Partitioning Methods

Outline

- Introduction to Hardware-Software Codesign
- Models, Architectures, Languages
- Partitioning Methods
- Design Quality Estimation
- Specification Refinement
- Co-synthesis Techniques
- Function-Architecture Codesign Paradigm
- Codesign Case Studies
  - ATM Virtual Private Network
  - Digital Camera with JPEG
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

Hardware/Software Partitioning

- Informal Definition
  - The process of deciding, for each subsystem, whether the required functionality is more advantageously implemented in hardware or software

- Goal
  - To achieve a partition that will give us the required performance within the overall system requirements (in size, weight, power, cost, etc.)

- This is a multivariate optimization problem that when automated, is an NP-hard problem
**HW/SW Partitioning**

**Formal Definition**

- A hardware/software partition is defined using two sets $H$ and $S$, where $H \subseteq O$, $S \subseteq O$, $H \cup S = O$, $H \cap S = \emptyset$.

- Associated metrics:
  - $\text{Hsize}(H)$ is the size of the hardware needed to implement the functions in $H$ (e.g., number of transistors).
  - $\text{Performance}(G)$ is the total execution time for the group of functions in $G$ for a given partition $\{H,S\}$.
  - A set of performance constraints, $\text{Cons} = (C_1, \ldots, C_m)$, where $C_j = (G, \text{timecon})$, indicates the maximum execution time allowed for all the functions in group $G$ and $G \subseteq O$.

**Performance Satisfying Partition**

- A performance satisfying partition is one for which $\text{performance}(C_j,G) \leq C_j.\text{timecon}$, for all $j=1\ldots m$.

- Given $O$ and $\text{Cons}$, the hardware/software partitioning problem is to find a performance satisfying partition $\{H,S\}$ such that $\text{Hsize}(H)$ is minimized.

- The all-hardware size of $O$ is defined as the size of an all hardware partition (i.e., $\text{Hsize}(O)$).
**HW/SW Partitioning Issues**

- Partitioning into hardware and software affects overall system cost and performance

- Hardware implementation
  - Provides higher performance via hardware speeds and parallel execution of operations
  - Incurs additional expense of fabricating ASICs

- Software implementation
  - May run on high-performance processors at low cost (due to high-volume production)
  - Incurs high cost of developing and maintaining (complex) software

**Structural vs. Functional Partitioning**

- Structural: Implement structure, then partition
  - Good for the hardware (size & pin) estimation.
  - Size/performance tradeoffs are difficult.
  - Suffer for large possible number of objects.
  - Difficult for HW/SW tradeoff.

- 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
Partitioning Approaches

- Start with all functionality in software and move portions into hardware which are time-critical and can not be allocated to software (software-oriented partitioning)

- Start with all functionality in hardware and move portions into software implementation (hardware-oriented partitioning)

System Partitioning
(Functional Partitioning)

- System partitioning in the context of hardware/software codesign is also referred to as functional partitioning
- Partitioning functional objects among system components is done as follows
  - The system’s functionality is described as collection of indivisible functional objects
  - Each system component’s functionality is implemented in either hardware or software
- An important advantage of functional partitioning is that it allows hardware/software solutions
**Binding Software to Hardware**

- Binding: assigning software to hardware components
- After parallel implementation of assigned modules, all design threads are joined for system integration
  - Early binding commits a design process to a certain course
  - Late binding, on the other hand, provides greater flexibility for last minute changes

**Hardware/Software System Architecture Trends**

- Some operations in special-purpose hardware
  - Generally take the form of a coprocessor communicating with the CPU over its bus
    - Computation must be long enough to compensate for the communication overhead
  - May be implemented totally in hardware to avoid instruction interpretation overhead
    - Utilize high-level synthesis algorithms to generate a register transfer implementation from a behavior description
- Partitioning algorithms are closely related to the process scheduling model used for the software side of the implementation
Basic Partitioning Issues

- Specification-abstraction level: input definition
  - Executable languages becoming a requirement
    • Although natural languages common in practice.
  - 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

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. k1F(area,c)+k2F(delay,c)+k3F(power,c)

- Output: format and uses
  - e.g. new specification, hints to synthesis tool

- Flow of control and designer interaction
Issues in Partitioning (Cont.)

- High Level Abstraction
- Decomposition of functional objects
  - Metrics and estimations
  - Partitioning algorithms
  - Objective and closeness functions
- Component allocation
- Output

Specification Abstraction Levels

- Task-level dataflow graph
  - A Dataflow graph where each operation represents a task
- Task
  - Each task is described as a sequential program
- Arithmetic-level dataflow graph
  - A Dataflow graph of arithmetic operations along with some control operations
  - The most common model used in the partitioning techniques
- Finite state machine (FSM) with datapath
  - A finite state machine, with possibly complex expressions being computed in a state or during a transition
**Specification Abstraction Levels (Cont.)**

- **Register transfers**
  - The transfers between registers for each machine state are described

- **Structure**
  - A structural interconnection of physical components
  - Often called a net-list

---

**Granularity Issues in Partitioning**

- The granularity of the decomposition is a measure of the size of the specification in each object
- The specification is first decomposed into functional objects, which are then partitioned among system components
  - Coarse granularity means that each object contains a large amount of the specification.
  - Fine granularity means that each object contains only a small amount of the specification
    - Many more objects
    - More possible partitions
      - Better optimizations can be achieved
System Component Allocation

- The process of choosing system component types from among those allowed, and selecting a number of each to use in a given design
- The set of selected components is called an allocation
  - Various allocations can be used to implement a specification, each differing primarily in monetary cost and performance
  - Allocation is typically done manually or in conjunction with a partitioning algorithm
- A partitioning technique must designate the types of system components to which functional objects can be mapped
  - ASICs, memories, etc.

Metrics and Estimations Issues

- A technique must define the attributes of a partition that determine its quality
  - Such attributes are called metrics
    - Examples include monetary cost, execution time, communication bit-rates, power consumption, area, pins, testability, reliability, program size, data size, and memory size
    - Closeness metrics are used to predict the benefit of grouping any two objects
- Need to compute a metric’s value
  - Because all metrics are defined in terms of the structure (or software) that implements the functional objects, it is difficult to compute costs as no such implementation exists during partitioning
**Metrics in HW/SW Partitioning**

- Two key metrics are used in hardware/software partitioning
  - Performance: Generally improved by moving objects to hardware
  - Hardware size: Hardware size is generally improved by moving objects out of hardware

**Computation of Metrics**

- Two approaches to computing metrics
  - Creating a detailed implementation
    - Produces accurate metric values
    - Impractical as it requires too much time
  - Creating a rough implementation
    - Includes the major register transfer components of a design
    - Skips details such as precise routing or optimized logic, which require much design time
    - Determining metric values from a rough implementation is called estimation
Estimation of Partitioning Metrics

- Deterministic estimation techniques
  - Can be used only with a fully specified model with all data dependencies removed and all component costs known
  - Result in very good partitions
- Statistical estimation techniques
  - Used when the model is not fully specified
  - Based on the analysis of similar systems and certain design parameters
- Profiling techniques
  - Examine control flow and data flow within an architecture to determine computationally expensive parts which are better realized in hardware

Objective and Closeness Functions

- Multiple metrics, such as cost, power, and performance are weighed against one another
  - An expression combining multiple metric values into a single value that defines the quality of a partition is called an Objective Function
  - The value returned by such a function is called cost
  - Because many metrics may be of varying importance, a weighted sum objective function is used
    - e.g., Objfct = k1 * area + k2 * delay + k3 * power
  - Because constraints always exist on each design, they must be taken into account
    - e.g, Objfct = k1 * F(area, area_constr) + k2 * F(delay, delay_constr) + k3 * F(power, power_constr)
**Partitioning Algorithm Issues**

- Given a set of functional objects and a set of system components, a partitioning algorithm searches for the best partition, which is the one with the lowest cost, as computed by an objective function.
- While the best partition can be found through exhaustive search, this method is impractical because of the inordinate amount of computation and time required.
- The essence of a partitioning algorithm is the manner in which it chooses the subset of all possible partitions to examine.

**Partitioning Algorithm Classes**

- **Constructive algorithms**
  - Group objects into a complete partition
  - Use closeness metrics to group objects, hoping for a good partition
  - Spend computation time constructing a small number of partitions

- **Iterative algorithms**
  - Modify a complete partition in the hope that such modifications will improve the partition
  - Use an objective function to evaluate each partition
  - Yield more accurate evaluations than closeness functions used by constructive algorithms

- In practice, a combination of constructive and iterative algorithms is often employed.
Iterative Partitioning Algorithms

- The computation time in an iterative algorithm is spent evaluating large numbers of partitions.
- Iterative algorithms differ from one another primarily in the ways in which they modify the partition and in which they accept or reject bad modifications.
- The goal is to find global minimum while performing as little computation as possible.

![Graph showing A, B - Local minima, C - Global minimum]

Iterative Partitioning Algorithms (Cont.)

- Greedy algorithms
  - Only accept moves that decrease cost
  - Can get trapped in local minima

- Hill-climbing algorithms
  - Allow moves in directions increasing cost (retracing)
    - Through use of stochastic functions
  - Can escape local minima
  - E.g., simulated annealing
Typical partitioning-system configuration

![Diagram](image)

Basic partitioning algorithms

- Random mapping
  - Only used for the creation of the initial partition.
- Clustering and multi-stage clustering
- Group migration (a.k.a. min-cut or Kernighan/Lin)
- Ratio cut
- Simulated annealing
- Genetic evolution
- Integer linear programming
Hierarchical clustering

- One of constructive algorithm based on closeness metrics to group objects

- Fundamental steps:
  - Groups closest objects
  - Recompute closenesses
  - Repeat 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 oi loop
  pi=oi
  P=P∪pi
end loop

/* Compute closenesses between objects */
for each pi loop
  for each pj loop
    ci,j=ComputeCloseness(pi,pj)
  end loop
end loop

/* Merge closest objects and recompute closenesses */
While not Terminate(P) loop
  pi,pj=FindClosestObjects(P,C)
  P=P∪pi−pj∪upj
  for each pk loop
    ci,j,k=ComputeCloseness(pij,pk)
  end loop
end loop
return P
Hierarchical clustering example

Greedy partitioning for HW/SW partition

- Two-way partition algorithm between the groups of HW and SW.
- Suffer from local minimum problem.

Repeat
    \[ P_{\text{orig}} = P \]
    for \( i \) in 1 to \( n \) loop
        if \( \text{Objfct}(\text{Move}(P,o)) < \text{Objfct}(P) \) then
            \( P = \text{Move}(P,o) \)
        end if
    end loop
Until \( P = P_{\text{orig}} \)
**Multi-stage clustering**

- Start hierarchical clustering with one metric and then continue with another metric.
- Each clustering with a particular metric is called a stage.

**Group migration**

- Another iteration improvement algorithm extended from two-way partitioning algorithm that suffer from local minimum problem.
- The movement of objects between groups depends on if it produces the greatest decrease or the smallest increase in cost.
  - To prevent an infinite loop in the algorithm, each object can only be moved once.
**Group migration’s Algorithm**

\[ P = P_{\text{in}} \]

Loop

/*Initialize*/

prev\_P = P

prev\_cost = Objfct(P)

bestpart\_cost = ∞

for each object loop

\( o_{i} \cdot \text{moved} = \text{false} \)

end loop

/*create a sequence of n moves*/

for \( i \) in 1 to \( n \) loop

bestmove\_cost = ∞

for each object not \( o_{i} \cdot \text{moved} \) loop

\( \text{cost} = \text{Objfct}(\text{Move}(P, o_{i})) \)

if \( \text{cost} < \text{bestmove\_cost} \) then

bestmove\_cost = \( \text{cost} \)

bestmove\_obj = \( o_{i} \)

end if

end loop

\( P = \text{Move}(P, \text{bestmove\_obj}) \)

bestmove\_obj.moved = true

/*Save the best partition during the sequence*/

if \( \text{bestmove\_cost} < \text{bestpart\_cost} \) then

bestpart\_P = P

bestpart\_cost = bestmove\_cost

end if

end loop

/*Update P if a better cost was found, else exit*/

If \( \text{bestpart\_cost} < \text{prev\_cost} \) then

\( P = \text{bestpart\_P} \)

else return prev\_P

end if

end loop

**Ratio Cut**

- A constructive algorithm that groups objects until a terminal condition has been met.
- A new metric **ratio** is defined as

\[
\text{ratio} = \frac{\text{cut}(P)}{\text{size}(p_{i}) \times \text{size}(p_{j})}
\]

- Cut(P): sum of the weights of the edges that cross \( p_{1} \) and \( p_{2} \).
- Size(\( p_{i} \)): size of \( p_{i} \).
- The ratio metric balances the competing goals of grouping objects to reduce the cutsize without grouping distance objects.
- Based on this new metric, the partition algorithms try to group objects to reduce the cutsizes without grouping objects that are not close.
Simulated annealing

- Iterative algorithm modeled after physical annealing process that to avoid local minimum problem.
- 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}})
```
Genetic evolution

- Genetic algorithms treat a set of partitions as a generation, and create a new generation from a current one by imitating three evolution methods found in nature.
- Three evolution methods
  - Selection: random selected partition.
  - Crossover: randomly selected from two strong partitions.
  - Mutation: randomly selected partition after some randomly modification.
- Produce good result but suffer from long run times.

Genetic evolution’s algorithm

/*Create first generation with gen_size random partitions*/
G=∅
for i in 1 to gen_size loop
  G=GUCreateRandomPart(O)
end loop
P_best=BestPart(G)

/*Evolve generation*/
While not Terminate loop
  G=Select*G,num_sel) U Cross(G,num_cross)
  Mutate(G,num_mutatae)
  If Objfct(BestPart(G))<Objfct(P_best)then
    P_best=BestPart(G)
  end if
end loop

/*Return best partition in final generation*/
return P_best
**Integer Linear Programming**

- A linear program formulation consists of a set of variables, a set of linear inequalities, and a single linear function of the variables that serves as an objective function.
  - A integer linear program is a linear program in which the variables can only hold integers.
- For partition purpose, the variables are used to represent partitioning decision or metric estimations.
- Still a NP-hard problem that requires some heuristics.

**Partition example**

- The Yorktown Silicon compiler uses a hierarchical clustering algorithm with the following closeness as the terminal conditions:

\[
\text{Closeness}(p_i, p_j) = \left( \frac{\text{Conn}_{i,j}}{\text{MaxConn}(P)} \right)^{k_2} \cdot \left( \frac{\text{size}_{\text{max}}}{\text{Min}(\text{size}_i, \text{size}_j)} \right)^{k_3} \cdot \left( \frac{\text{size}_{\text{max}}}{\text{size}_i + \text{size}_j} \right)
\]

- \(\text{Conn}_{i,j}\): \(\text{k1} \cdot \text{inputs}_{i,j} + \text{wire}_{i,j}\)
- \(\text{inputs}_{i,j}\): # of common inputs shared
- \(\text{wires}_{i,j}\): # of op to ip and ip to op
- \(\text{MaxConn}(P)\): maximum Conn over all pairs
- \(\text{size}_i\): estimated size of group \(p_i\)
- \(\text{size}_{\text{max}}\): maximum group size allowed
- \(k1, k2, k3\): constants
**Ysc partitioning example**

YSC partitioning example:

(a) input
(b) operation
(c) operation closeness values
(d) Clusters formed with 0.5 threshold

Closeness(\(+, =\)) = \(\frac{8+0}{8} \times \frac{300}{120} \times \frac{300}{120 + 140} = 2.9\)

Closeness(\(-, <\)) = \(\frac{0+4}{8} \times \frac{300}{160} \times \frac{300}{160 + 180} = 0.8\)

All other operation pairs have a closeness value of 0. The closeness values between all operations are shown in figure 6.6(c).

Figure 6.6(d) shows the results of hierarchical clustering with a closeness threshold of 0.5; the + and = operations form one cluster, and the < and – operations from a second cluster.

---

**Ysc partitioning with similarities**

Ysc partitioning with similarities:

(a) clusters formed with 3.0 closeness threshold
(b) operation similarity table
(c) closeness values
With similarities (d) clusters formed.
Output Issues in Partitioning

- Any partitioning technique must define the representation format and potential use of its output
  - E.g., the format may be a list indicating which functional object is mapped to which system component
  - E.g., the output may be a revised specification
    - Containing structural objects for the system components
    - Defining a component’s functionality using the functional objects mapped to it

Flow of Control and Designer Interaction

- Sequence in making decisions is variable, and any partitioning technique must specify the appropriate sequences
  - E.g., selection of granularity, closeness metrics, closeness functions
- Two classes of interaction
  - Directives
    - Include possible actions the designer can perform manually, such as allocation, overriding estimations, etc.
  - Feedback
    - Describe the current design information available to the designer (e.g., graphs of wires between objects, histograms, etc.)
Comparing Partitions Using Cost Functions

- A cost function is a function Cost(H, S, Cons, I) which returns a natural number that summarizes the overall quality of a given partition
  - I contains any additional information that is not contained in H or S or Cons
  - A smaller cost function value is desired
- An iterative improvement partitioning algorithm is defined as a procedure
  Part_Alg(H, S, Cons, I, Cost( ))
  which returns a partition H', S' such that
  Cost(H', S', Cons, I) ≤ Cost(H, S, Cons, I)