SoC Architecture and Codesign

COE838: Systems-on-Chip Design
http://www.ecb.torontomu.ca/~courses/coe838/

Dr. Gul N. Khan
http://www.ecb.torontomu.ca/~gnkhan
Elect., Computer & Biomedical Engineering
Toronto Metropolitan University

Overview

- Introduction to Hardware-Software Co-design
- Hardware Options for System-on-Chip
- Accelerator based SoC Architectures
- CPU-Accelerator Co-design Process

Some Material from Chapter 8 of the Text by M. Wolf

SoC Architecture Design

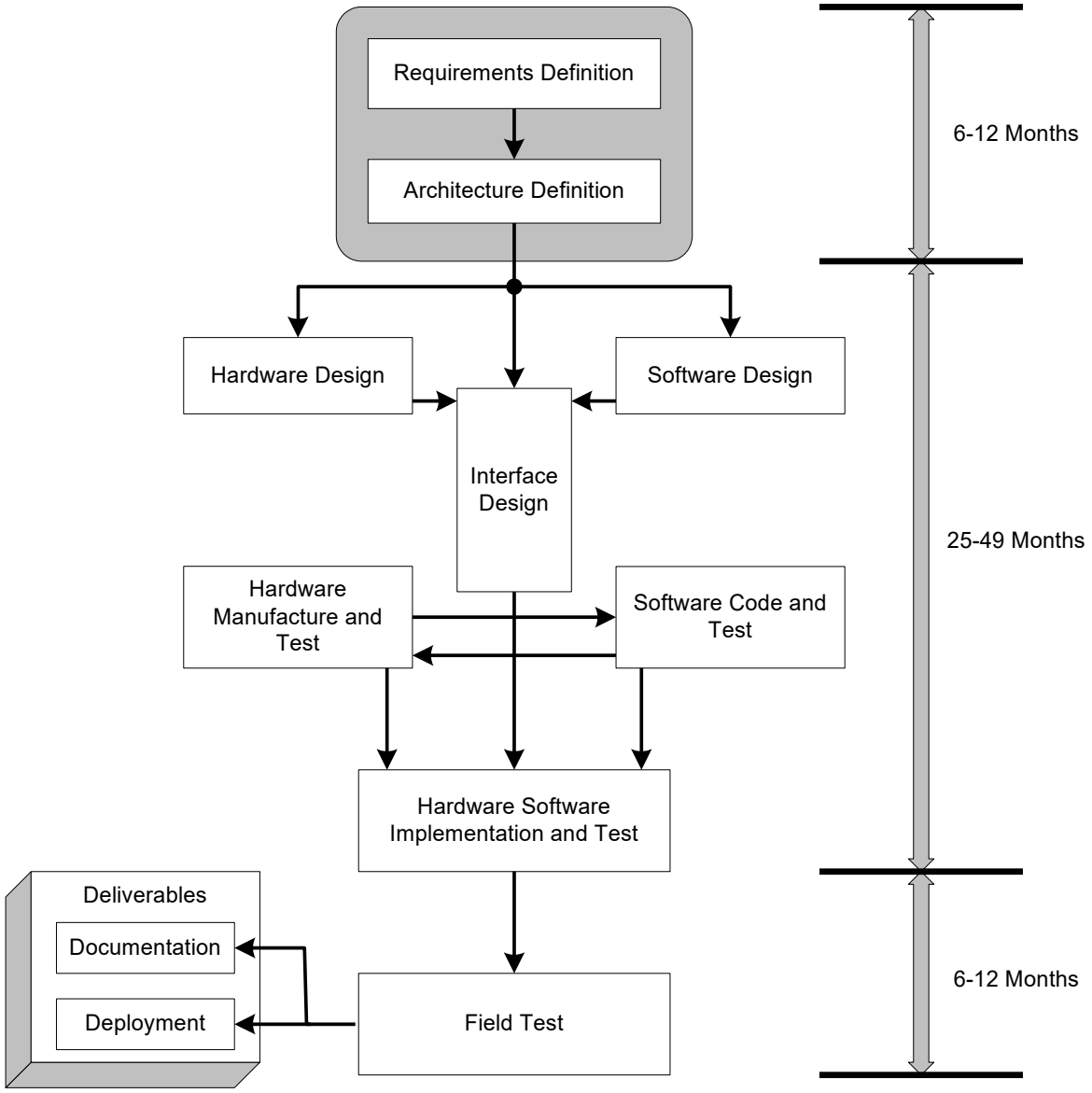
- Real-time System Design
  - Performance analysis
  - Scheduling and allocation
- Accelerated systems
- Use additional computational unit dedicated to some functions?
  - Hardwired Logic e.g. FPGA

- **Hardware/software co-design**: a joint design of hardware and software architectures of SoC.

Traditional Design Practices

- **Performance Requirements make it impossible to execute the entire application in software.**
- **Computationally intensive parts are to be extracted and realized as custom hardware.**
- **Early Design Cycle Partitioning Problems:**
  - **Design Space is not fully Explored**
  - **High Cost Design**
  - **Inefficient**

Traditional Design Practice



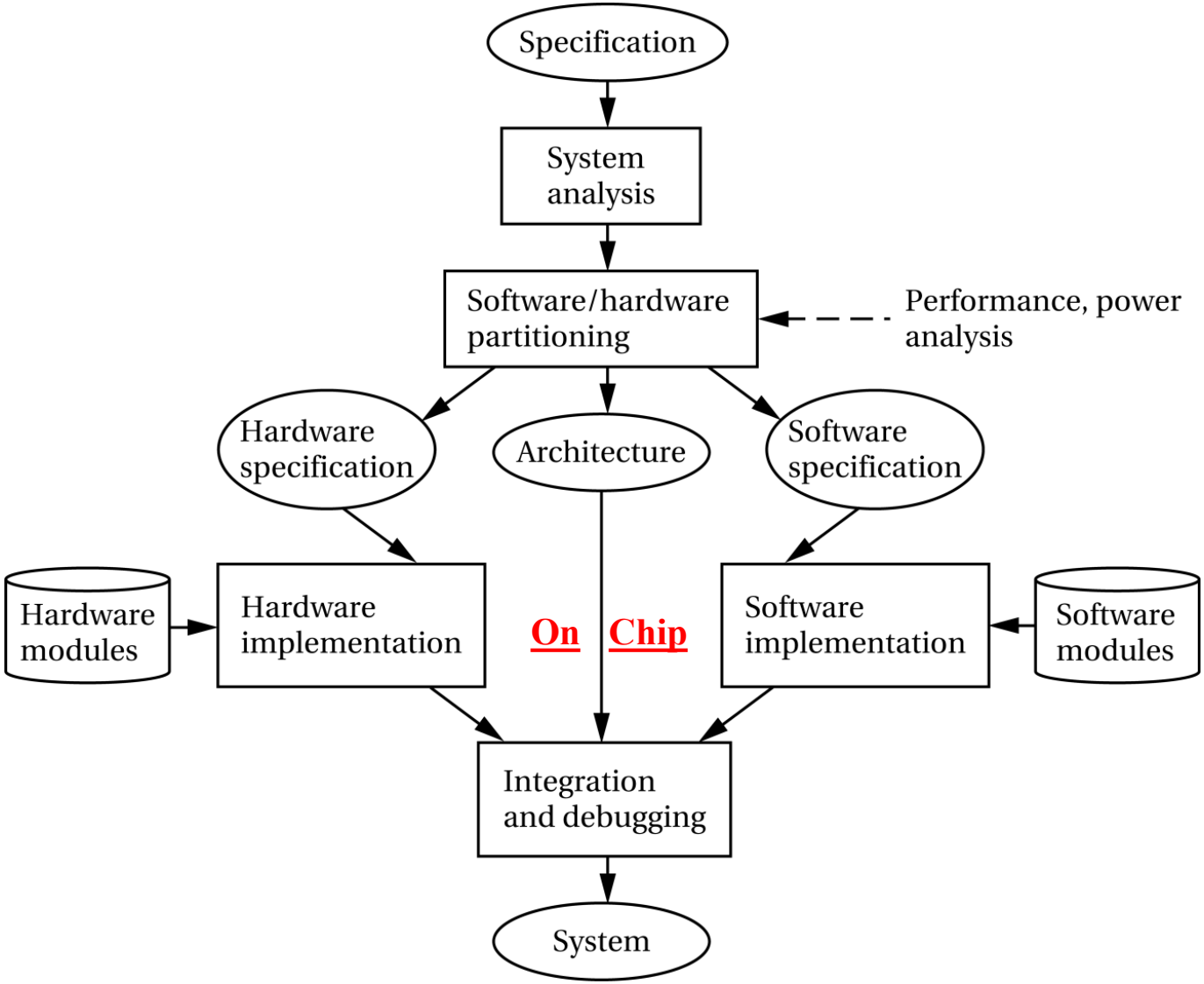
Advancements

- VLSI Technology advances has led to:
  - Smaller and Faster IP Cores
  - Reconfigurable Logic
- Matured Hardware Design Methodology
- Matured Software Design Methodology
- Joint design – Still in progress and very popular!

Hardware-Software SoC Codesign

- An approach utilizing the maximum efficiency of Hardware and Software is needed
- Recent developments in CAD Tools
- Result -- Hardware Software Co-design
  - Large Design Space Exploration
  - Improved Time to Market

Codesign Methodology



Hardware-Software Codesign

Study of the design of (SoC-based) computer systems encompassing the following parts:

- **Co-Specification**

  Developing system specification that describes hardware and software modules and relationship among them

- **Co- Synthesis**

  Automatic and semi-automatic design of hardware and software elements to meet the specification.

- **Co-Simulation**

  Simultaneous simulation of hardware and software.

HW/SW Co-Specification

- Consider System functionality at an Abstract Level
- No Concept of Hardware or Software
- Must Capture SoC functionality precisely
- Common Specification Approaches

Using High Level Languages

- **C/C++**
  - **Inclined to Software description**
- **VHDL/Verilog**
  - **Inclined to Hardware description**
- **SystemC**
  - **Combined features for Hardware and Software Representation**

Hardware-Software Co-Specification

- **Common Specification Approaches**

  Using Task Graphs

  - **Grey Area**
    - **High Level descriptions are usually translated to some form of a task graph**
  - **Common Forms**
    - **Data flow graphs**
    - **Control flow Graphs**
    - **Control-Data Flow Graphs**

Hardware-Software Partitioning

- Assignment of System parts to implementation units (Hardware and Software)
- Goals of Partitioning Algorithm
  - **Meeting the constraints**
  - **Minimize the system/SoC cost**
- Directly affects the cost and performance of final system

  Main Problems
- Significant Research Area

Hardware-Software Partitioning

- **Granularity: System Components' Size i.e. assigned to Hardware or Software**
- **Coarse Grain Approaches**

  Assign Complete Function or Processes to hardware or software

  - **A single task is a large block of system functionality**
  - **Prevent Excess Communications**
  - **Example tasks: MPEG decode, JPEG-2000 encode, FFT, DCT, etc.**

Hardware-Software Partitioning

- **Fine Grain Approaches: Operate on basic operations (add, subtract, multiply, etc.)**

  Avoid the problem of dealing with poorly defined functional specifications

  - **Useful when dealing with partially re-configurable processors (IP cores that can be modified during the design process)**
  - **Here tasks are often referred as base blocks.**

Hardware Software Partitioning

Flexible Granularity: Computationally intensive parts are small loops which are hidden inside a function or process.

- **Coarse grain approaches lead to costly designs, some redundant parts may be moved to hardware**
- **Fine Grain approaches make up such a large design space which is usually very hard to explore**

- **Perform HW/SW partitioning handling decisions at both high and low levels.**

Partitioning Approaches

Optimal Partitioning Approaches

Exhaustive

- Computationally very Intensive
- Limited to small task graphs

Branch and Bound

- Start from a good Initial Condition
- Try pre-sorted combinations
- Exhaustive in limiting case

Heuristic Approaches

Heuristic Optimization Techniques

Simulated Annealing, Tabu Search and (GA) Genetic Algorithm

Greedy Approaches:

Start from all SW or HW architectures and move parts to HW or SW.

http://www.ecb.torontomu.ca/~courses/coe718/lectures/HS-Codesign-Overview.pdf

Hardware-Software Co-Synthesis

Four Principle Phases of Co-synthesis:

- **Partitioning**

  Dividing the functionality of a specific system or SoC into units of computation.

- **Scheduling - Pipelining**

  Task Start Timing: Choosing time at which various computation units will occur.

- **Allocation**

  Determining the processing elements (PEs) on which computations will occur.

  Selection, type of Processing Elements and Communication Structure (System Architecture)

- **Assignment**

  Task Mapping: Choosing particular component types for the allocated units (of computations).

Hardware Software Co-Synthesis

- Automatic System/SoC Architecture definition (e.g., bus, tree, …)
- Tightly coupled with HW/SW Partitioning
- Must schedule system to determine feasibility of each solution being evaluated

Hardware Software Co-Synthesis

Approaches

Optimal

- Try all possible combinations
- Limited use due to Large Design Space
- Examples

Heuristic

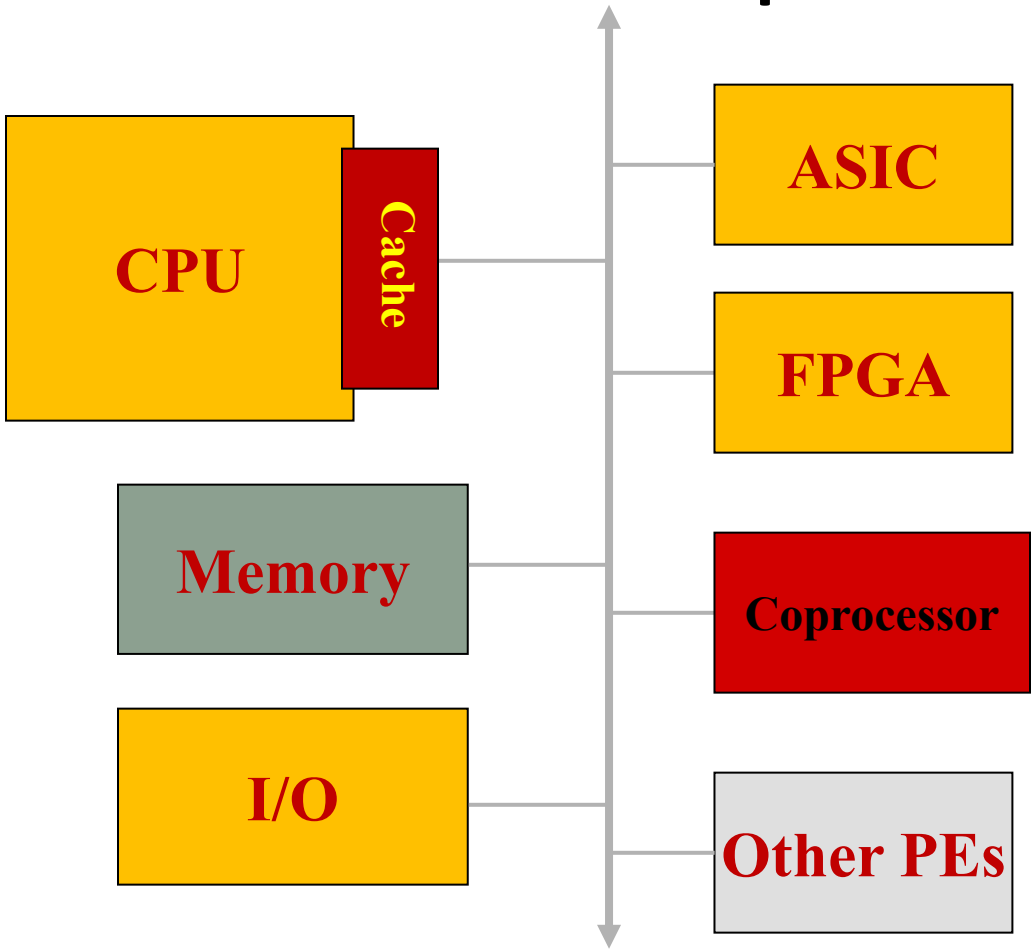
- Avoid large Execution times
- Usually give ‘Good’ results

Hardware Software Co-Synthesis

- **Iterative Heuristic Approaches**
  - **Start with an initial solution**
  - **With each iteration of the algorithm improve the solution somewhat.**
  - **As the algorithm progresses the solution is refined.**

SoC Hardware Structure

Various Hardware Options:



System Partitioning

Introduces a design methodology that uses several techniques:

- **Partition system specification into tasks (processes).**
  The best way to partition a specification depends on the characteristics of the underlying hardware platform.

- **Determine the performance of the function when executed on the hardware platform.**

  We usually rely on approximating. Exact performance depends on hardware-software details.

- **Allocate processes onto various processing elements.**

Hardware-Software Partitioning

- **Hardware/software system design involve: Modeling, Validation and Implementation**
  - **System Implementation involves:**
    Hardware-software partitioning (Cosynthesis)
  - **Hardware-Software Partitioning**
    Identifying parts of the system model best implemented in hardware and software modules
  - **Such partitions can be decided by the designer**

    or determined by the codesign (CAD) tools

Hardware-Software Partitioning

- For embedded systems, such partitioning represents a physical partition of the system functionality into:
  - Hardware (CPU), accelerators, GPU, etc.
  - Software executing on one or more CPUs

  Various formation of the Partitioning Problem that are based on:
  - Architectural Assumptions
  - Partitioning Goals
  - Solution Strategies

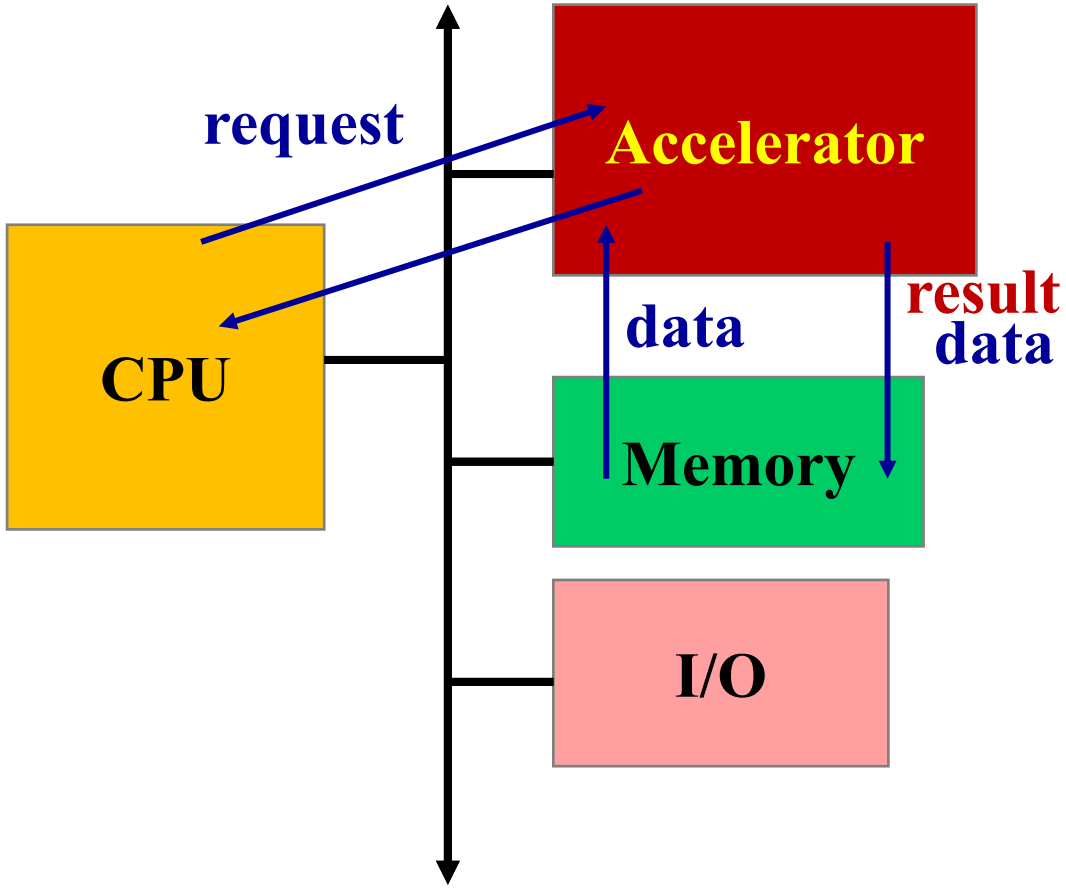
  COWARE: A design environment for application specific architectures targeting telecom applications.

Partitioning Techniques

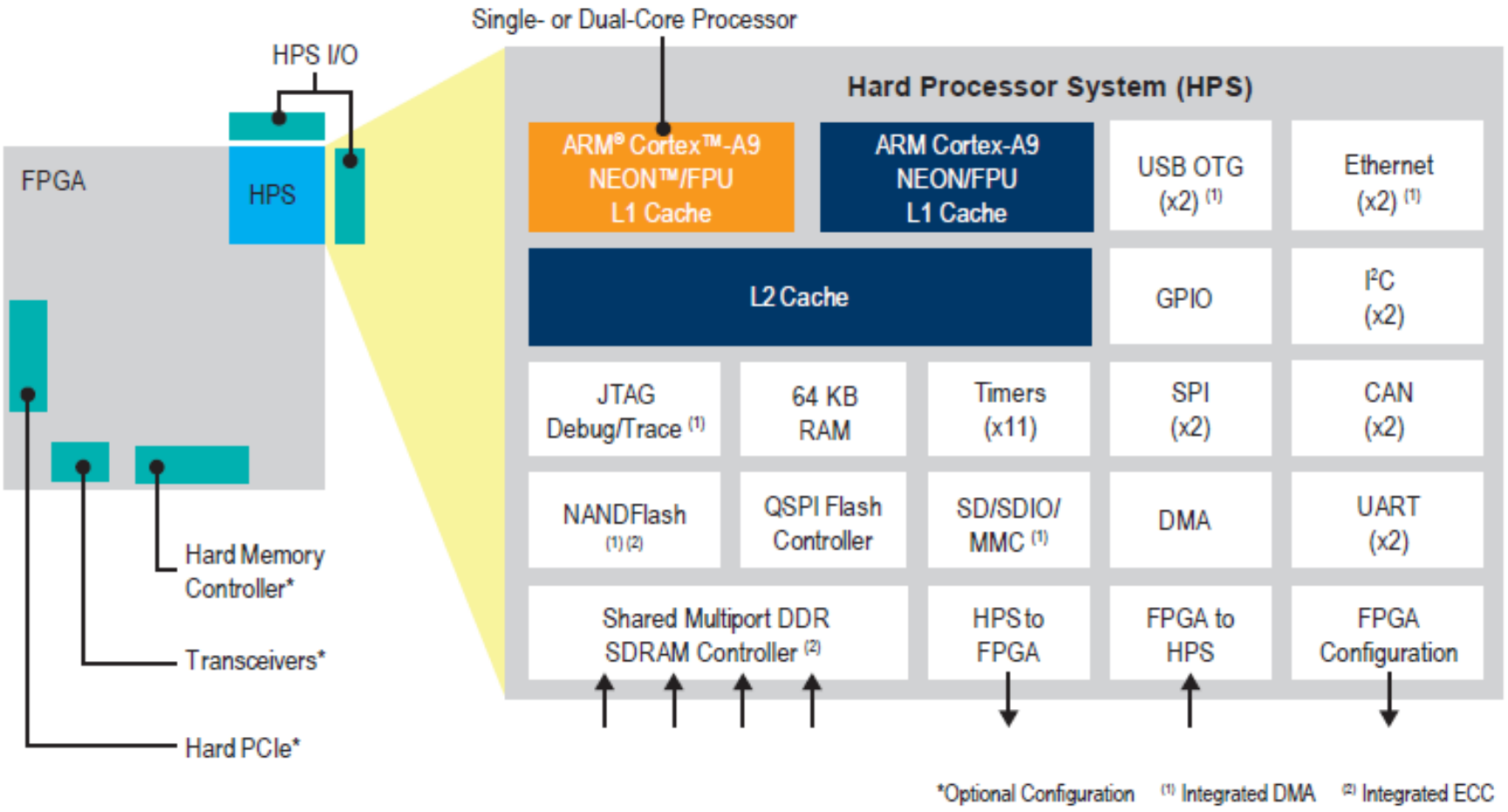
Hardware-Software Homogeneous System Model => task graph

- **For each node of the task graph, determine the implementation choices (HW or SW)**
  - **Keep the scheduling of nodes at the same time Meeting the real-time constraints (deadline, Exe time)**
    - **There is an intimate relationship between partitioning and scheduling.**
    - **Wide variation in timing properties of the hardware and software implementation of a task.**

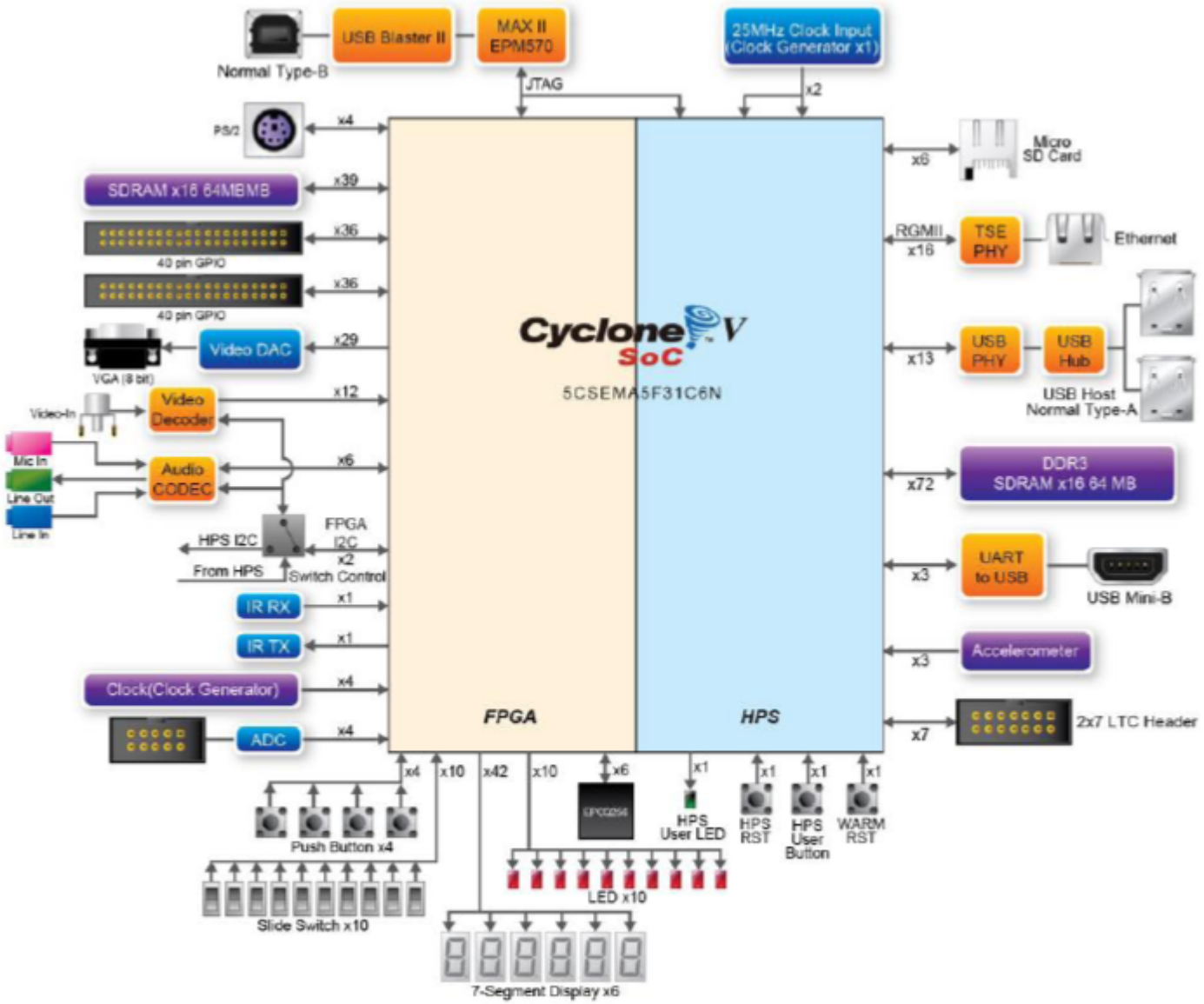
Accelerator (ASIC, FPGA, etc.) System Architectures



SoC-HPS
Cyclone-V Hard Processor System



DE1-SoC


USB Blaster II
MAX II EPM570
Normal Type-B
JTAG
25MHz Clock Input (Clock Generator x1)
x2
PS/2
x4
SDRAM x16 64MB
x39
40 pin GPIO
x36
40 pin GPIO
x36
VGA (8 bit)
Video DAC
x29
Video-In
Video Decoder
x12
Mic In
Line Out
Line In
Audio CODEC
x6
HPS I2C
FPGA I2C
x2
From HPS
Switch Control
IR RX
x1
IR TX
x1
Clock(Clock Generator)
x4
ADC
x4
Cyclone V SoC
5CSEMA5F31C6N
FPGA
HPS
Micro SD Card
x6
RGMII
x16
TSE PHY
Ethernet
x13
USB PHY
USB Hub
USB Host Normal Type-A
x72
DDR3 SDRAM x16 64 MB
x3
UART to USB
USB Mini-B
x3
Accelerometer
x7
2x7 LTC Header
x4
x10
Push Button x4
Slide Switch x10
x42
x10
LED x10
7-Segment Display x6
x6
x1
HPS User LED
x1
HPS RST
x1
HPS User Button
x1
WARM RST

Accelerator *vs* Coprocessor

- A co-processor executes instructions
  - Instructions are dispatched by the CPU
- Accelerator appears as a device on the bus
  - Accelerator is mainly controlled by registers

Accelerator Implementations

- Application-specific integrated circuit
- Field-programmable gate array
- Standard component

Example

SoC Architecture Design Tasks

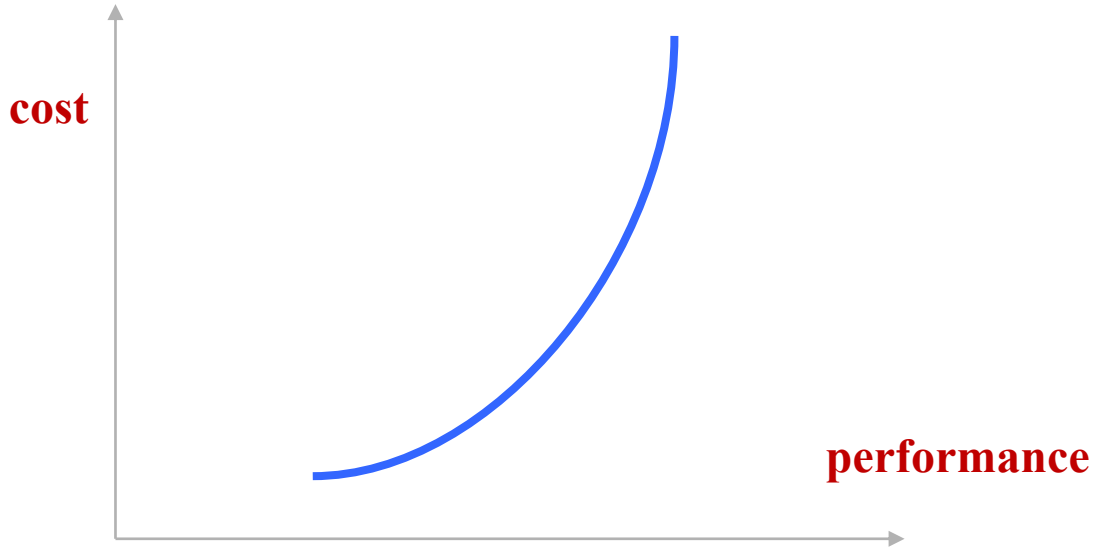
- Design Heterogeneous PEs Multiprocessor SoC architecture.
  - Processing Elements (PEs)

    CPU, accelerator, etc.
- Program the system

Why Accelerators?

Better Cost/Performance

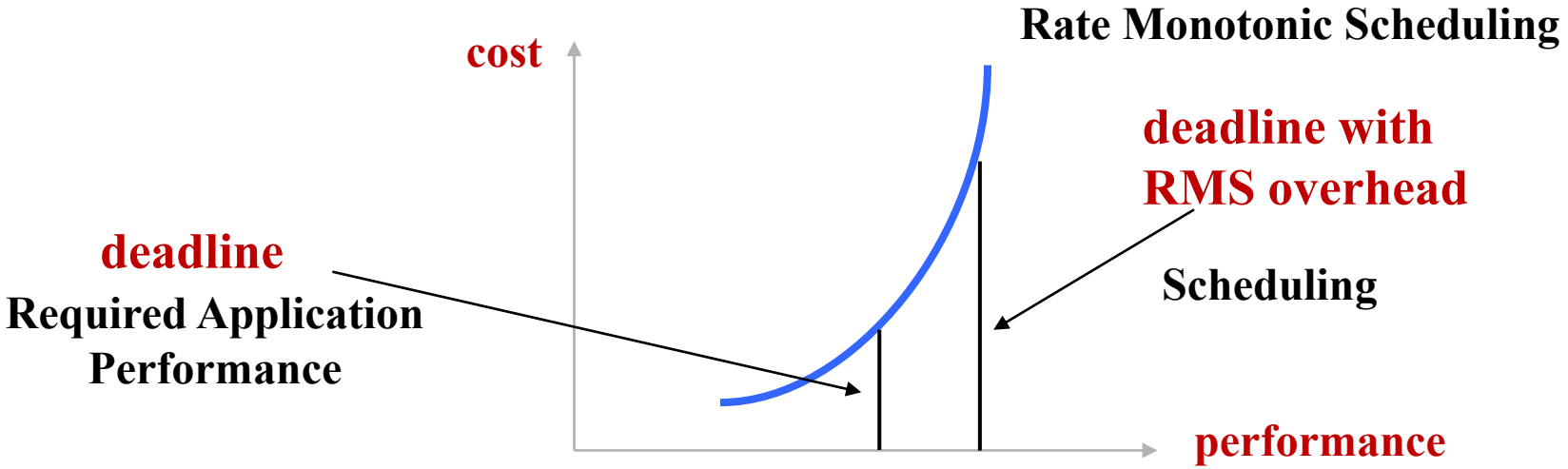
- Custom logic may be able to performthe operation faster than a CPU of equivalent cost
- CPU cost is a non-linear function of performance



CPU and Accelerators

Better Real-time Performance

- Put (schedule) time-critical functions on lightly-loaded processing elements.
- Remember RMS utilization (<100%) --- extra CPU cycles must be reserved to meet deadlines.



Why Accelerators?

- Good for processing I/O in real-time.
- May consume less energy.
- May be better at streaming data.
- May not be able to do all the work on even the largest single CPU.

Accelerated System Design

- First, determine that the system really needs to be accelerated.
  - How much faster is the accelerator on the core function?
  - The data transfer overhead?
- Design the accelerator itself
- Design CPU interface to the accelerator

Performance Analysis

- **Critical parameter --**

How much faster is the system with the accelerator?

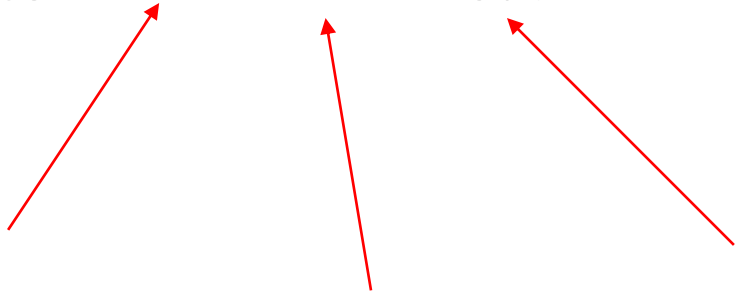
- **Must take into account:**
  - **Accelerator execution time**
  - **Data transfer time**

    (in-between CPU and Accelerator
  - **Synchronization with the (master) CPU**

Accelerator Execution Time

Total accelerator execution time:

$$t_{accel} = t_{in} + t_x + t_{out}$$



Data Input/Output Times

- For Bus-based SoCs - bus transactions include:
  - Flushing register/cache values to main memory
  - Time required for CPU to set up the transaction
  - Overhead of data transfers by bus packets, handshaking, etc.

Accelerator Speedup

- Assume loop is executed for ***n*** times.
- Compare accelerated system to non-accelerated system:
- Speedup =

$$= n(t_{CPU} - t_{accel})$$

$$= n[t_{CPU} - (t_{in} + t_x + t_{out})]$$

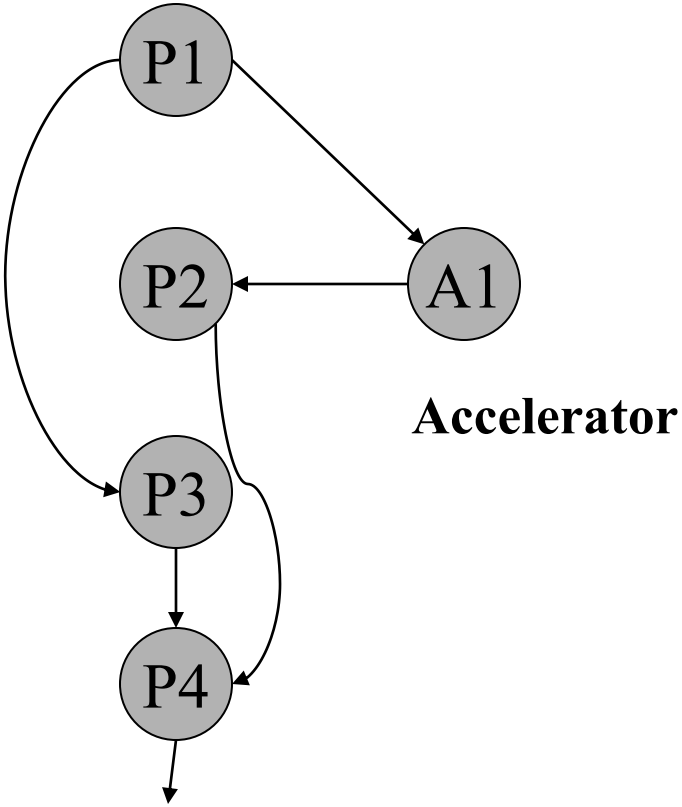
Single-threaded vs Multi-threading

- One critical factor is the available parallelism in the application:
  - Single-threaded/blocking:
    CPU waits for the accelerator
  - Multithreaded/non-blocking:
    CPU continues to execute along with accelerator.
- For multithread, CPU must have some useful work to do while accelerators perform some tasks.
  - Software environment must also support multi-threading.

Total Execution Time

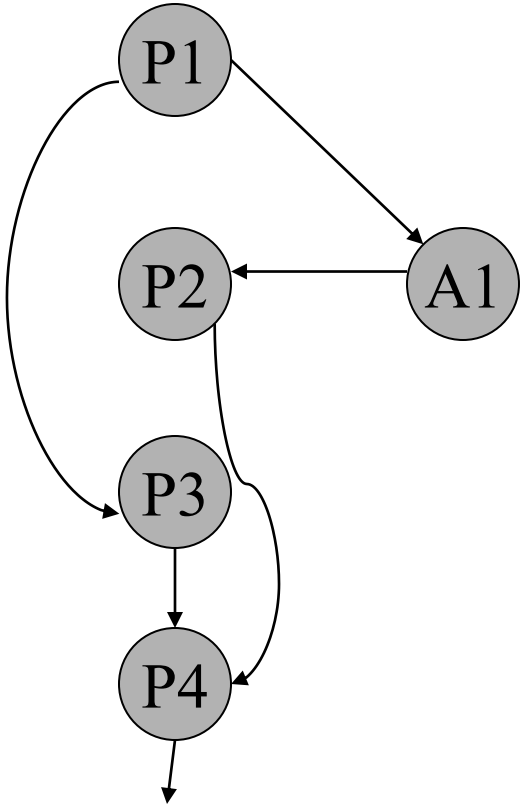
Single-threaded:

Count execution time of all component processes.



Multi-threaded:

Find longest path through execution.



Sources of Parallelism

- **Overlap I/O and the Accelerator Computation.**
  - **Perform operations in batches, read in second batch of data while computing on first batch.**
- **Find other work to do on the CPU.**
  - **May reschedule operations to move work after accelerator initiation.**

Accelerator/CPU Interface

- Accelerator registers provide control registers for CPU.
- Data registers can be used for small data objects.
- Accelerator may include special-purpose read/write logic.
  - Valuable for large data transfers

Bus-based SoC

An Earlier Slide

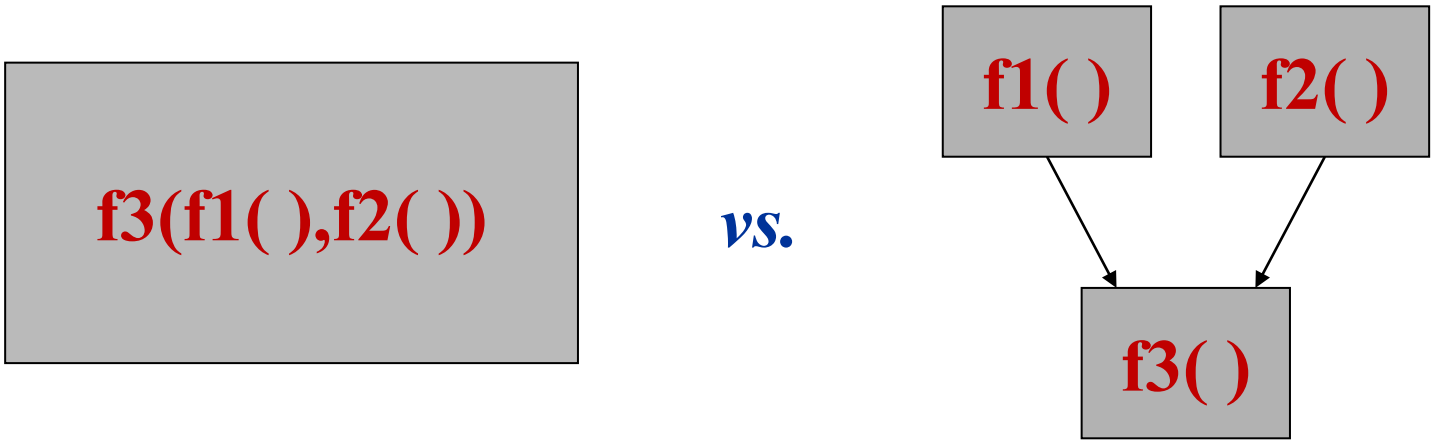
- **Bus transactions include:**
  - **flushing register/cache values to main memory**
  - **time required for CPU to set up transaction**
  - **overhead of data transfers by bus packets, handshaking, etc.**

Memory Caching Problems

- Main memory provides the primary data transfer mechanism to the accelerator.
- Programs must ensure that caching does not invalidate main memory data.
  - CPU reads location S
  - Accelerator writes location S
  - CPU writes location S

Partitioning

- Divide functional specification into units.
  - Map units (e.g., SystemC Modules) onto PEs
  - Units can become processes (tasks)
- Determine proper level of parallelism.



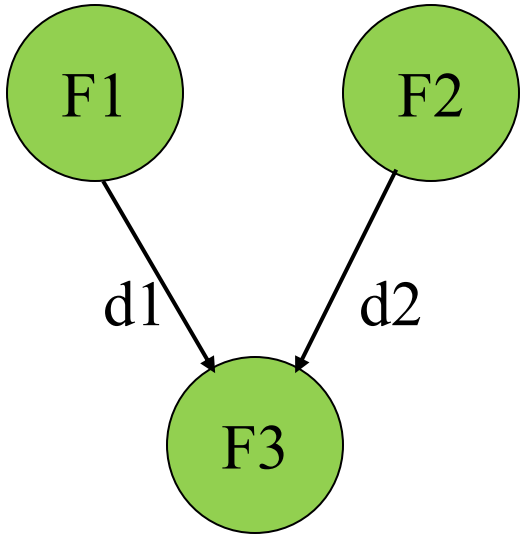
Scheduling and Allocation

Scheduling for calculating TOTAL execution time

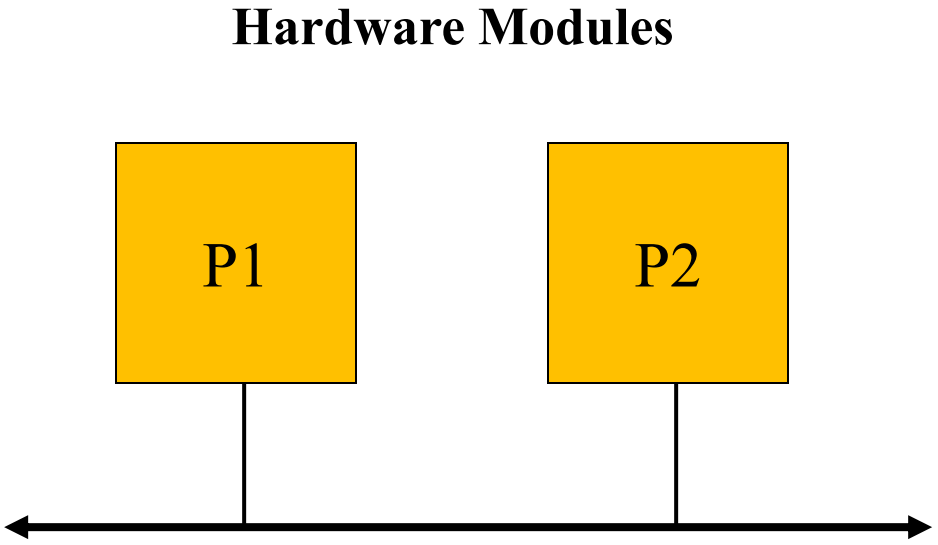
- **We must:**
  - **Schedule operations in time**
  - **Allocate computations to processing elements**
- **Scheduling and allocation interact among each other. However, separating them will be helpful.**
  - **Alternatively allocate, then schedule.**

Scheduling and Allocation

An Example



Task graph



Hardware platform

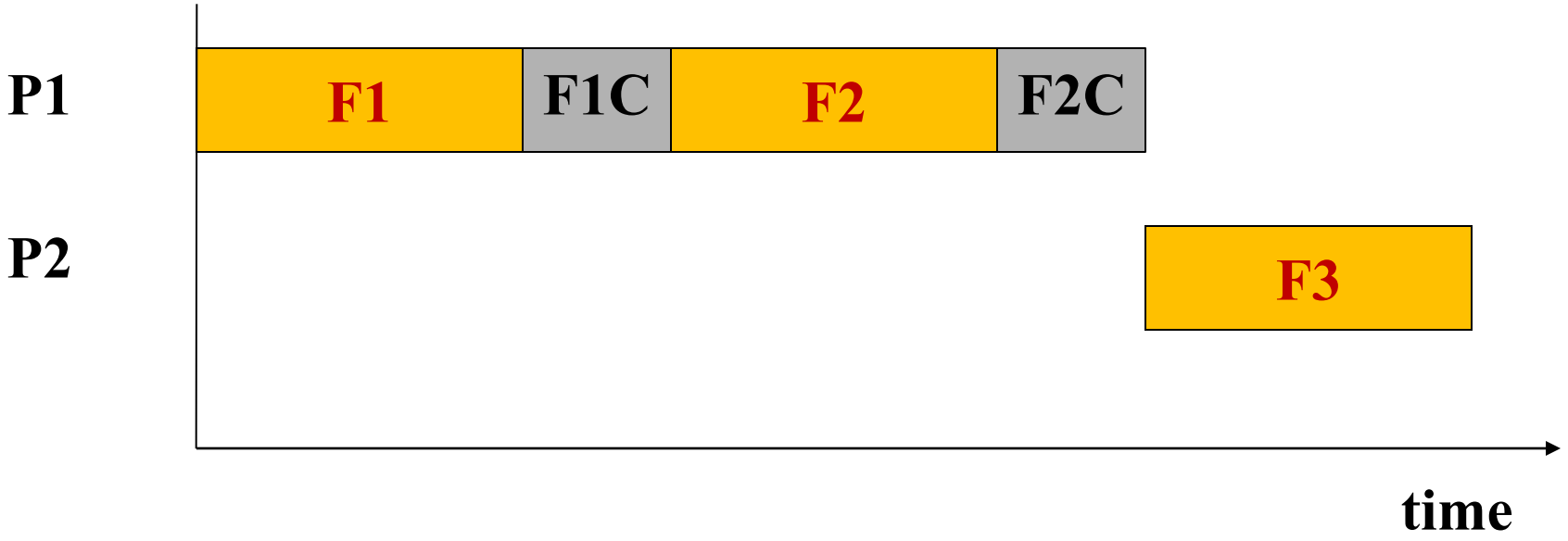
Process Execution Times

Time to perform F1, F2 & F3 on P1 and P2 hardware HW (Library)

	P1(Accelerator)	P2(CPU)
F1	5	5
F2	5	6
F3	-	4

First design

- Allocate F1, F2 -> P1; F3 -> P2.



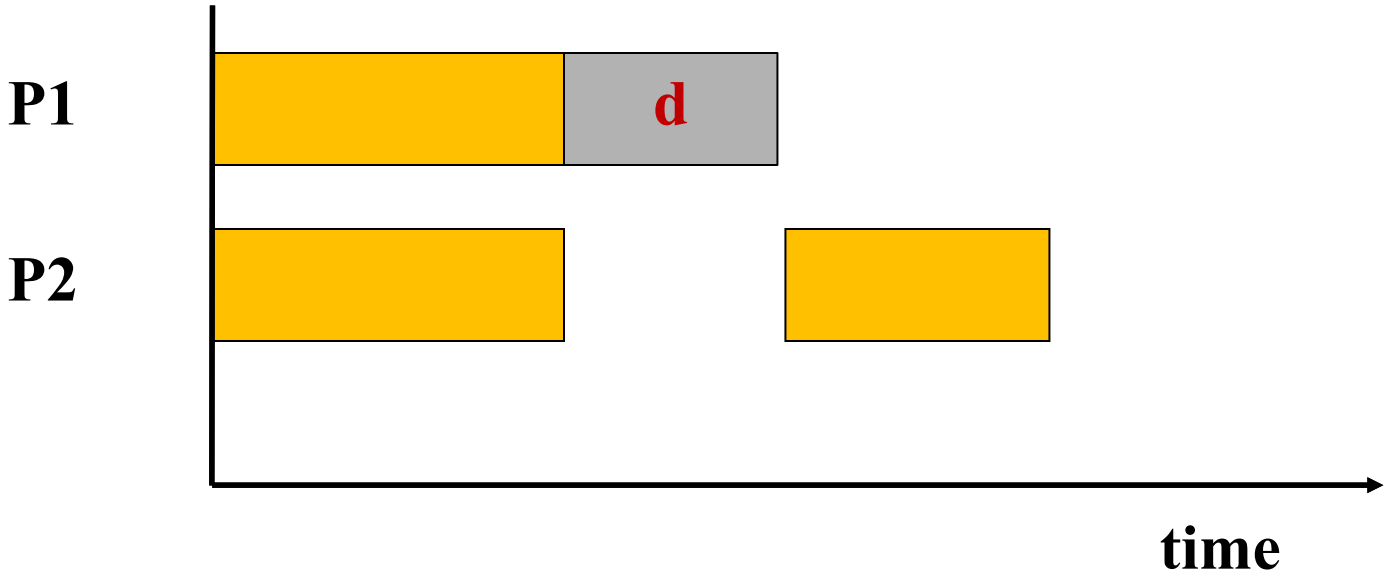
Communication Model

Example

- Assume communication within PE is free
- Cost of communication from F1 to F3 d1 =2;
- Cost of F2 to F3 communication d2 = 4

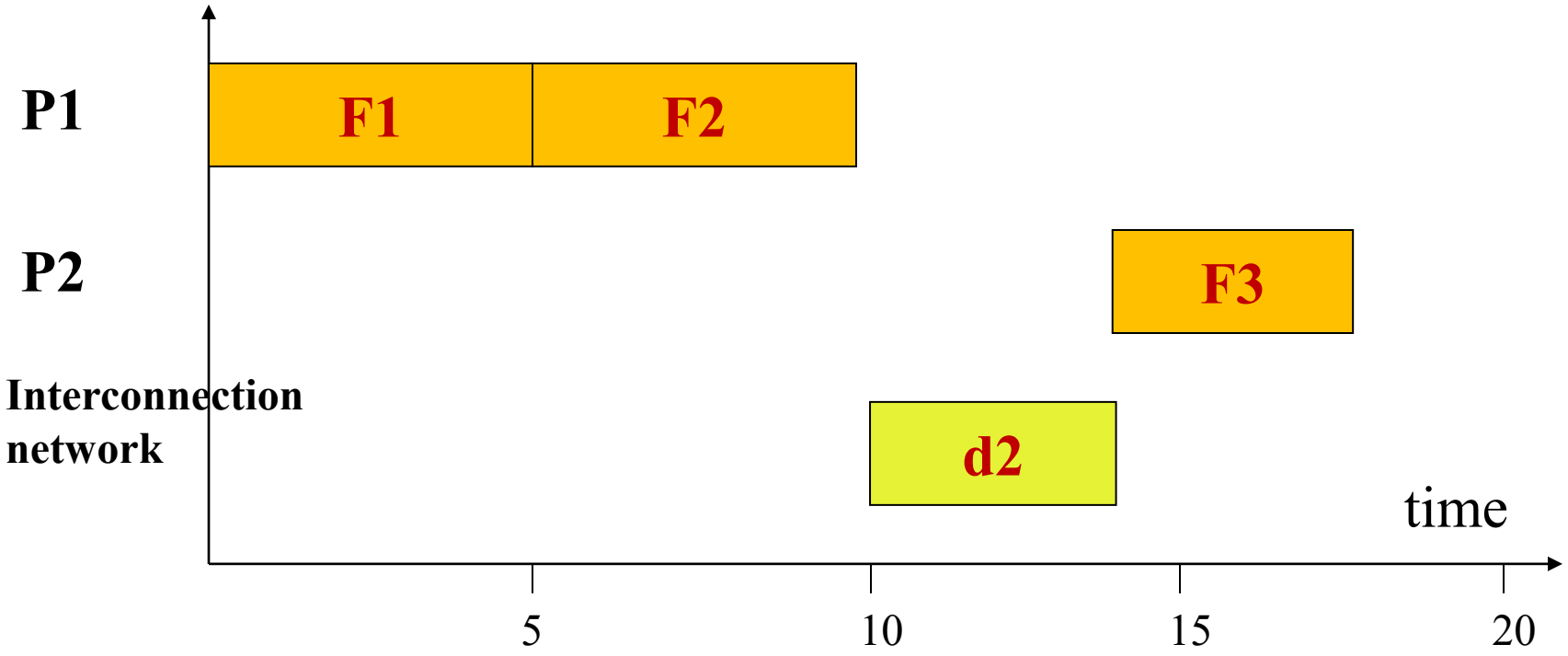
Second design

- Allocate F1 -> P1; F2, F3 -> P2



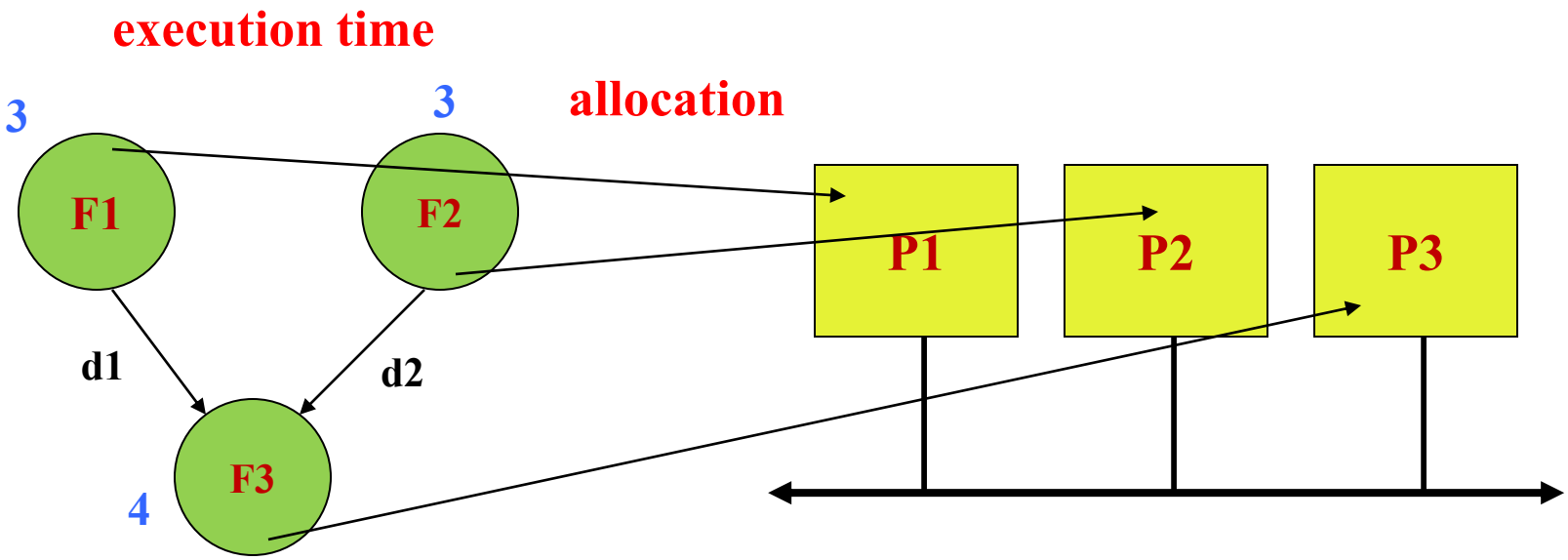
3rd Design Option

- Allocate F1, F2 -> P1; F3 -> P2

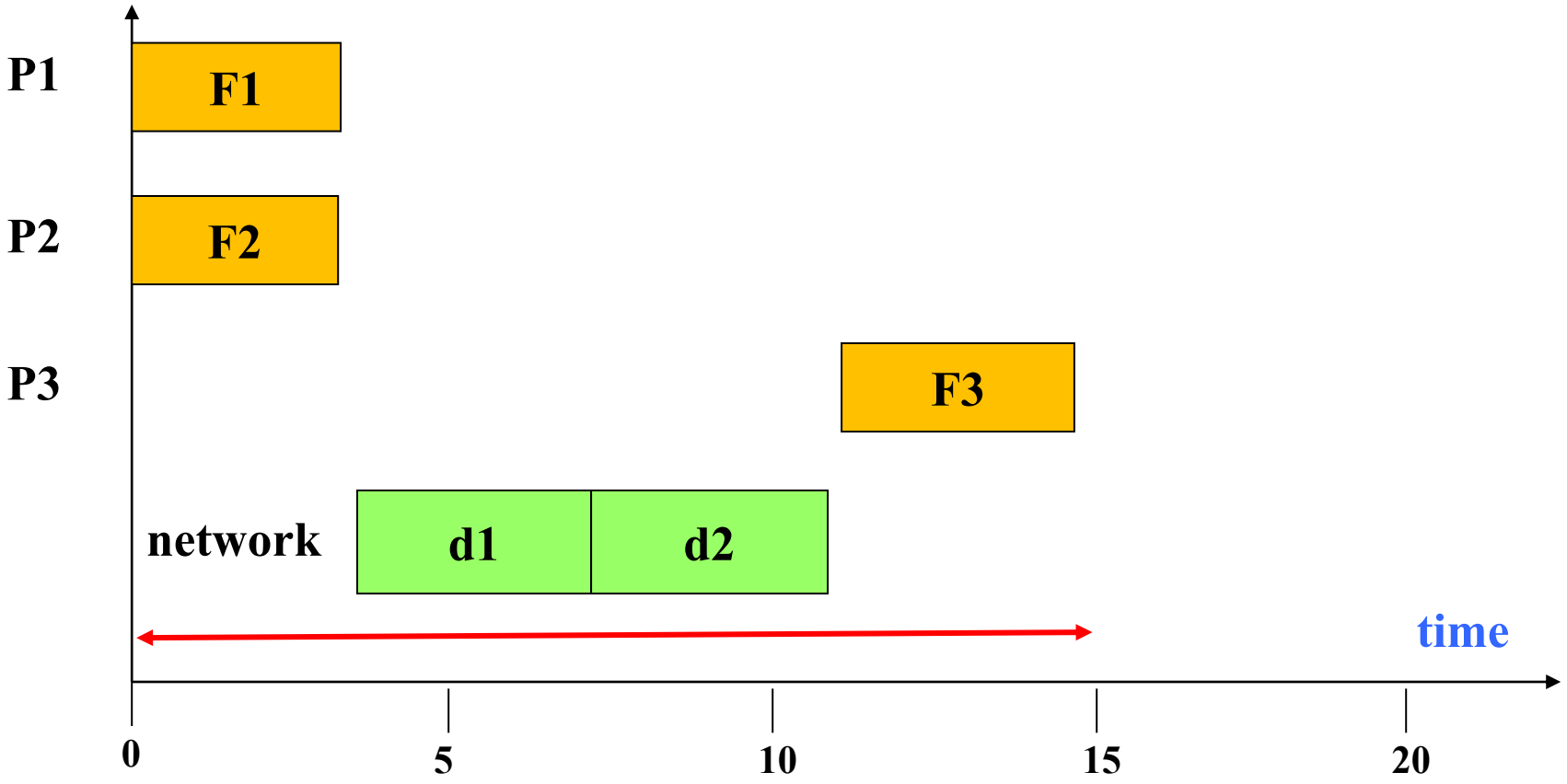


Example: Adjusting Data Size to Reduce Delay

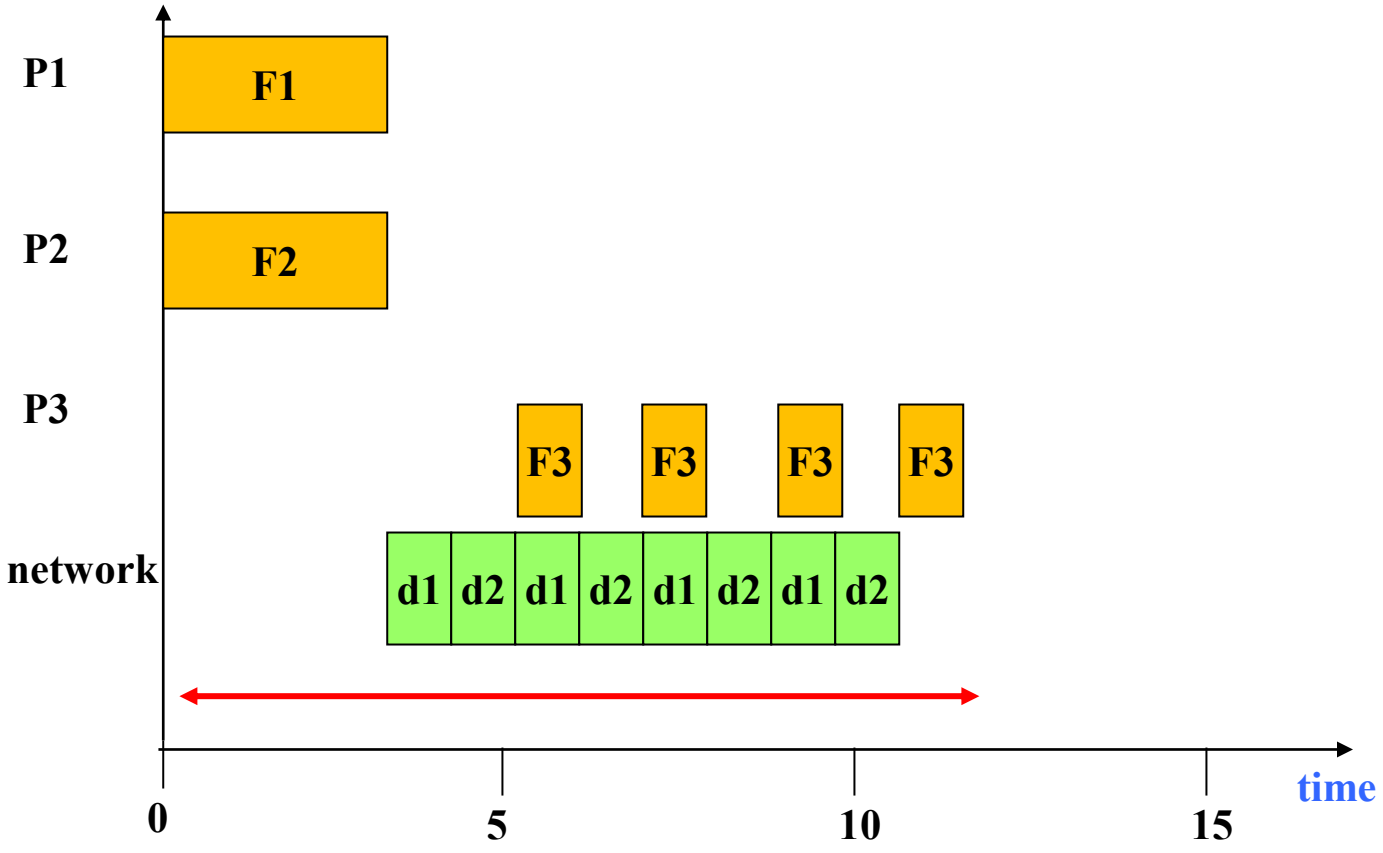
- **Task graph:**
- **Processor Network:**



Initial schedule



An Efficient Schedule

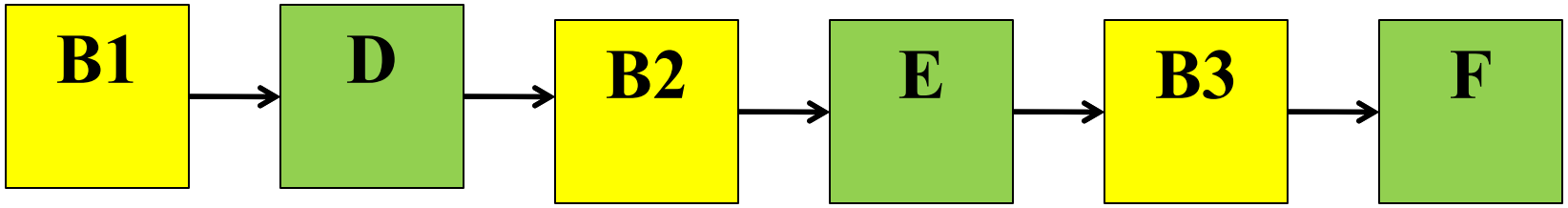


Buffering and performance

- Buffering can sequentialize operations.
  - Next process must wait for data to enter buffer before it can continue.
- Buffer policy (queue, RAM) can affect the available parallelism.

Buffers and latency

- Three processes D, E and F separated by buffers B1, B2 and B3:



Buffers and Latency Schedules

D[0]
D[1]
…
E[0]
E[1]
…
F[0]
F[1]
…

D[0]
E[0]
F[0]
D[1]
E[1]
F[1]
…

System Integration and Debugging

- Try to debug the CPU/accelerator interface separately from the accelerator core.
- Build scaffolding to test the accelerator.
- Hardware/software co-simulation can be useful.
  - Seamless: Mentor Graphics

Accelerator Case Study

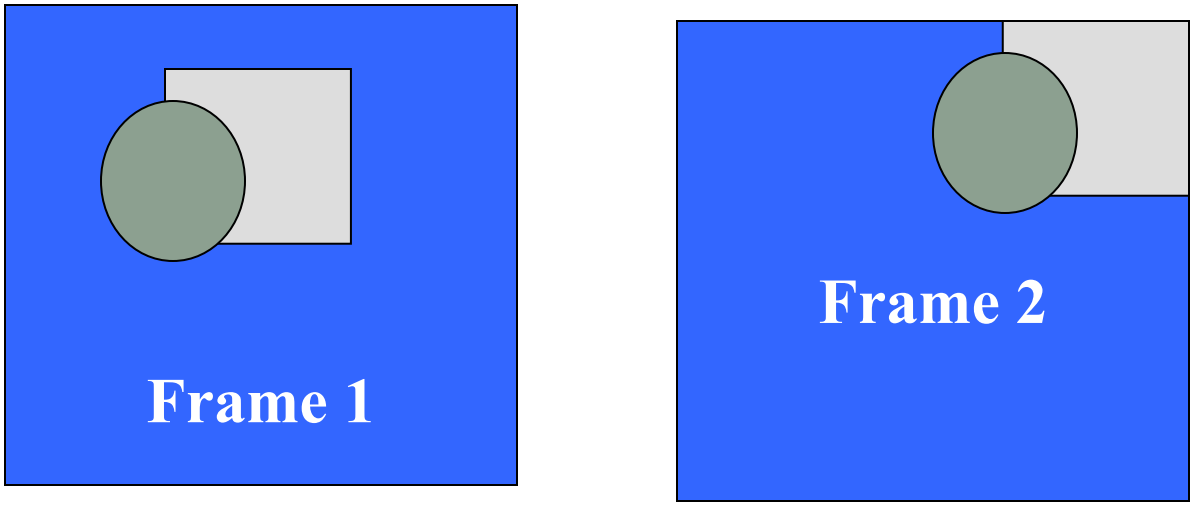
An Example:

- Video accelerator

for MPEG-4

Basics - Concept

- Build accelerator for block motion estimation, a major step in video compression.
- Perform two-dimensional correlation:



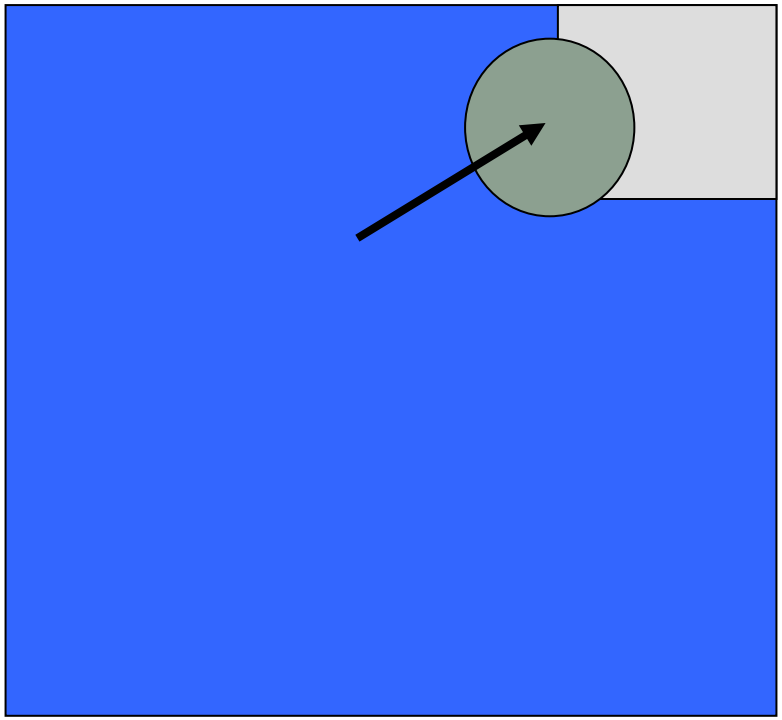
Block Motion Estimation

- MPEG divides frame into 16 x 16 macroblocks for motion estimation.
- Search for best match within a search range.
- Measure similarity with sum-of-absolute-differences (SAD):

$$\sum | M(i,j) - S(i-o_x, j-o_y) |$$

Best Match

Best match produces motion vector for the motion of an image block



Full Search Algorithm

```
bestx = 0; besty = 0;
bestsad = MAXSAD;
for (ox = - SEARCHSIZE; ox < SEARCHSIZE; ox++) {
    for (oy = -SEARCHSIZE; oy < SEARCHSIZE; oy++) {
        int result = 0;
        for (i=0; i<MBSIZE; i++) {
            for (j=0; j<MBSIZE; j++) {
             result += iabs(mb[i][j] -
                        search[i-ox+XCENTER][j-oy-YCENTER]);
            }
        }
        if (result <= bestsad) { bestsad = result;
                                 bestx = ox; besty = oy;   }
}}
```

Computational Requirements

- Let MBSIZE = 16, SEARCHSIZE = 8.
- Search area is 8+8+1 in each dimension
- Must perform:

$$n_{ops} = (16 \times 16) \times (17 \times 17) = 73984 \text{ ops}$$

- For an image of 512 X 512 pixels =>
  32 X 32 macroblocks