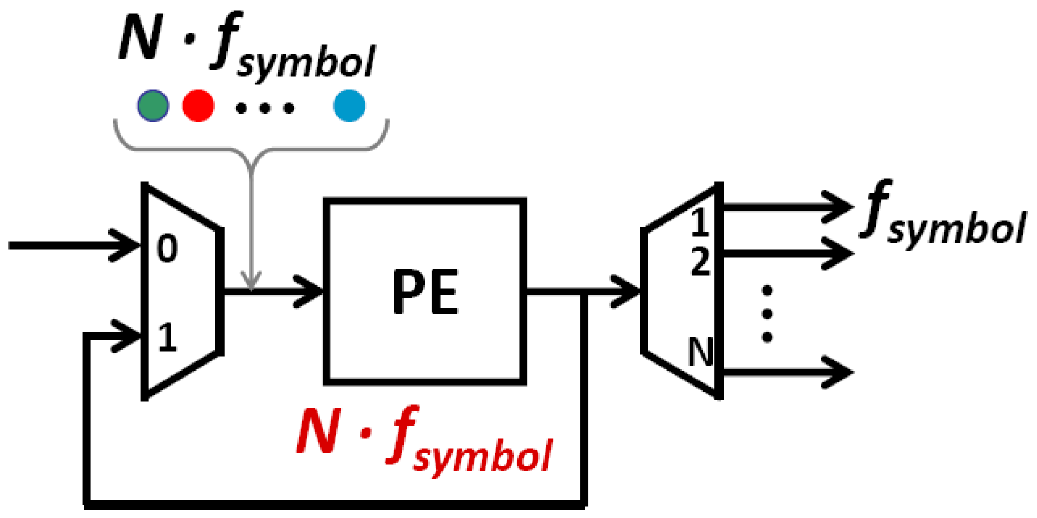
Folding

PE = recursive operation



- PEs in serial
- Large area

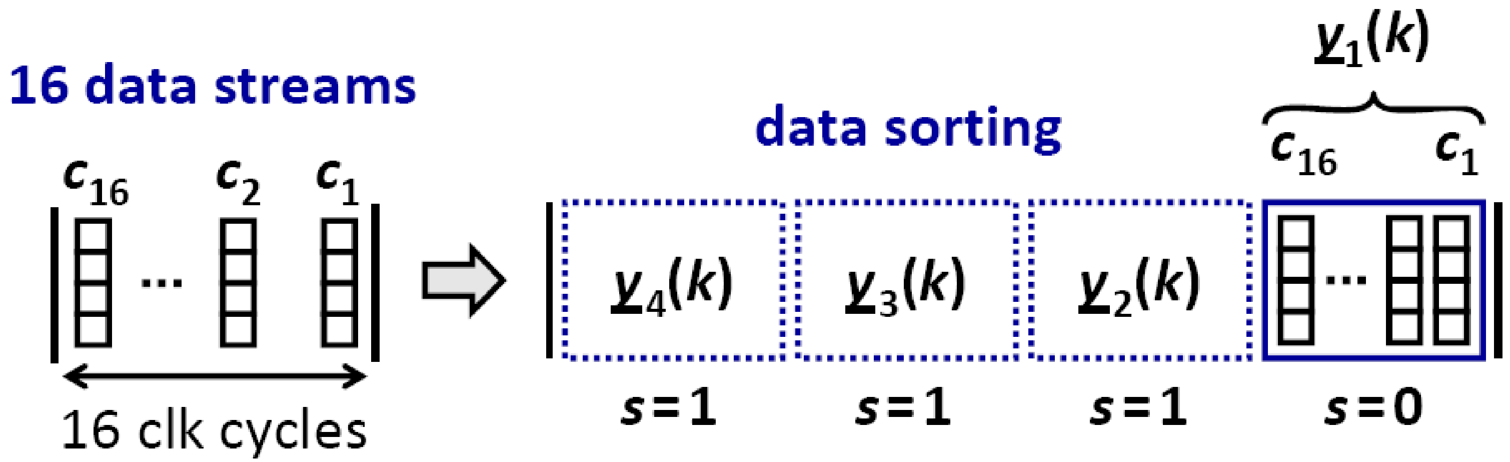
Folded Architecture

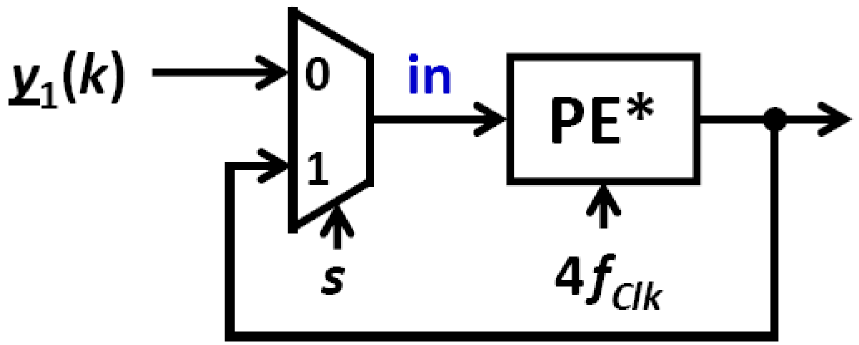


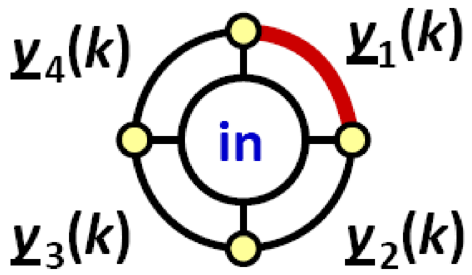
- Reduced area
- P/S overhead
- Pipelined

Folding Example

- **Folding = up-sampling & pipelining**
  - Reduced area (shared datapath logic)

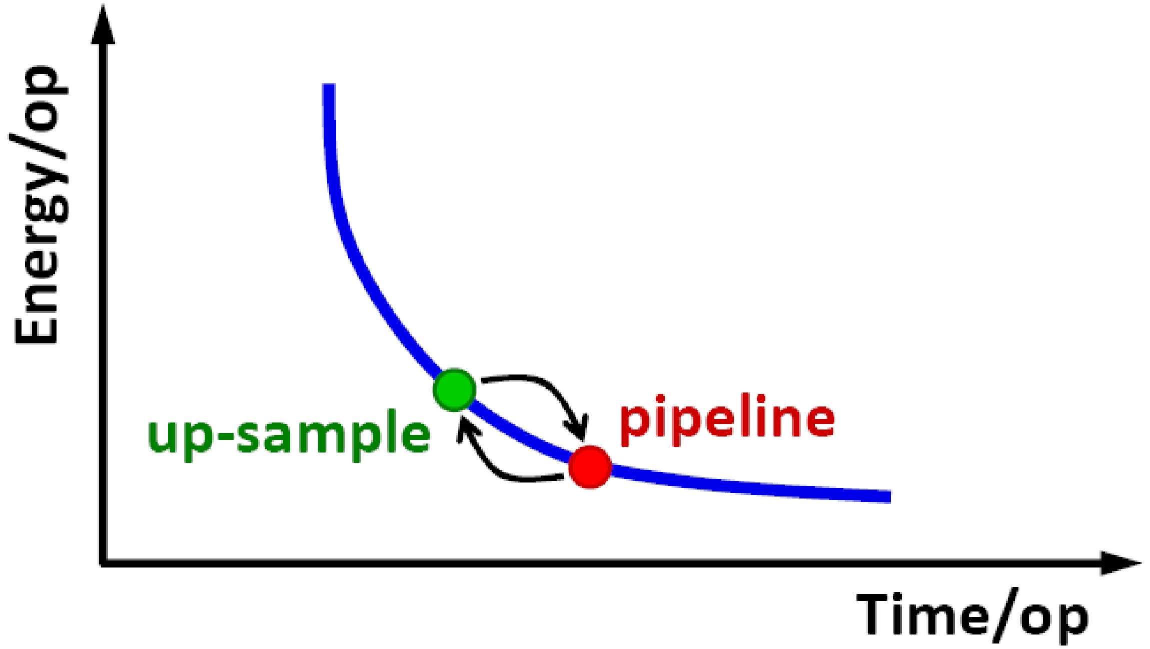






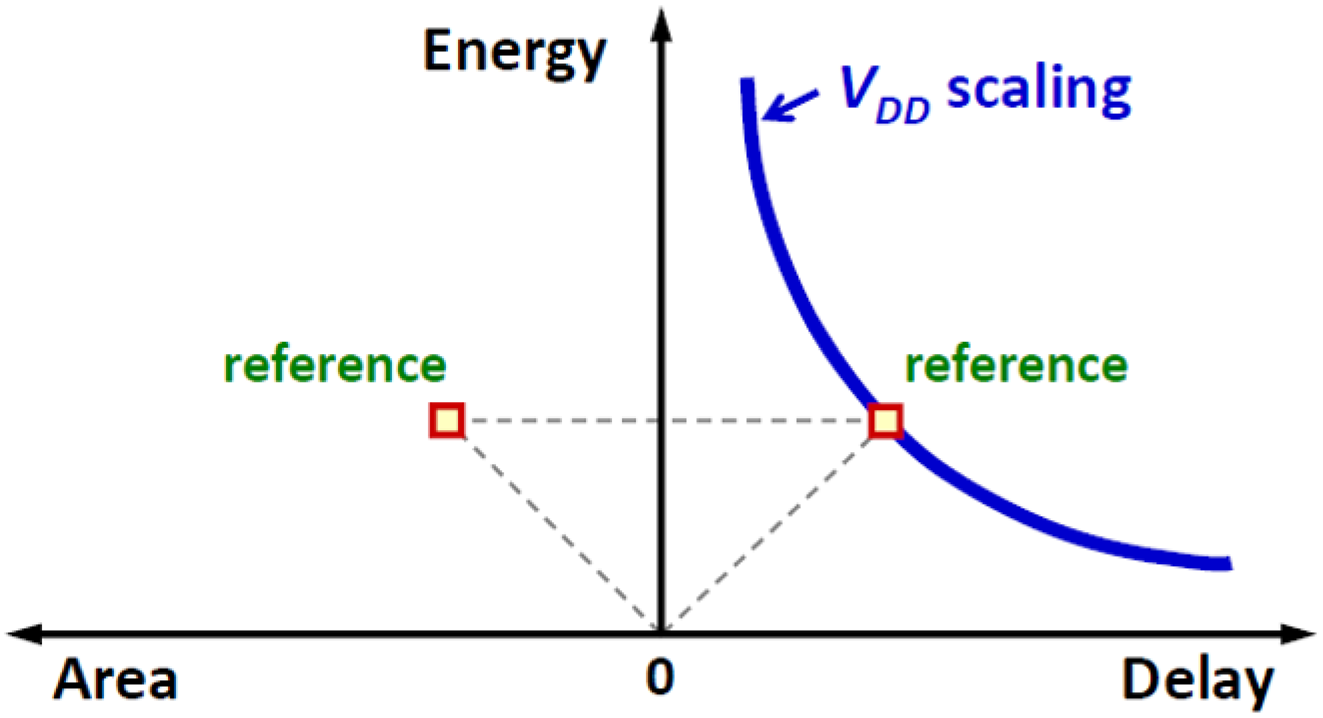
Area Benefit of Interleaving and Folding

- Area: $A = A_{logic} + A_{registers}$
- Interleaving or folding of level N
  - $A = A_{logic} + N \times A_{registers}$
- Timing and Energy stay the same



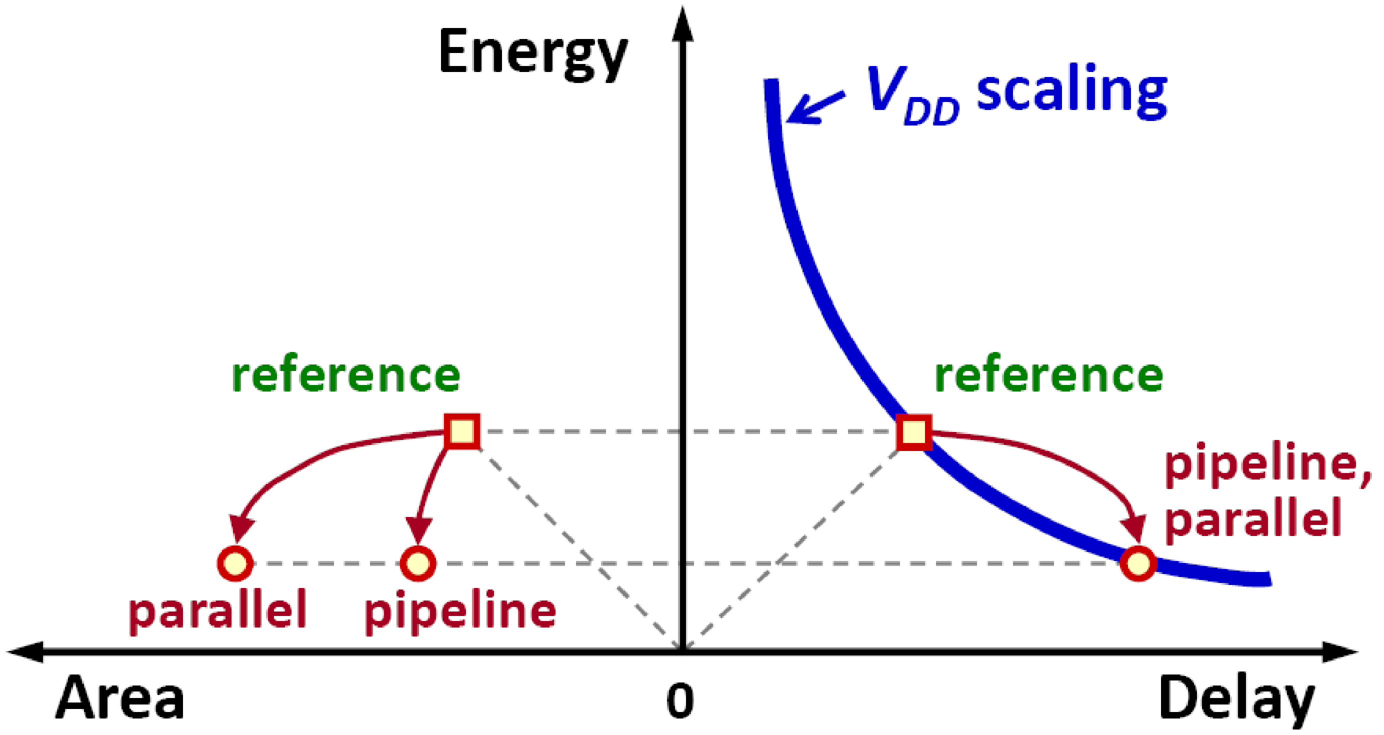
Architecture Transformations

- Procedure: move toward desired E-D point while minimizing area



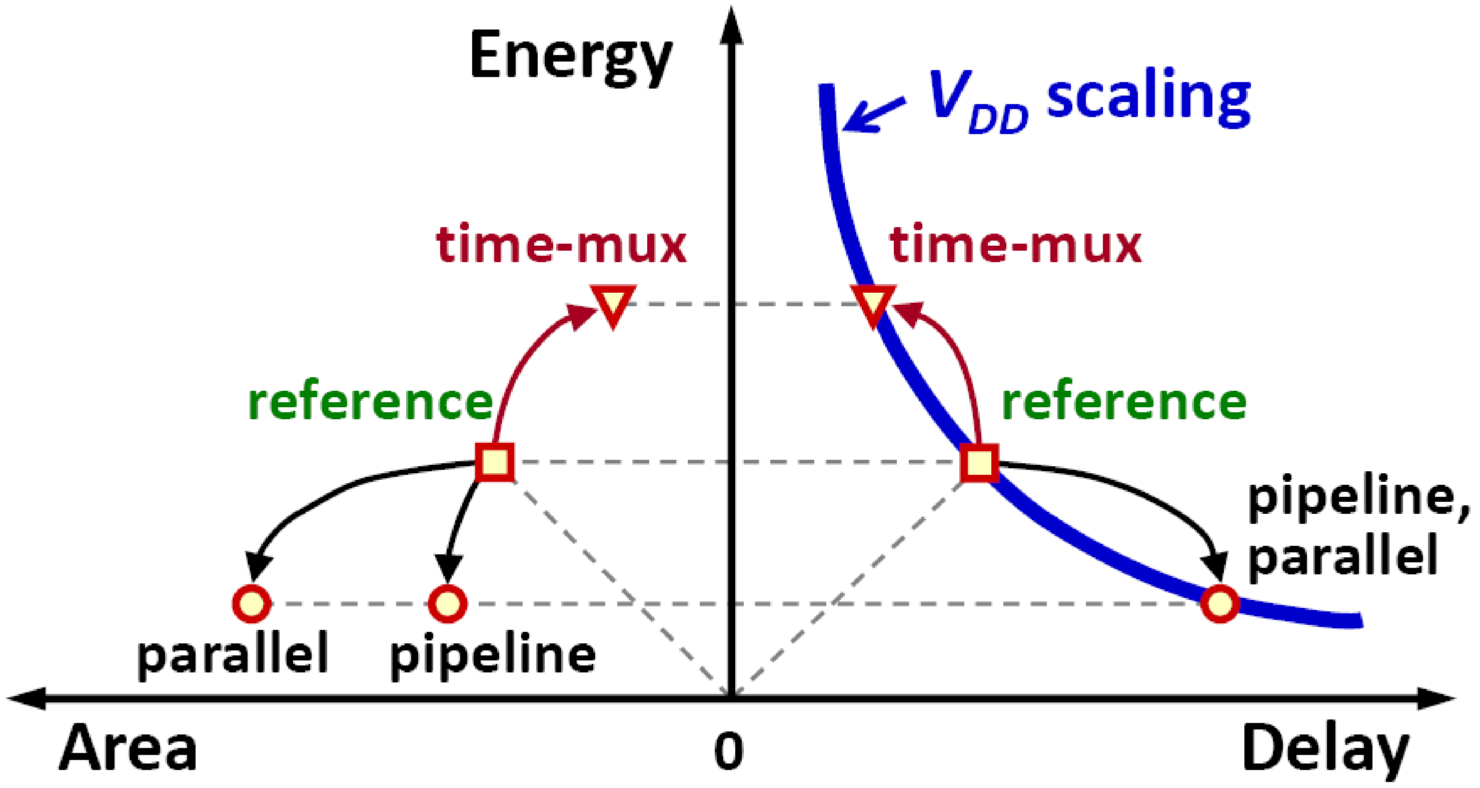
Parallelism & Pipelining Transformations

- **Parallelism & Pipelining**
  - reduce Energy, increase Area



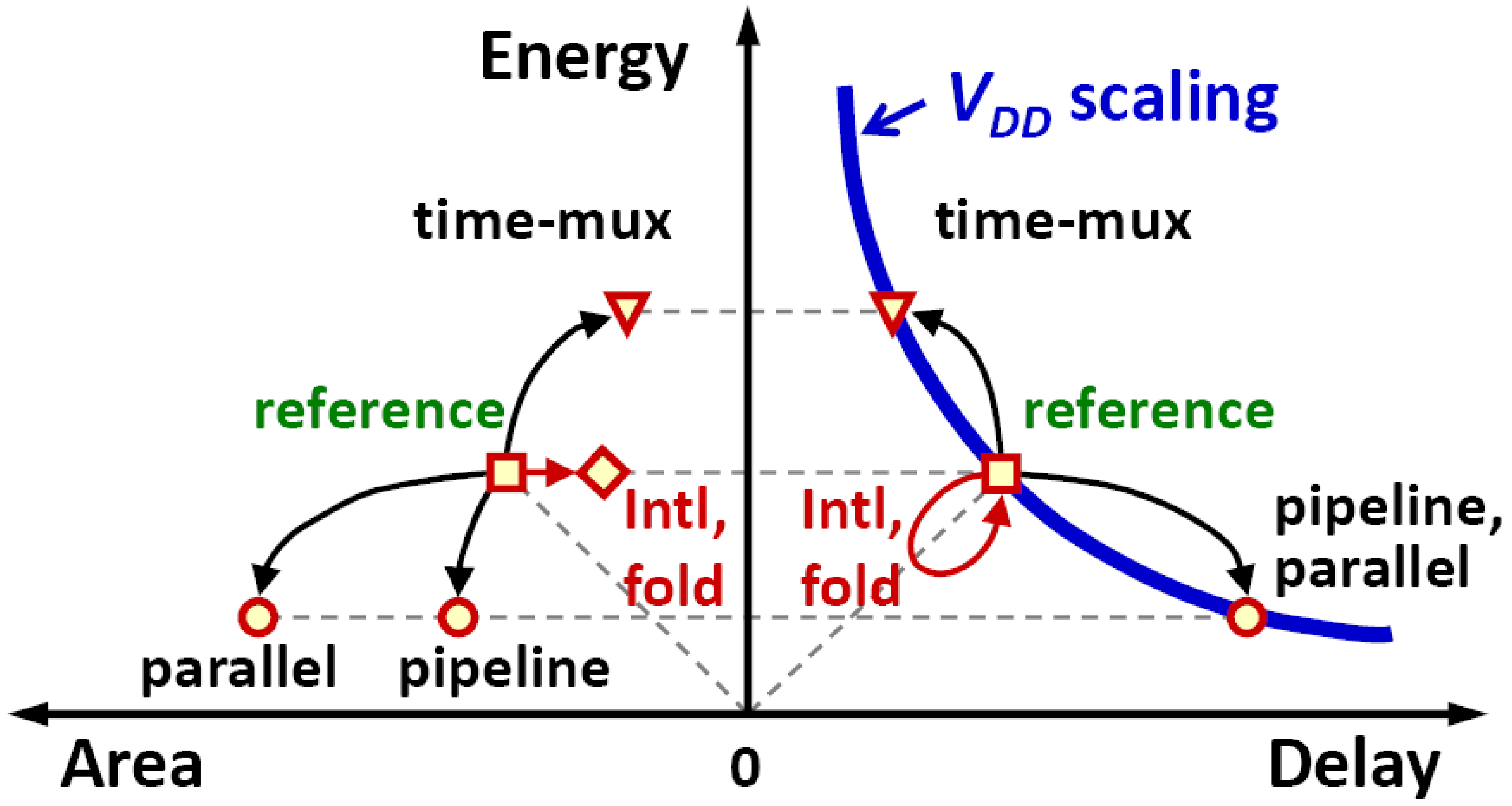
Time Multiplexing Transformations

- **Time multiplexing**
  - increase Energy, reduce Area



Interleaving & Folding Transformations

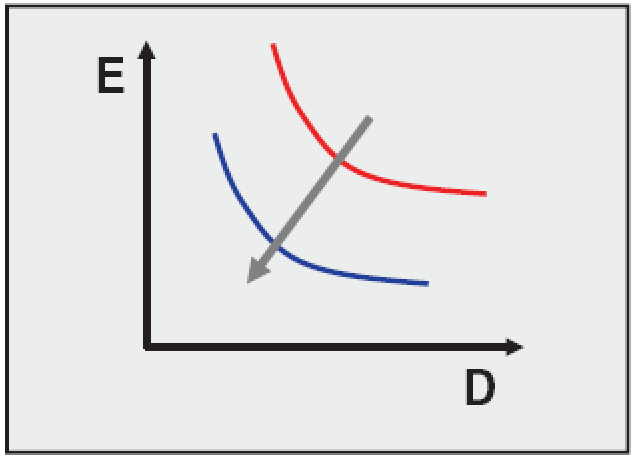
- **Interleaving & Folding (time-mux + pipeline)**
  - ≈ const Energy, reduce Area



Some Energy-Inspired Design Guidelines

- **For maximum performance**
  - Maximize use of concurrency at the cost of area
- **For given performance**
  - Optimal amount of concurrency for minimum energy
- **For given energy**
  - Least amount of concurrency that meets performance goals
- **For minimum energy**
  - Solution with minimum overhead (that is – direct mapping between function and architecture)

Improving Computational Efficiency



Implementations for a given function maybe inefficient and can often be replaced with more efficient versions **without penalty in energy or delay**

- **Inefficiencies arise from:**
  - Over-dimensioning or over-design
  - Generality of function
  - Design methodologies
  - Limited design time
  - Need for flexibility, re-use, and programmability