System partitioning

- System functionality is implemented on system components
  - ASICs, processors, memories, buses
- Two design tasks:
  - **Allocate** system components or ASIC constraints
  - **Partition** functionality among components
- Constraints
  - Cost, performance, size, power
- Partitioning is a central system design task

Outline

- Structural vs. functional partitioning
- Natural vs. executable language specifications
- Basic partitioning issues and algorithms
- Functional partitioning techniques for hardware
- Hardware/software partitioning
- Functional partitioning techniques for software
- Exploring tradeoffs with functional partitioning

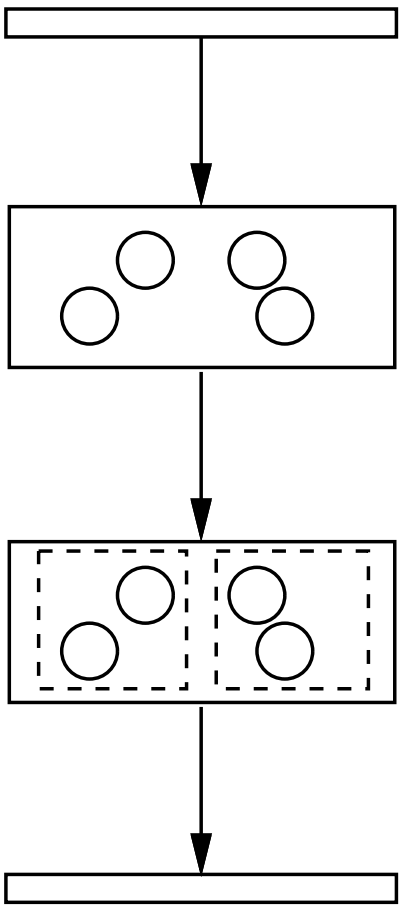
Structural vs. functional partitioning

- Structural: Implement structure, then partition
- Functional: Partition function, then implement
  - Enables better size/performance tradeoffs
  - Uses fewer objects, better for algorithms/humans
  - Permits hardware/software solutions
  - But, it's harder than graph partitioning

Natural vs. executable language specifications

- Alternative methods for specifying functionality
- Natural languages common in practice
- Executable languages becoming popular
  - Automated estimation/partitioning explores solutions
  - Early verification reduces costly late changes
  - Precision eases integration

Basic partitioning issues



Specification abstraction−level

Granularity

Metrics and estimations

Partitioning algorithms

Objective and closeness functions

System−component allocation

Output

Basic partitioning issues (cont.)

- Specification-abstraction level: input definition
  - Just indicating the language is insufficient
  - Abstraction-level indicates amount of design already done
  - e.g. task DFG, tasks, CDFG, FSMD
- Granularity: specification size in each object
  - Fine granularity yields more possible designs
  - Coarse granularity better for computation, designer interaction
  - e.g. tasks, procedures, statement blocks, statements
- Component allocation: types and numbers
  - e.g. ASICs, processors, memories, buses
- Output: format and uses
  - e.g. new specification, hints to synthesis tool

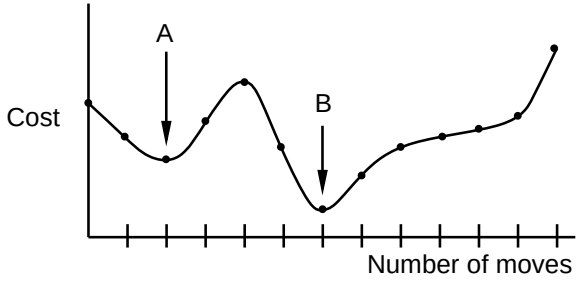
Basic partitioning issues (cont.)

- Metrics and estimations: "good" partition attributes
  - e.g. cost, speed, power, size, pins, testability, reliability
  - Estimates derived from quick, rough implementation
  - Speed and accuracy are competing goals of estimation

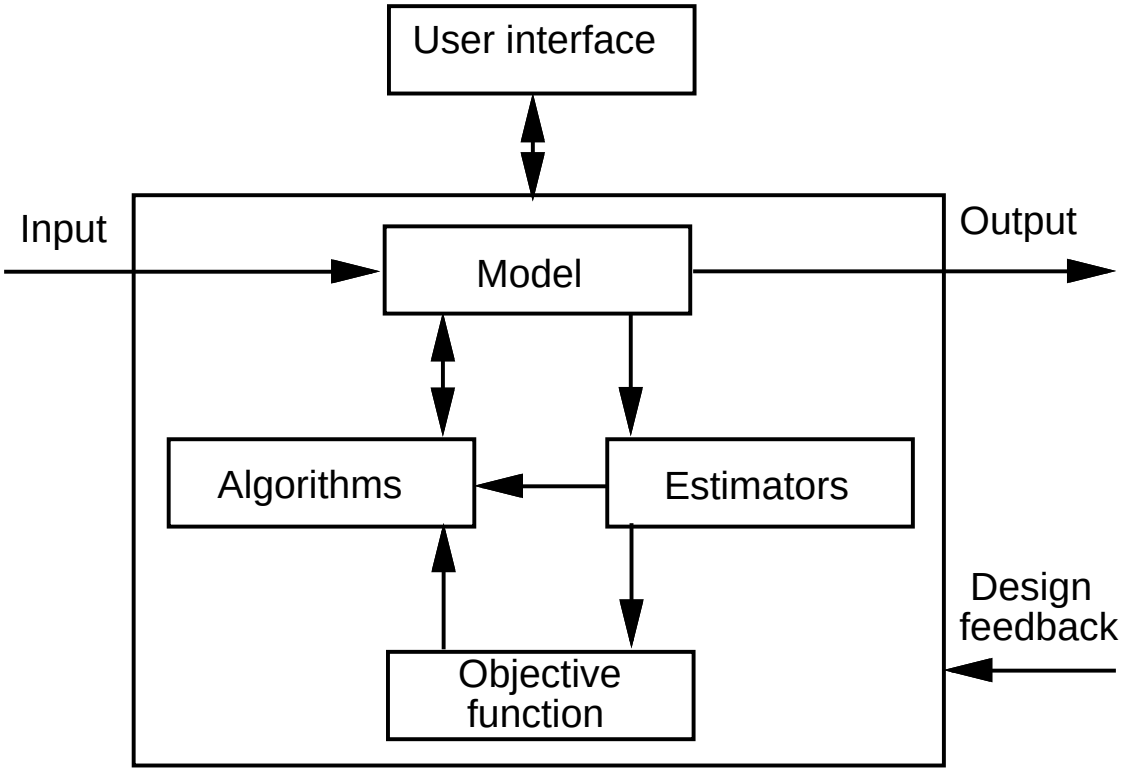
- Objective and closeness functions
  - Combines multiple metric values
  - Closeness used for grouping before complete partition
  - Weighted sum common
  - e.g. $k_1 F(area, c) + k_2 F(delay, c) + k_3 F(power, c)$

Basic partitioning issues (cont.)

- Algorithms: control strategies seeking best partition
  - Constructive creates partition
  - Iterative improves partition
  - Key is to escape local minimum



Typical partitioning-system configuration



Basic partitioning algorithms

- Clustering and multi-stage clustering [Joh67, LT91]
- Group migration (a.k.a. min-cut or Kernighan/Lin) [KL70, FM82]
- Ratio cut [KC91]
- Simulated annealing [KGV83]
- Genetic evolution
- Integer linear programming

Hierarchical clustering

- Constructive algorithm using closeness metrics
- Overview
  - Groups closest objects
  - Recomputes closenesses
  - Repeats until termination condition met
- Cluster tree maintains history of merges
  - Cutline across the tree defines a partition

Hierarchical clustering algorithm

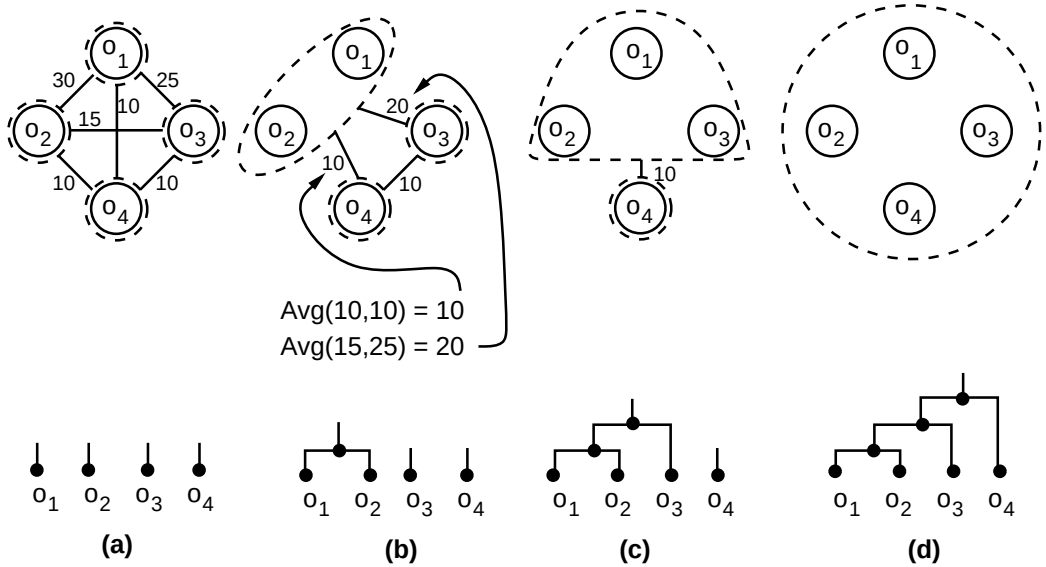
```
/* Initialize each object as a group */
for each o_i loop
    p_i = o_i
    P = P ∪ p_i
end loop

/* Compute closenesses between objects */
for each p_i loop
    for each p_j loop
        c_{i,j} = ComputeCloseness(p_i, p_j)
    end loop
end loop

/* Merge closest objects and recompute closenesses
*/
while not Terminate(P) loop
    p_i, p_j = FindClosestObjects(P, C)
    P = P − p_i − p_j ∪ p_ij
    for each p_k loop
        c_{ij,k} = ComputeCloseness(p_ij, p_k)
    end loop
end loop

return P
```

Hierarchical clustering example



Simulated annealing

- Iterative algorithm modeled after physical annealing process
- Overview
  - Starts with initial partition and temperature
  - Slowly decreases temperature
  - For each temperature, generates random moves
  - Accepts any move that improves cost
  - Accepts some bad moves, less likely at low temperatures
- Results and complexity depend on temperature decrease rate

Simulated annealing algorithm

```
temp = initial temperature
cost = Objfct(P)
while not Frozen loop
   while not Equilibrium loop
      P_tentative = Move(P)
      cost_tentative = Objfct(P_tentative)
      cost = cost_tentative − cost
      if (Accept(cost, temp) > Random(0, 1)) then
         P = P_tentative
         cost = cost_tentative
      end if
   end loop
   temp = DecreaseTemp(temp)
end loop
```

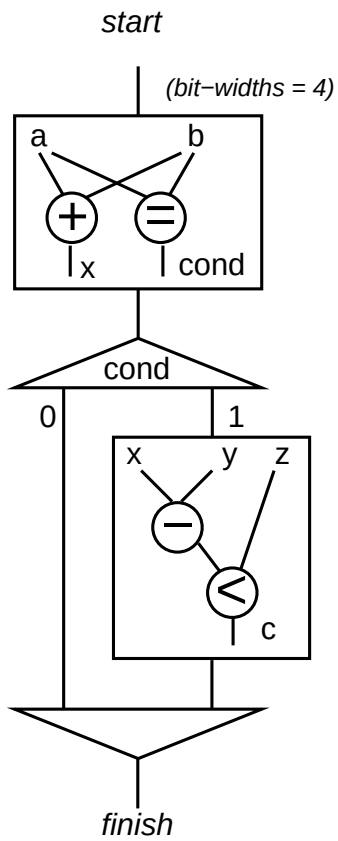
where: $Accept(cost, temp) = min(1, e^{-\frac{cost}{temp}})$

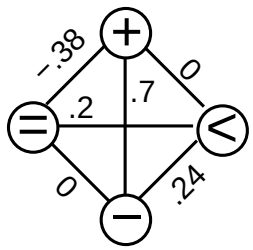
Functional partitioning for hardware: BUD

- Goal: incorporate area/time into synthesis [MK90]
- Clusters CDFG operations into datapath modules
- Closeness metrics:
  - Interconnecting wires
  - Concurrency
  - Shared hardware
- Each clustering corresponds to an allocation/scheduling
- Selects clustering with best area/time

BUD example

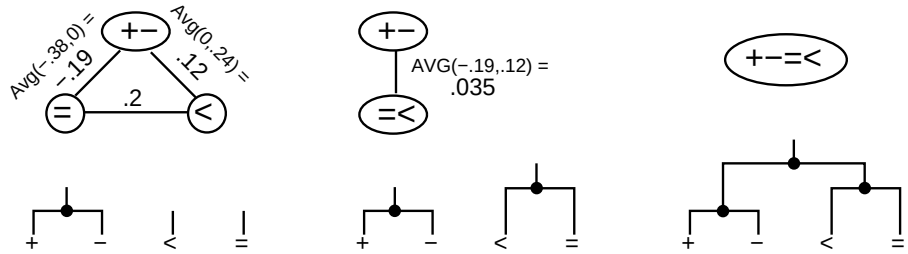
```
x := a + b;
if (a = b)
   c := ((x − y) < z);
```





(a) **(b)** **(c)**

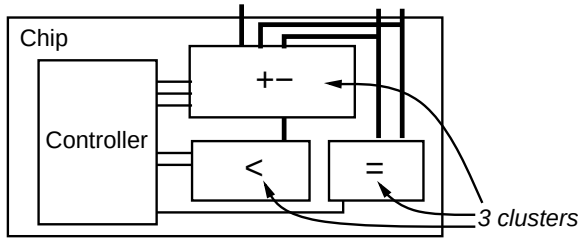
BUD example (cont.)



(a)

Clusters	Chip area A	Expected cycle time T	Objfct = AxT
+−=<	17.5	36	630
+−, =<	15.8	26	411
+−, =, <	13.8	26	359 (best)
+, −, =, <	16.4	26	426

(b)

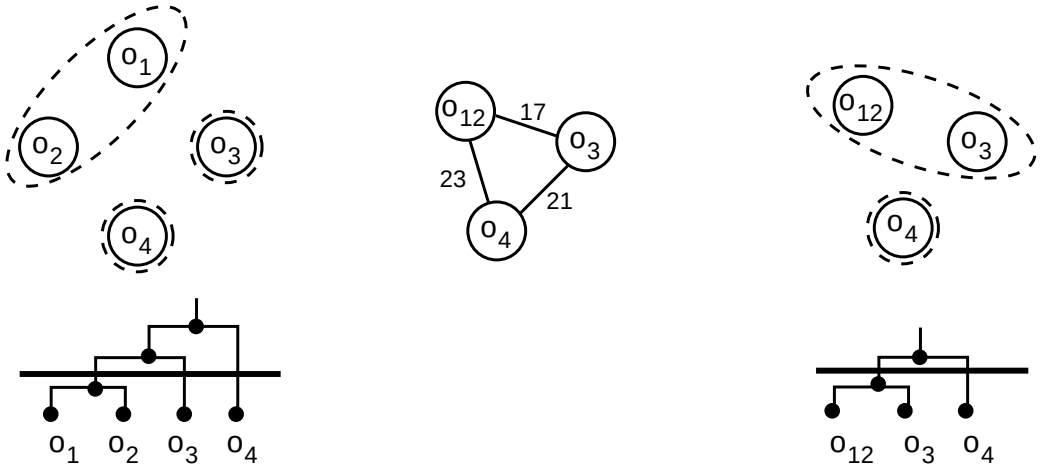


(c)

Functional partitioning for hardware: Aparty

- Extends BUD clustering to multiple stages [LT91]
  - Different closeness metrics for each stage
- Closeness metrics:
  - Control transfer reduction
  - Data transfer reduction
  - Hardware sharing

Aparty example



(a) **(b)** **(c)**

Hardware/software partitioning

- Combined hardware/software systems are common
- Software is cheap, modi able,and quick to design
- Hardware is fast
- Special algorithms are needed to favor software
- Proposed algorithms
  - Greedy [GD92]
  - Hill climbing [EHB94]
  - Binary-constraint search with hill climbing [VGG93]

Functional partitioning for systems: Vulcan, Cosyma

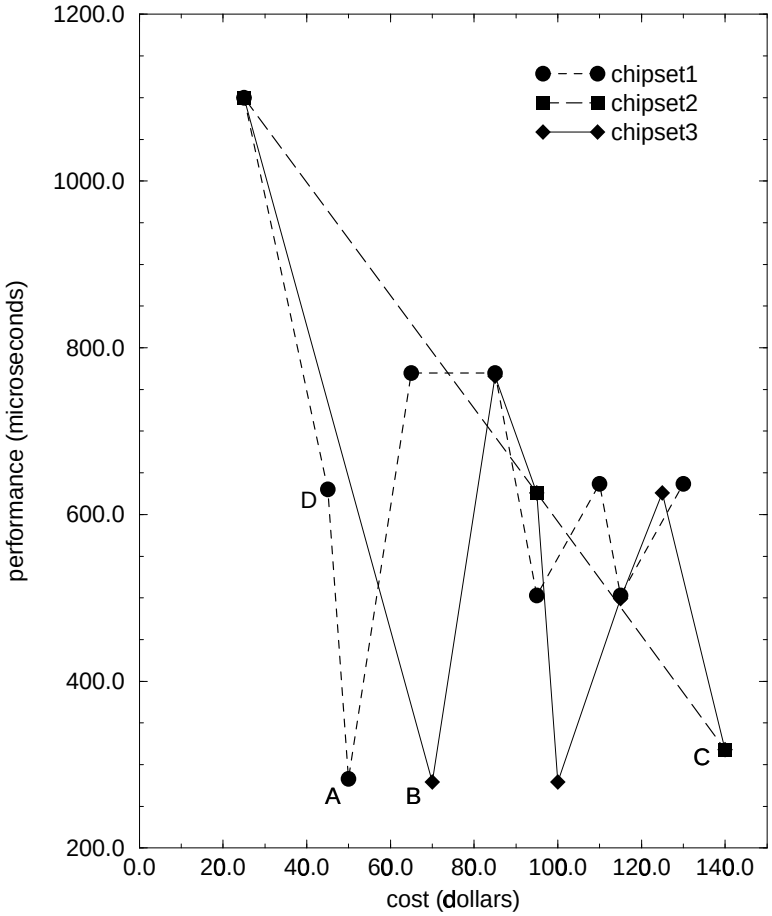
- Vulcan [GD90]I
  - Partitions CDFG operations among hardware only
  - Group migration and simulated annealing algorithms
- Vulcan II [GD93]
  - Partitions operations among hardware/software
  - Architecture: processor, hardware, memory, bus
  - All communication through memory
  - Uses greedy algorithm, extracts behaviors from hardware
- Cosyma [EHB94]
  - Partitions statement blocks among hardware/software
  - Architecture: processor, hardware, memory, bus
  - Simulated annealing, extracts behaviors from software

Functional partitioning for systems: SpecSyn

- Solves three partitioning problems
  - Behaviors to processors/ASICs
  - Variables to memories
  - Communication channels to buses
- Uses fast incremental-update estimators
- Covers both hardware and hardware/software partitioning [GVN94, VG92]

Exploring tradeoffs with functional partitioning

- Each line represents a different vendor's chip set
- Each point represents an allocation and partition
- Many designs quickly examined



UC Irvine

Summary

- Partitioning heavily influences design quality
- Functional partitioning is necessary
- Executable specification enables:
  - Automation
  - Exploration
  - Documentation
- Variety of algorithms exist
- Variety of techniques exist for different applications

Future directions

- Metrics from real design to guide partitioning
- Comparison of functional partitioning algorithms
- Impact of metric selections and orderings
- Impact of of granularity on partition quality
- Exploitation of regularity in partitioning