Chapter 7-1: Hardware/Software Co-Design

Soo-Ik Chae

Topics

- Platforms.
- Performance analysis.
- Design representations.

Embedded computing system

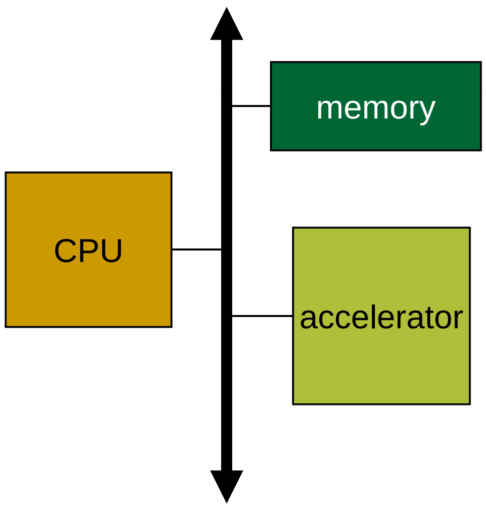
- Requirements: tight three constraints
  - Low cost
  - Low power
  - High performance
- These three constraints must be met simultaneously
- Co-design as a solution: Hardware + Software

Design platforms

- Different levels of integration:
  - PC + board: very low volume
  - Custom board with CPU + FPGA or ASIC: lower cost and lower-power.
  - Platform FPGA: more expensive than custom chips.
  - System-on-chip.

CPU/accelerator architecture

- CPU is sometimes called host.
  - Talk to the accelerator through data and control registers
  - The registers on the accelerator allows the CPU to monitor the accelerator's operation and to give it commands
- Accelerator communicate via shared memory
  - If a large volume of data is needed.
  - May use DMA to communicate.



Example: Xilinx Virtex-4

- System-on-chip:
  - FPGA fabric.
  - PowerPC.
  - On-chip RAM.
  - Specialized I/O devices.
- FPGA fabric is connected to PowerPC bus.
- MicroBlaze CPU can be added in FPGA fabric.

Virtex-5

- **SoC Challenge**:
  - Create High speed frequency design
  - Use very High speed communication links
  - Keep flexibility for modification
- **Xilinx response**:
  - FPGA provides hardware structure that enables integrated high speed design (up to 550Mhz)
  - FPGA offers integrated differential solution (LVDS) for DDR high speed communication + Hard IP Transceiver (Up to 3.2Gbps)
  - FPGA is by default the best hardware flexible solution offered through hardware reconfiguration (even partial reconfiguration)
  - FPGA can implement processor core as
    - Soft IP core (Microblaze)
    - Hard IP core (PowerPC)

Virtex 5: high Speed communication links

- Components interconnection:
  - Data width may be large and may require a huge number of IOBs
    - PCB Integrity signal

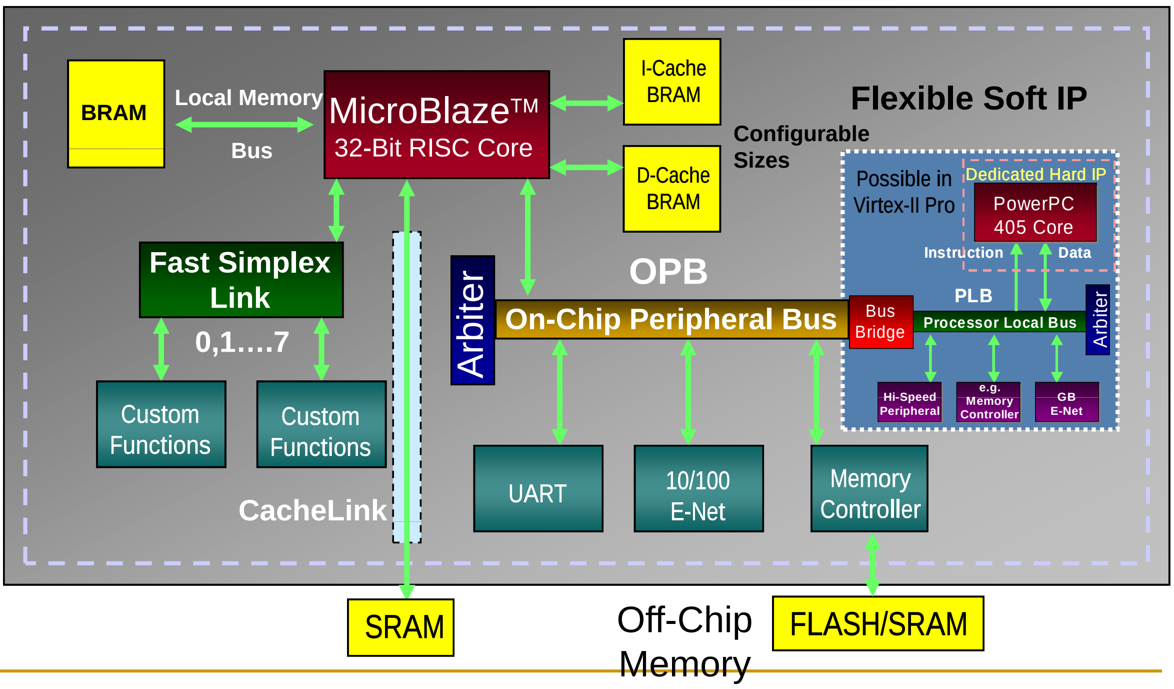
    ➔ Xilinx Sparse Chevron + LVDS
    - Power consumption

    ➔LVDS
- System communication
  - Ethernet

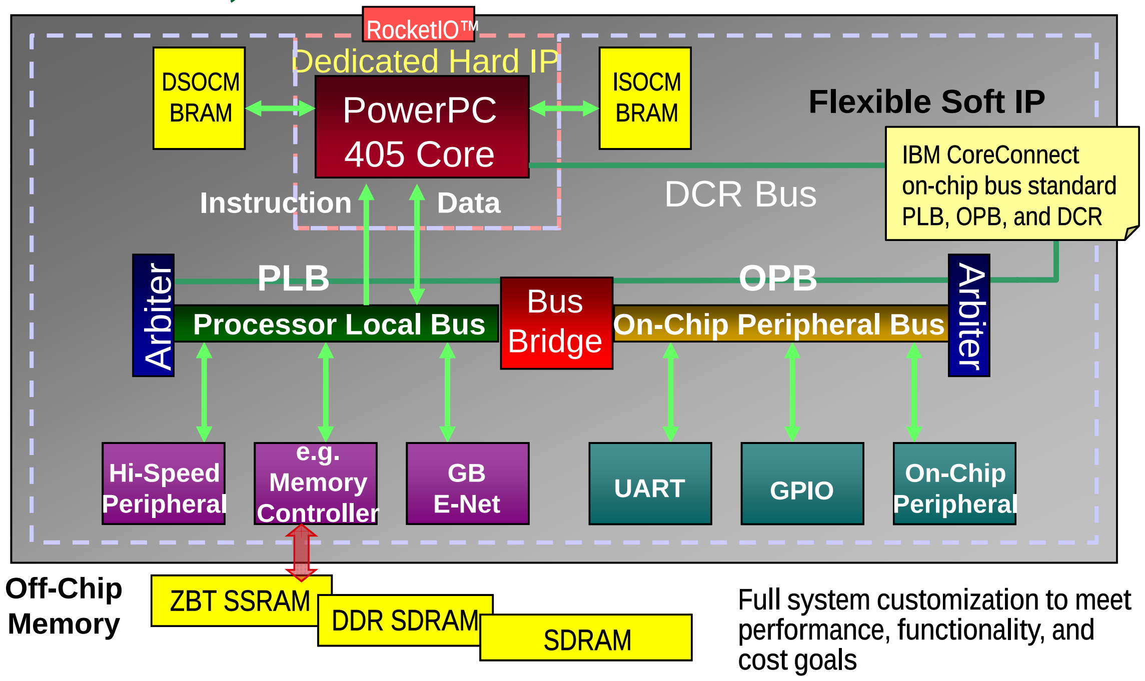
    ➔ Xilinx includes Tri-mode MAC Hard IP (10/1000/1000Mbps) in Virtex4 FX and Virtex5 LXT
  - PCI Express

    ➔ Xilinx includes PCI Express Hard IP in the newest Virtex5 LXT family.
      - PCIe x1,x2,x4 and x8

MicroBlaze-Based Embedded Design (Soft IP)



PowerPC-Based Embedded Design (Hard IP)



Full system customization to meet performance, functionality, and cost goals

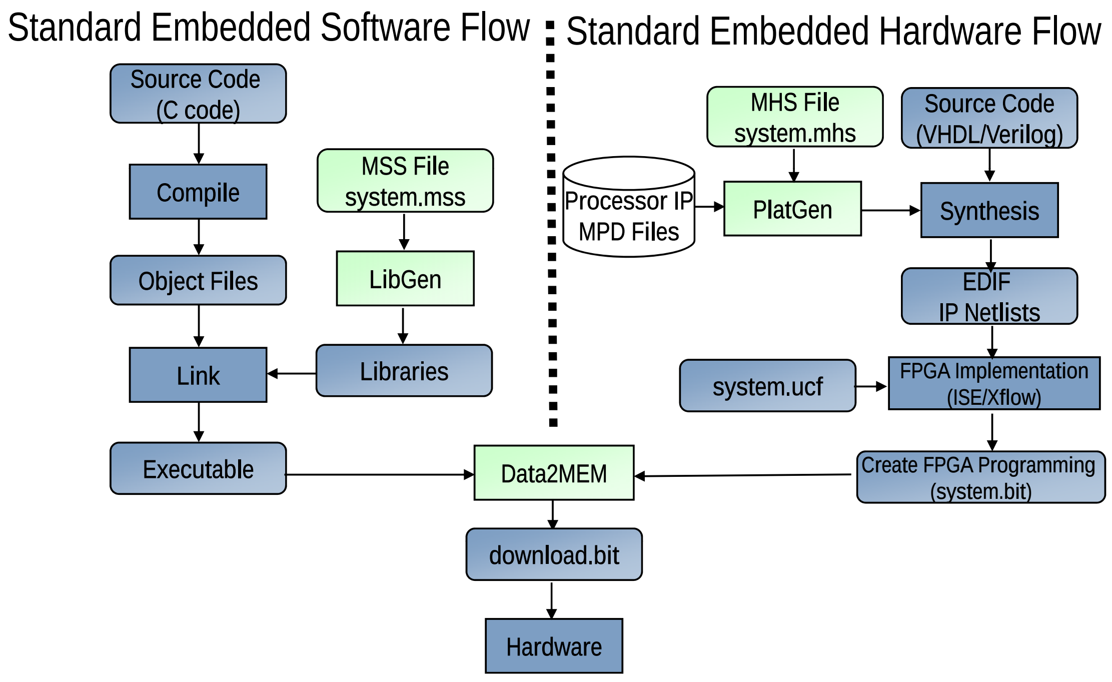
Embedded Development Kit

- What is the Embedded Development Kit (EDK)?
  - The Embedded Development Kit is the Xilinx software suite for designing complete embedded programmable systems
  - The kit includes all the tools, documentation, and IP that you require for designing systems with embedded IBM PowerPC™ hard processor cores, and/or Xilinx MicroBlaze™ soft processor cores
  - It enables the integration of both hardware and software components of an embedded system

Embedded System Tools

- GNU software development tools
  - C/C++ compiler for the MicroBlaze™ and PowerPC™ processors (gcc)
  - Debugger for the MicroBlaze and PowerPC processors (gdb)
- Hardware and software development tools
  - Base System Builder Wizard
  - Hardware netlist generation tool: PlatGen
  - Software library generation tool: LibGen
  - Simulation model generation tool: SimGen
  - Create and Import Peripheral wizard
  - Xilinx Microprocessor Debugger (XMD)
  - Hardware debugging using ChipScope™ Pro Analyzer cores
  - Eclipse IDE-based Software Development Kit (SDK)
  - Application code profiling tools
  - Virtual platform generator: VPGen
  - Flash Writer utility

Detailed EDK Design Flow



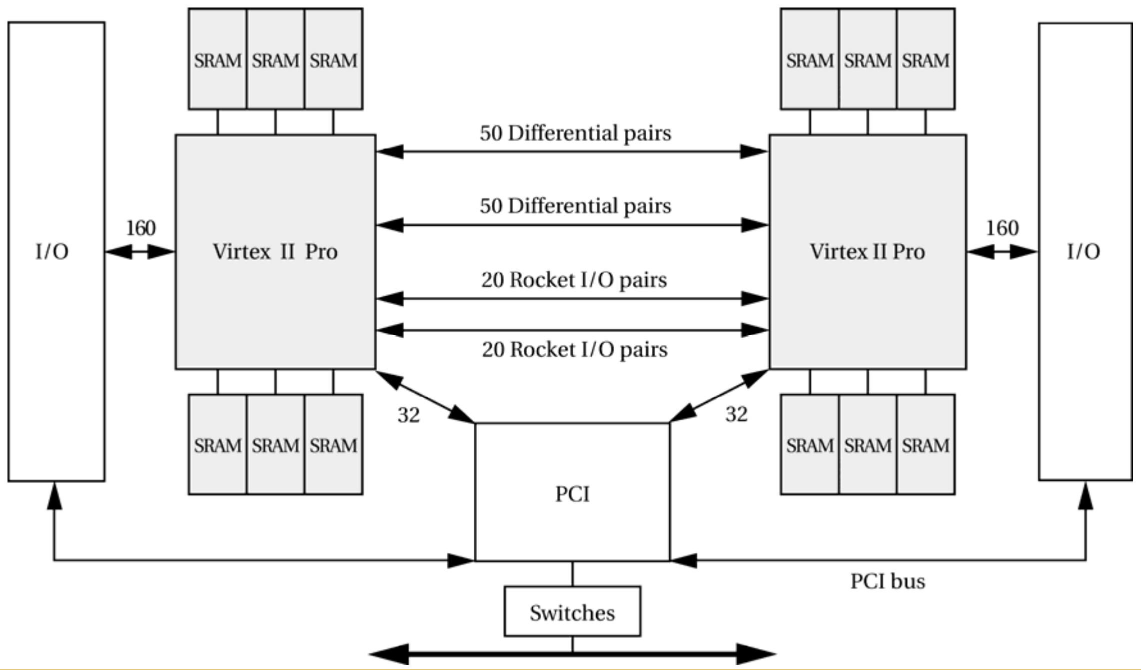
Virtex-5 LXT FPGAs

- Built on Virtex-5 LX platform 65nm ExpressFabric technology
- FPGA industry’s first built-in PCIe & Ethernet blocks
- Compliance tested at PCISIG Plugfest and UNH IOL
- Industry’s lowest power 65nm transceivers: <100mW @ 3.2Gbps
- Support for all major protocols: PCIe, GbE, XAUI, OC-48, etc.
- Six devices ranging from 30K to 330K logic cells

Virtex summary

- Depending of your Digital system, you may use Xilinx FPGA as the solution for System On Chip.
  - Today Xilinx can provide in 1 component (Virtex4 or Virtex5):
    - Embedded PowerPC 405
    - Embedded Ethernet MAC 10/100/1000
    - Embedded MAC DSP
    - Embedded High Speed Transceivers
    - Embedded PCI Express (Virtex5 only)
    - Programmable Logic Cells
    - ....

Example: WILDSTAR II Pro

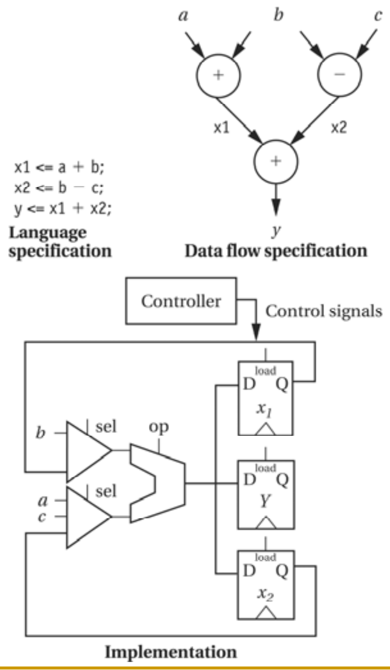


Performance analysis

- Must analyze accelerator performance to determine system speedup.
- High-level synthesis helps:
  - Use as estimator for accelerator performance.
  - Use to raise the level of abstraction for hardware designers
    - In implementing accelerator.

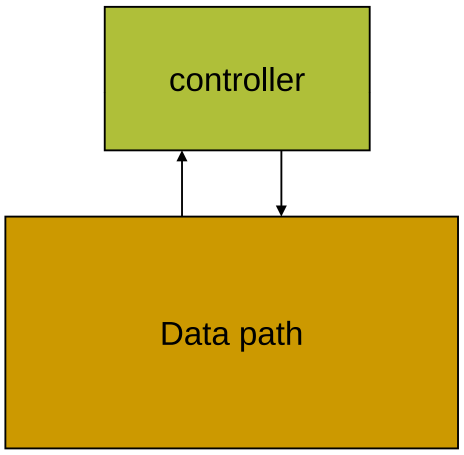
High-level synthesis

- High-level synthesis creates register-transfer description from behavioral description.
- Schedules and allocates:
  - Operators
    - Functional unit
    - On a particular clock cycle
  - Variables: registers
  - Connections: muxes
- Control step or time step is one cycle in system controller.
- Components may be selected from technology library.



Data path/controller architecture

- Data path performs regular operations, stores data in registers.
- Controller provides required sequencing.

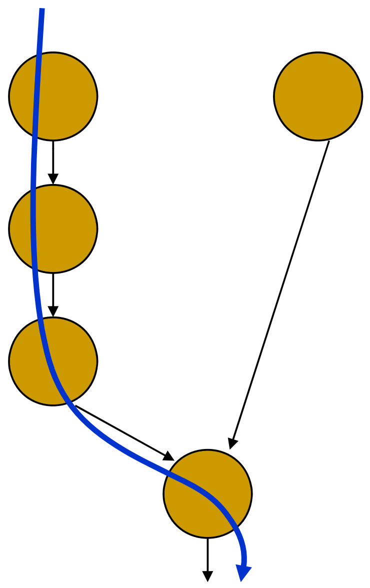


Multiplexers

- Sharing functional units and registers
- Controlled by a control FSM, which supplies the select signals to the muxes.
- In most cases, we don't need demuxes at the outputs of shared units because the hardware is generally designed to ignore values that aren't used on any given clock cycle.
- Muxes add three types of costs to the implementation
  - Delay
  - Logic
  - Wiring
- Sharing isn't always a win
  - Adders get smaller and faster by not sharing

Models

- Model computation as a data flow graph.
- Critical path is set of nodes on path that determines schedule length.



Simple example

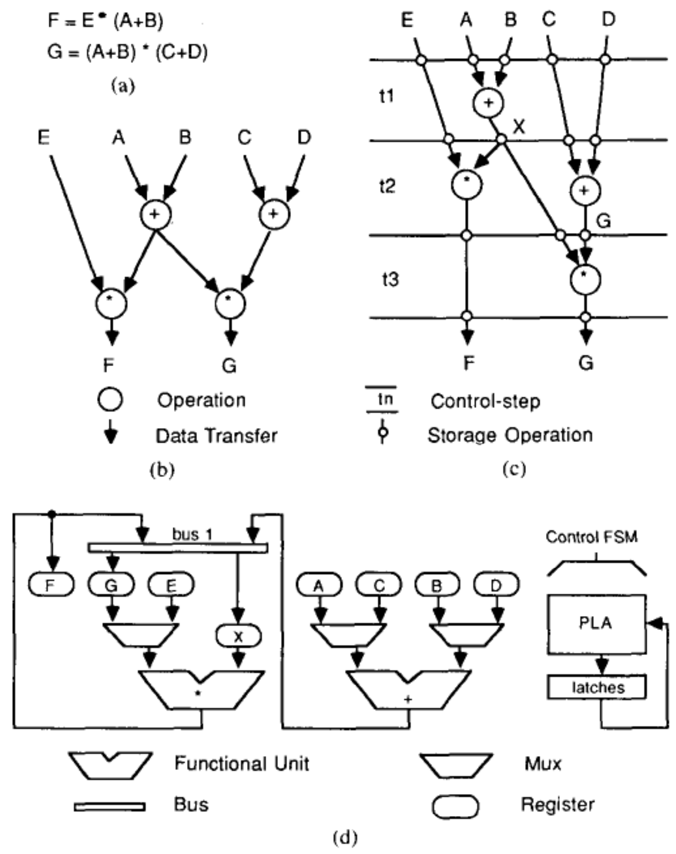
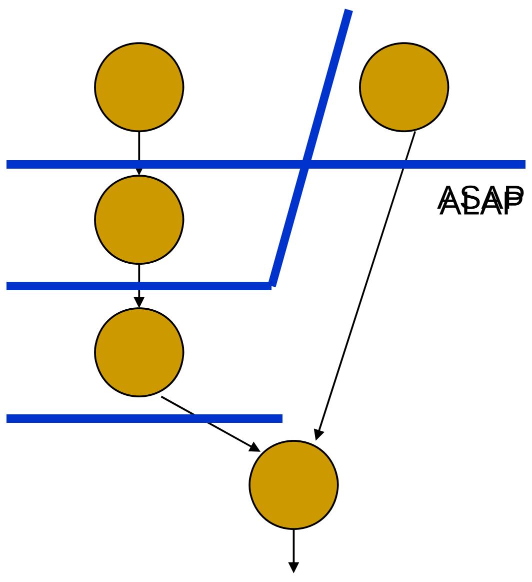


Fig. 1. Synthesis subtasks for a simple example. (a) HDL description. (b) Control data flow graph (CDFG). (c) Scheduled CDFG. (d) RTL architecture.

Schedules

- As-soon-as-possible (ASAP) pushes all nodes to start of slack region.
- As-late-as-possible (ASAP) pushes all nodes to end of slack region.
- Useful for bounding schedule length.



First-come first-served, critical path

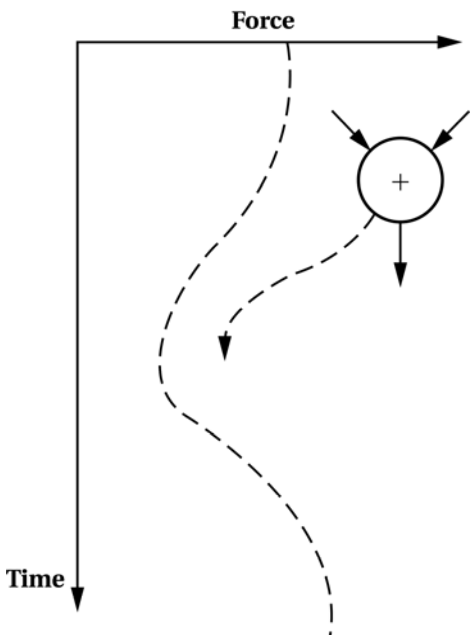
- FCFS scheduling walks through data flow graph from sources to sinks.
- Schedules each operator in first available slot based on available resources.
  - Because it chooses nodes at equal depth arbitrarily, it may delay a critical operation.
- Critical-path scheduling walks through critical nodes first.

List scheduling

- An effective heuristic that tries to improve on critical-path scheduling by providing a more balanced consideration of off-critical-path nodes.
  - Improvement on critical path scheduling.
- Estimates importance of nodes off the critical path.
  - Estimates how close a node is to being critical by measuring D, number of descendants.
  - Node with fewer descendants is less likely to become critical.
- Traverse graph from sources to sinks.
  - For nodes at a given depth, order nodes by criticality.

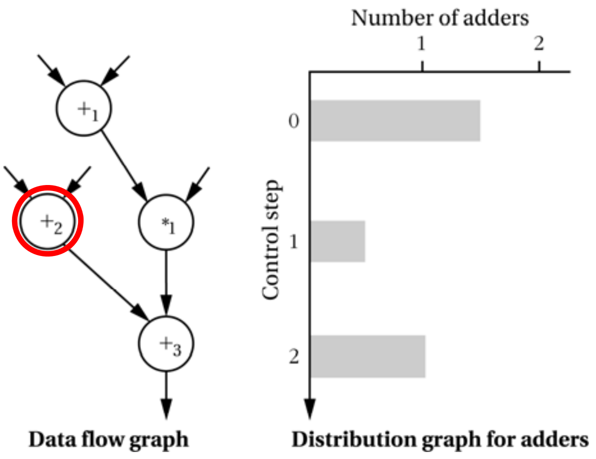
Force-directed scheduling

- Tries to minimize the hardware cost by balancing the use of functional units across cycles
- Forces model the connections to other operators.
  - Forces on operator change as schedule of related operators change.
- Forces are a linear function of displacement.
- Predecessor/successor forces relate operator to nearby operators.
- Place operator at minimum-force location in schedule.



Distribution graph

- Bound schedule using ASAP, ALAP.
- Count number of operators of a given type at each point in the schedule.
  - Weight by how likely each operator is to be at that time in the schedule.



Distribution graph

```
y" + 3xy' + 3y = 0 :

while (x < a) repeat:
        xl = x + dx ;
        ul = u - (3 * x * u * dx) - (3 * y * dx) ;
        yl = y + (u * dx) ;
        x = xl ; u = ul ; y = yl ;
end ;
```

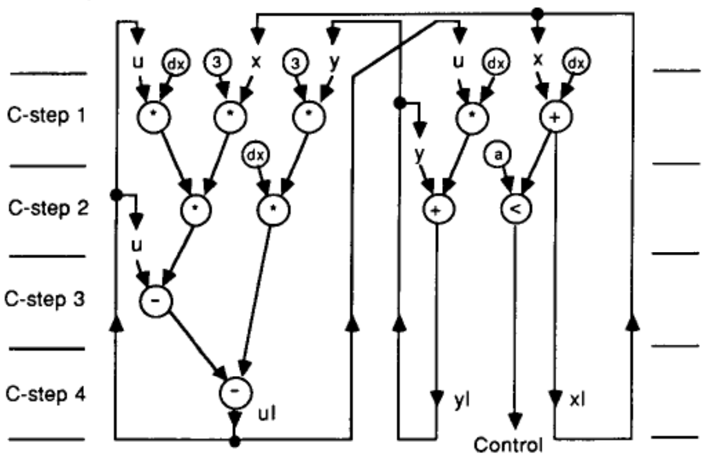


Fig. 3. Control data flow graph (CDFG) for first example.

Distribution graph

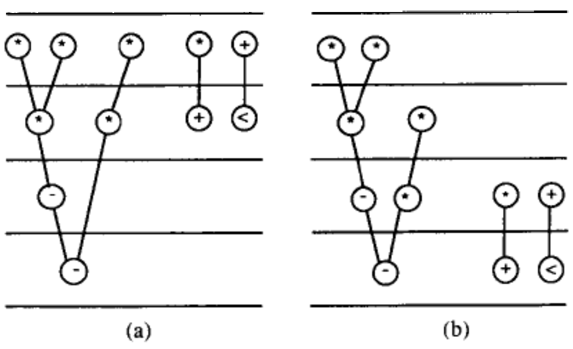


Fig. 4. (a) ASAP and (b) ALAP schedules.

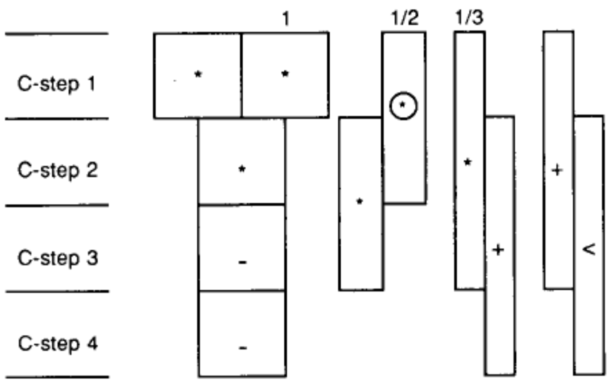


Fig. 5. Time frames of operations (initial state).

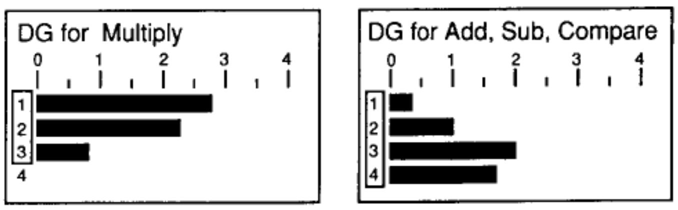


Fig. 6. Distribution graphs (initial state).

Distribution

- For each DG, the distribution in c-step i is given by
  - DG(i) = $\sum_{\text{for all operation}}$ Probability (operation, i) (1)
- We assume that each operation has a uniform probability of being assigned to any feasible control step.
- A distribution graph shows the expected value of the number of operators of a given type being assigned to each c-step.
  As shown in Figure 7, some operation in different branches are mutually exclusive
- The same function unit can be shared by those operation as they will never execute concurrently

Distribution graph

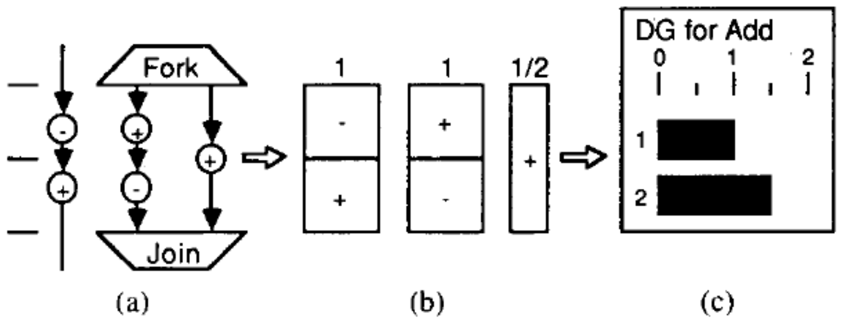


Fig. 7. Distribution evaluation for CDFG with conditional statement. (a) DFG). (b) Time frames. (c) DG.

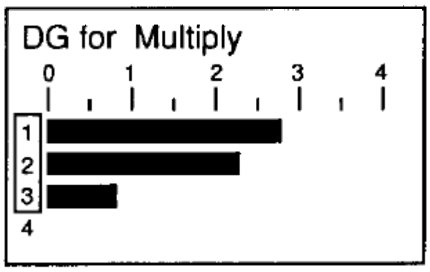
Calculation of self-forces

- Each operation of the CDFG will have a self force associated with each c-step i of its time frame.
- This is a quantity which reflects the effect of an attempted control step assignment on the overall operation concurrency.
- It is positive if the assignment causes an increase of operation concurrency, and negative for a decrease.
- The force is much like that exerted by a spring that obeys Hooke's law.
  - Force = K * x, x: displacement
- Each DG can be represented as a series of springs (one for each c-step)

Calculation of self-forces

- The constant of each spring K is given by the value of DG(i), where i is the c-step number for which the force is calculated.
- The displacement of the spring x is given by the increase (or decrease) of the probability of the operation in each c-step due to a rescheduling of the operation.
- For a given operation whose initial time frame spans c-steps t to b (t <=b), the force in c-step i is given by
  - Force(i) = DG(i) * x(i)
- The total self force associated with the assignment of an operation to c-step j (t <= j <= b)
  - Self Force(j) = $\sum_{i=t}^{b}$ Force(i) (2)

Calculation of self-forces



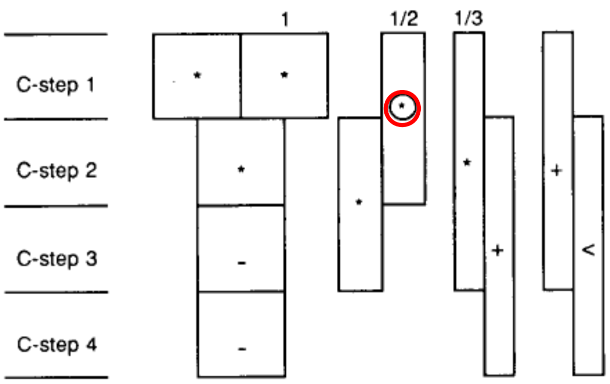


Fig. 5. Time frames of operations (initial state).

Calculation of self-forces

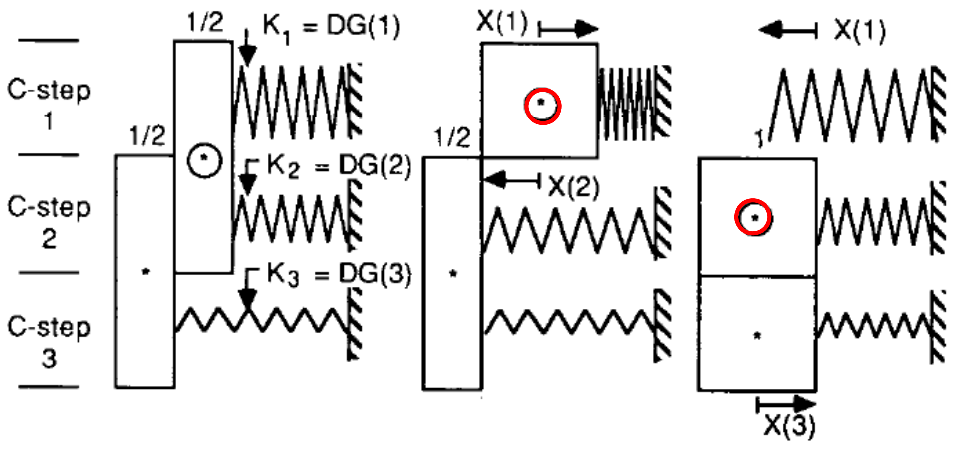


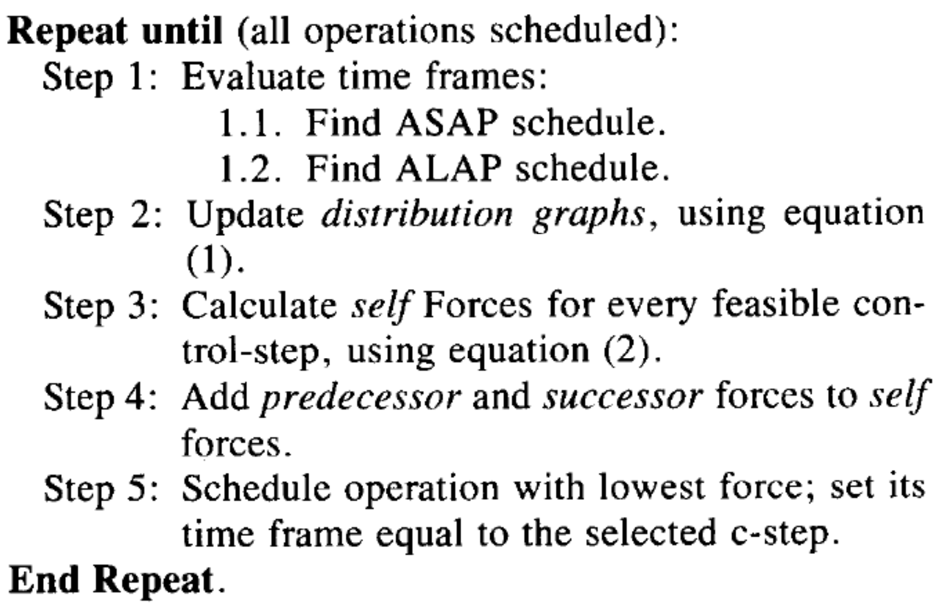
Fig. 8. Time frame modifications for force calculations.

- We will attempt to schedule the circled multiply operation in c-step 1 as depicted in Fig. 8(b)

Calculation of self-forces

- Self Force(1) = 2.833*0.5 +2.333*(-0.5)=0.25
- The force is positive
  - This will have an adverse effect on the overall distribution
- Self Force(2)= 2.833*(-0.5)+2.333*0.5= -0.25
- A modification will be propagated to the time frames of the predecessor and/or successor operations
- Predecessor forces and successor forces
- Succ Force(3) = 2.333*(-0.5) + 0.833*0.5 = -0.75
- Total Force(2)= Self Force(2) + Succ Force(3) = -1.0
  - Even better

Summary of force-directed scheduling



```
Repeat until (all operations scheduled):
  Step 1: Evaluate time frames:
          1.1. Find ASAP schedule.
          1.2. Find ALAP schedule.
  Step 2: Update distribution graphs, using equation
          (1).
  Step 3: Calculate self Forces for every feasible con-
          trol-step, using equation (2).
  Step 4: Add predecessor and successor forces to self
          forces.
  Step 5: Schedule operation with lowest force; set its
          time frame equal to the selected c-step.
End Repeat.
```

Path-based scheduling

- Minimizes the number of control states in controller.
- Schedules each path independently, then combines paths into a system schedule.
- Schedule path combinations using minimum clique covering.
  - Also known as as-fast-as-possible (AFAP) scheduling

Path-based scheduling

Abstract—In the context of synthesis, scheduling assigns operations to control steps. Operations are the atomic components used for describing behavior, for example, arithmetic and Boolean operations. They are ordered partially by data dependencies (data-flow graph) and by control constructs such as conditional branches and loops (control-flow graph). A control step usually corresponds to one state, one clock cycle, or one microprogram step. This paper presents a new, path-based scheduling algorithm. It yields solutions with the minimum number of control steps, taking into account arbitrary constraints that limit the amount of operations in each control step. The result is a finite state machine that implements the control. Although the complexity of the algorithm is proportional to the number of paths in the control-flow graph, it is shown to be practical for large examples with thousands of nodes.

Path-based scheduling

```
entity prefetch is
  port (branchpc, ibus  : in bit32;
          branch, ire      : in bit;
          ppc, popc, obus: out bit32);
end prefetch;

architecture behavior of prefetch is
begin
  process
   variable pc,oldpc : bit32 := 0;
  begin
    ppc  <= pc;            --1
    popc <= oldpc;         --2
    obus <= ibus + 4;      --3
    if (branch = '1')      --4
     then
      pc := branchpc;      --5
    end if;                --6
    wait until (ire = '1'); --7
    oldpc := pc;           --8
    pc    := pc + 4;       --9,10
  end process;
end behavior;
```

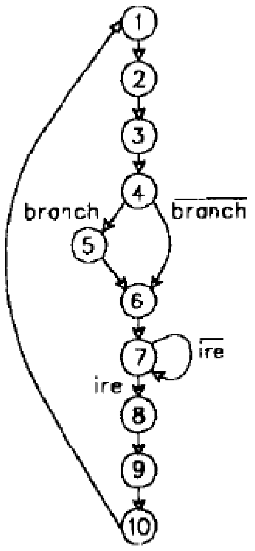


Fig. 1. Behavioral description example.

Path-based scheduling

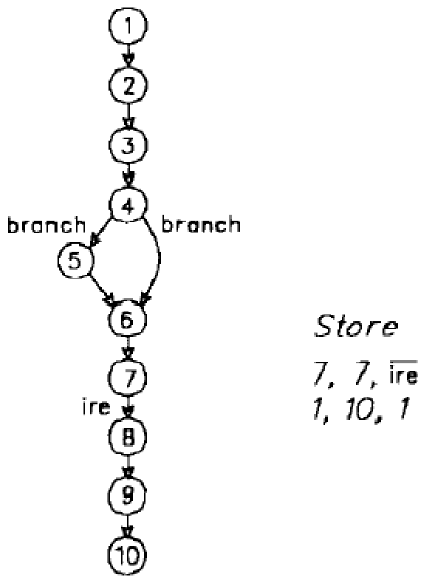


Fig. 3. Prefetch example after loop elimination.

AFAP Scheduling

In this section, the algorithm for AFAP scheduling is given. It involves keeping all paths in the control-flow graph and several *NP*-complete steps. At the end, substantial simplifications are suggested. The algorithm consists of four main steps.

1) Transforming the contol-flow graph *B* into a directed acyclic graph (DAG) and keeping lists for the loops.

2) All paths in the DAG are scheduled AFAP independently, according to the data-flow constraints in each path.

3) The schedules of step 2) are overlapped in a way that minimizes the number of control states.

4) The finite state machine for control is built.

AFAP scheduling of a path

- The idea
  - Find all longest paths
  - Then compute the constraints for each path
  - Schedule each path AFAP independently.
- A path corresponds to one possible execution sequence
- So, the number of different paths is a measure of how many different functions a design can perform.
- Although the number of paths in a graph can grow worse than exponentially, in practice we have found on the order of 1000 paths for the execution unit of a microprocessor.

Path

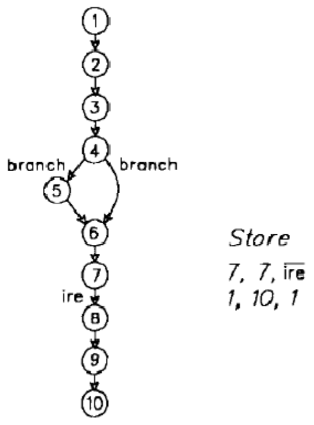


Fig. 3. Prefetch example after loop elimination.

- Path1 = {1,2,3,4,5,6,7,8,9,10}
- Path2 = {1,2,3,4,6,7,8,9,10}
- Path3 = {7,8,9,10}
- Note that path that starts at loop beginning v_F must also be considered.

Constraints

- Variables can be assigned only once in one control state.
- IO ports can be read or written only once in one control state
- Functional units can be used only once in a control state
- The maximal delay within one control state limits the number of operations that can be chained.
- The amount of storage and communication (buses, muxes) is not constrained presently.
- Obviously storage and communication can be optimized during allocation.
- Constraints are kept as sets of operation {vi}

Constraints and Interval graph

- Variables can be assigned only once in one control state.
- IO ports can be read or written only once in one control state
- Functional units can be used only once in a control state
- The maximal delay within one control state limits the number of operations that can be chained.
- The amount of storage and communication (buses, muxes) is not constrained presently.
- Obviously storage and communication can be optimized during allocation.
- Constraints are kept as sets of operation {vi}
  - If any v in {vi} is the first operation in the next state, the constraint is met.

Interval graph and Clique

- The interval graph for the set of constraints of each path is formed
- And a minimum clique covering is computed.
- In the interval graph each node corresponds to an interval and edges indicate that the corresponding tow intervals overlaps. A clique is a complete subgraph with all possible edges.
- A minimum clique covering (NP-complete in general) is a minimal number of cliques, so that each node is in one clique.

Constraints and Interval graph

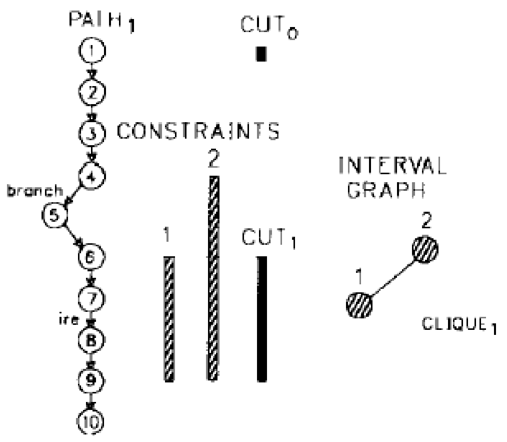


Fig. 4. Constraints and interval graph for one path in the Prefetch example.

Cut

- A cut corresponds to the set of nodes overlapped in each clique.
- It represents possible point where a state starts.
- States are ordered along the path.
- In addition, a cut of the first state is added.
- The cuts give the minimum number of control states to execute this path.
- A state starts at a cut corresponding to a clique

A minimum clique covering

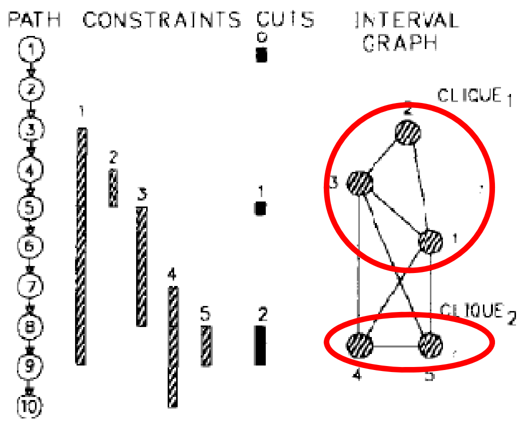


Fig. 5. Constraints and interval graph for a more complex example.

Overlapping of paths

- To find the minimum number of states for all paths, the schedule for each path must be overlapped.
- Define another graph
    - Nodes: cuts
    - Edges: join nods corresponding to overlapping cuts
- Find a minim clique of this new graph
    - Gives the minimum set of cuts that fulfills the fastest schedule for each path, and thus the minimum number of control states.

Minimum number of control states

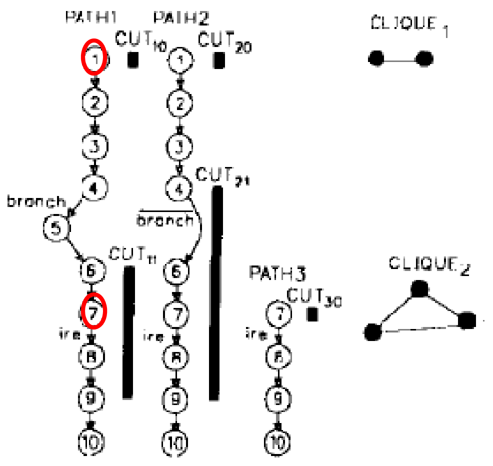


Fig. 6. Overlapping cuts for different paths.

Control finite state machine

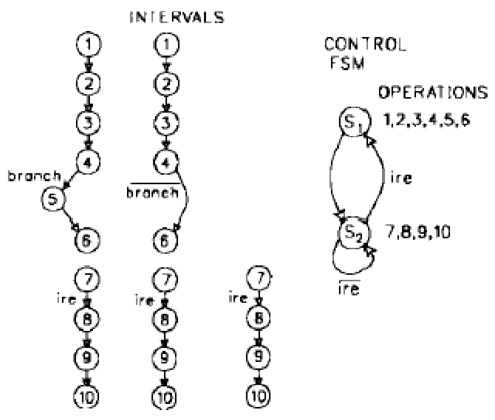


Fig. 7. Building the control finite state machine for the Prefetch example.

Buidling CFSM with no area constraint

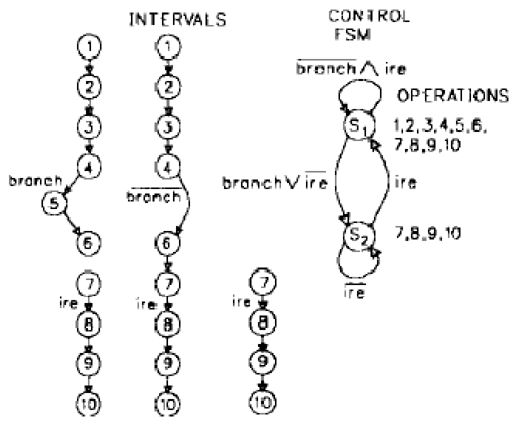


Fig. 8. Building the control finite state machine for Prefetch with no area constraints.

- Since states are mutually exclusive, scheduling one operation in more than one state.

Accelerator estimation

- Estimating the hardware cost of an accelerator requires balancing accuracy and efficiency.
- Estimation must be good enough to avoid misguiding the overall synthesis process.
- But the estimates must be generated quickly enough that co-synthesis can explore a large number of candidate designs.
- They just rely on scheduling and allocation to measure execution time and hardware size.

Accelerator estimation

- How do we use high-level synthesis, etc. to estimate the performance of an accelerator?
- We have a behavioral description of the accelerator function.
- Need an estimate of the number of clock cycles.
- Need to evaluate a large number of candidate accelerator designs.
  - Can’t afford to synthesize them all.

Estimation methods

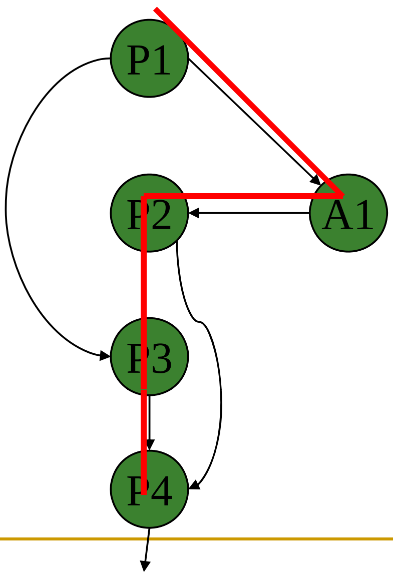
- Hermann et al. used numerical methods.
  - Estimated incremental costs due to adding blocks to the accelerator.
- Henkel and Ernst used path-based scheduling.
  - Cut CFDG into subgraphs: reduce loop iteration count; cut at large joins; divide into equal-sized pieces.
  - Schedule each subgraph independently.

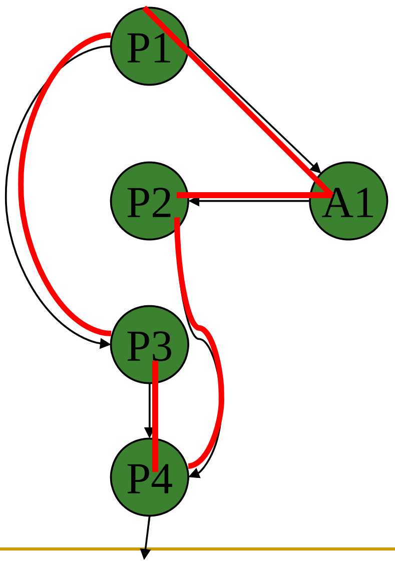
Single- vs. multi-threaded

- One critical factor is available parallelism:
  - single-threaded/blocking: CPU waits for accelerator;
  - multithreaded/non-blocking: CPU continues to execute along with accelerator.
- To multithread, CPU must have useful work to do.
  - But software must also support multithreading.

Total execution time

- Single-threaded:
- Multi-threaded:



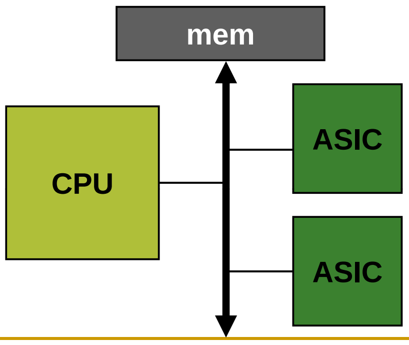


Execution time analysis

- Single-threaded:
  - Count execution time of all component processes.
- Multi-threaded:
  - Find longest path through execution.

Hardware-software partitioning

- Partitioning methods usually allow more than one ASIC.
- Typically ignore CPU memory traffic in bus utilization estimates.
- Typically assume that CPU process blocks while waiting for ASIC.

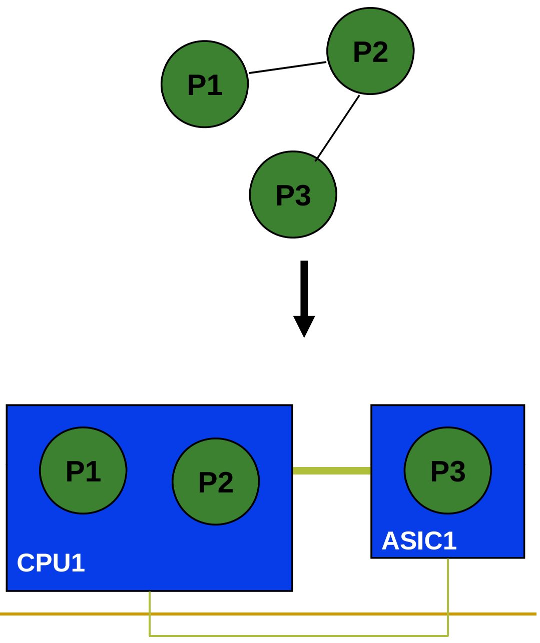


Co-design activities

- *Scheduling operations, including communication on the network and computation on the PEs*: make sure that data is available when it is needed.
- *Allocation computation to PE*: make sure that processes don’t compete for the PE.
- *Partitioning functional description into computation units*: break operations into separate processes to increase parallelism; put serial operations in one process to reduce communication.
- *Mapping*: take abstract PEs and communication links onto specific components; mapping selects specific components that can be associated with more precise cost, performance, and power

Scheduling and allocation

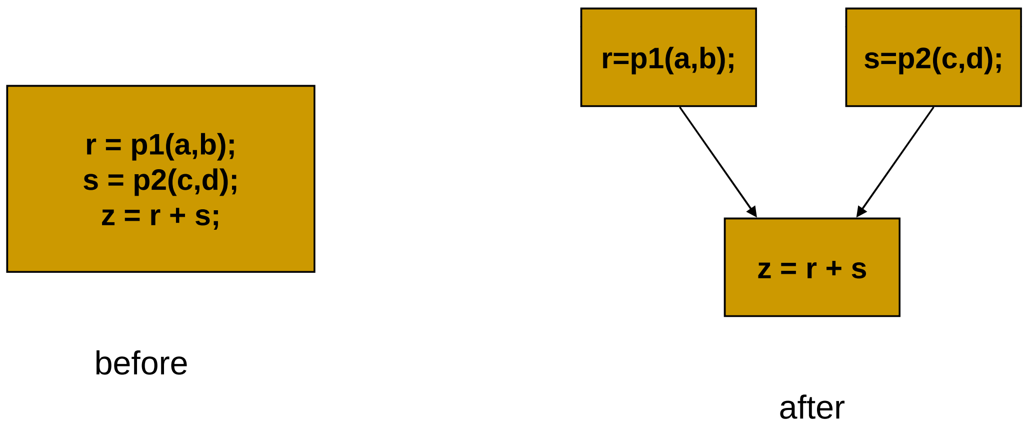
- Must schedule/allocate
  - computation
  - communication
- Performance may vary greatly with allocation choice.



Problems in scheduling/allocation

- Can multiple processes execute concurrently?
- Is the performance granularity of available components fine enough to allow efficient search of the solution space?
- Do computation and communication requirements conflict?
- How accurately can we estimate performance?
  - software
  - custom ASICs

Partitioning example



Problems in partitioning

- At what level of granularity must partitioning be performed?
- How well can you partition the system without an allocation?
- How does communication overhead figure into partitioning?

Problems in mapping

- Mapping and allocation are strongly connected when the components vary widely in performance.
- Software performance depends on bus configuration as well as CPU type.
- Mappings of PEs and communication links are closely related.

Program representations

- CDFG: single-threaded, executable, can extract some parallelism.
- Task graph: task-level parallelism, no operator-level detail.
  - TGFF (task graph for free) generates random task graphs.
- UNITY: based on parallel programming language.

Platform representations

- Technology table describes PE, channel characteristics.
  - CPU time.
  - Communication time.
  - Cost.
  - Power.
- Multiprocessor connectivity graph describes PEs, channels.

Type	Speed	cost
ARM 7	50E6	10
MIPS	50E6	8