

Partitioning

Introduction to Partitioning

Rabi Mahapatra

System partitioning

System level partitioning problem



Assignment of operations to
hardware or software

- Assignment of an operation to HW or SW determines the *delay of the operation*
- Assignment of operation to a processor and to more application-specific HW circuits involve additional *delays due to communication* overhead.

Good partitioning scheme \Rightarrow Minimize this communication

System partitioning contd..

- Increasing operations in software on a single processor
⇒ increases processor utilization
- **system performance:** depends on hw-sw partition on utilization of processor and bandwidth of bus between processor and application specific hardware.
- **Characteristic of Partitioning scheme:** capture and make use of partition's effect on system performance in making trade-off between hw and sw implementation of an operation.
 - Devise a “*partition cost function*”.

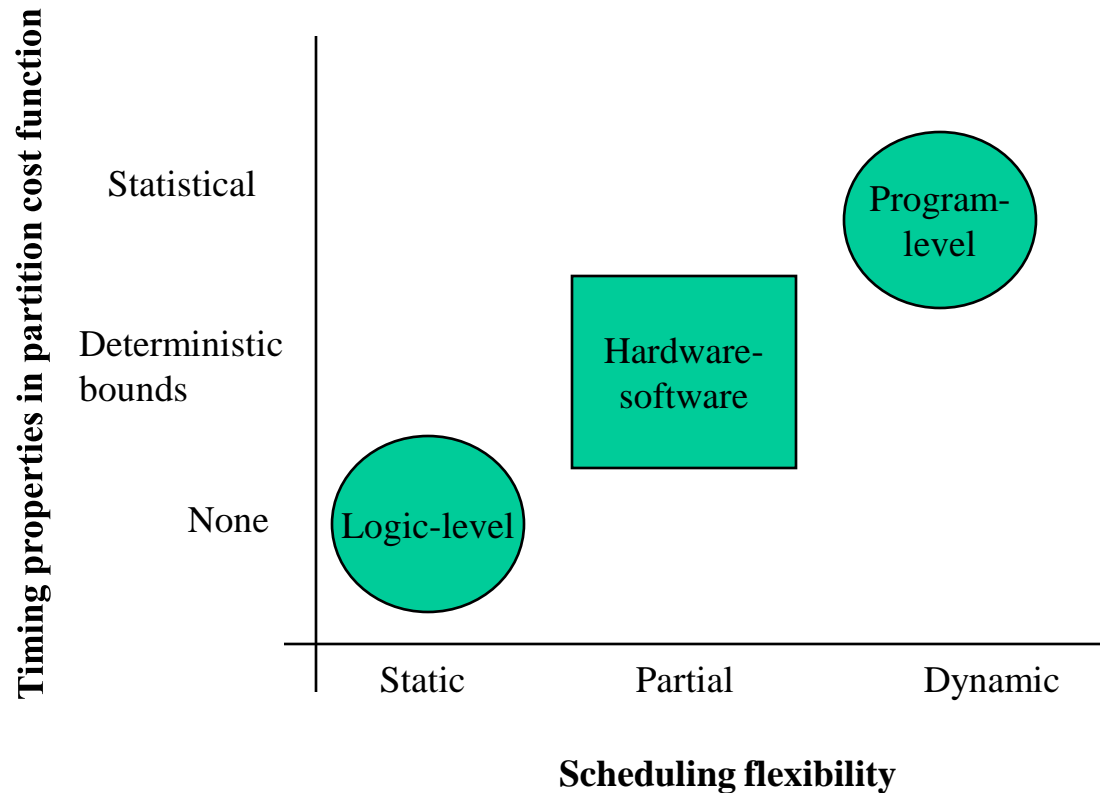
Partitioning

- **Cost function:**
 - Directs the partitioning algorithm towards desired solution
 - optimum solution is minimum cost function
- **Need to capture:**
 - effects of size of hw/sw parts
 - effects on timing behavior of these portions on cost function (contrast: optimized area/pinout)
 - difficult due to the problem being global in nature
 - approximation is used to account the effect on total latency

Partitioning

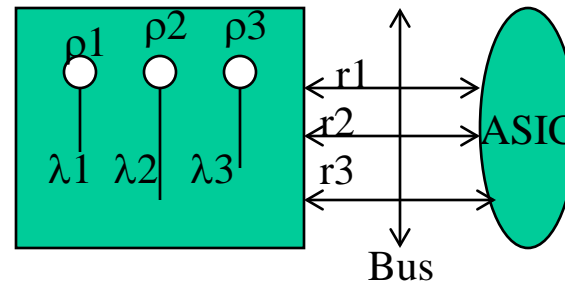
- Partitioning in **software**: extensive use of statistical timing properties to drive partitioning algorithm.
 - *Dynamic or runtime, excess time., flexible*
- Partitioning in **hardware**: attempts to divide circuits that implement schedule operations.
 - *Static, less time, non-flexible*
- An intermediate approach is advised: incrementally computable of cost function f .
 - *partial, deterministic bound on timing properties,*

Timing properties in partition cost function



A Partitioning cost function

- Consider software model in terms of set of program threads and cost function f .



- where, λ_i (per second) is thread latency: execution delay
- ρ_i (per second) thread reaction rate: invocation rate of the program thread

- processor utilization P is calculated by $P = \sum_{i=1}^n \lambda_i \rho_i$
- Bus utilization B (per second) = $\sum_{j=1}^m r_j$ “ m variables to be transferred, r_j = inverse of minimum time interval between consecutive samples for variable r ”.

Partition cost function

- Software characterization using λ , ρ , P and B: *static bound*
 - can be used to select appropriate partition of system functionality between hardware and software.
- over estimation of processor and bus bandwidth is possible (since actual distribution of data communication is not captured above)
- **Include S_H (hardware size)** bottom up.
 - From the size estimates of the resources implementing operations
- **Characterize interface using set of communication ports** (one per variable)
 - overhead due to communication between hw and sw is manifested by the utilization of bus bandwidth.

Partitioning with cost function

- From a given set of sequencing graph models and timing constraints, create two sets of sequencing graph models such that one can be implemented in hw and the other in sw and the following is true:
 - timing constraints are satisfied for the two sets of graph models
 - processor utilization, $P \leq 1$
 - bus utilization, $B \leq B'$
 - A partition cost function, $f = f(S_H, B, P^{-1}, m)$ is minimized.

Partitioning using heuristics

- Minimum cost function can be achieved by trying very large number of solutions (exponential relation to number of operations)
- ⇒ *heuristics are used for good solution that may show minimum cost function for some local properties*
- Start with constructive *initial solution* on which iterative procedure can be applied to improve the solution
 - exchange operations or paths between partitions, apply procedure
- A good heuristic is relatively insensitive to initial solution
 - exchange of large number of operation makes it more insensitive to starting solution

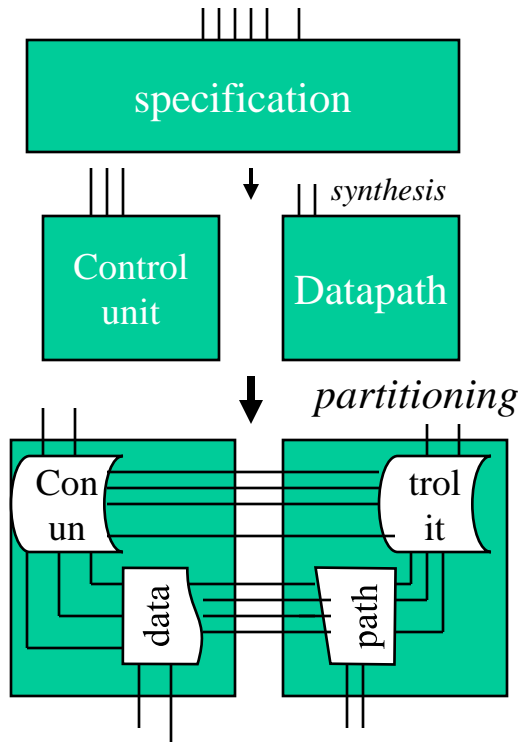
Partitioning trend

Many applications consist of one or small number of very large processes

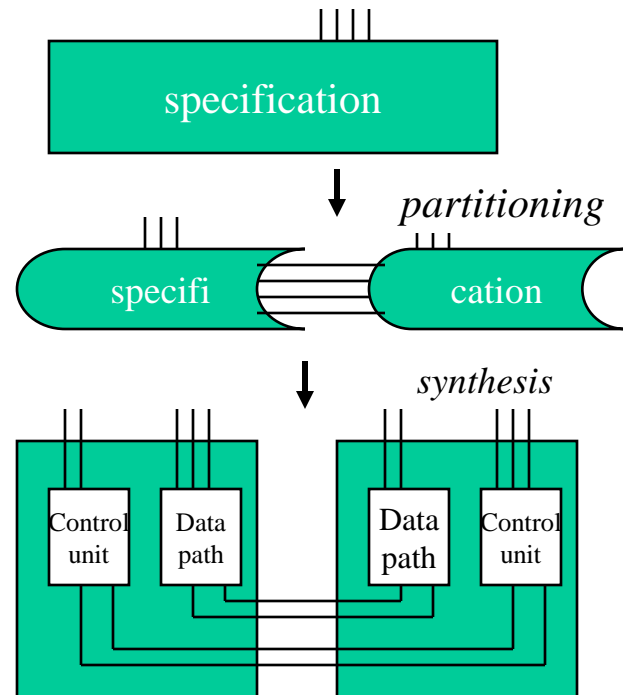
- Partitioning before synthesis or compilation has advantages
 - order of magnitude reduction in logic synthesis runtime.
 - Improved system performance as smaller processes can be synthesized with shorter clock period than one large processor.
 - Improved satisfaction of I/O and size constraints on a package, reducing inter-package signals (compared to structural partitioning)

Partitioning approaches

- **Structural**



- **Functional**



Functional Partitioning

- Divides a system's functional specification into multiple sub-specification.
- Each sub-specification represents the functionality of a system component, such as a custom-hardware or software processor.
- Then the components are synthesized down to gates or compiled to machine codes.

Advantages of FP

- Power reduction due to mutual exclusive components
- smaller board size, lower cost
- increase software speed
- concurrent synthesis and debugging
- less physical design problems

Problem description: Model

- Input: process x (C program or VHDL process)
- A view of the process: set of procedures $F = \{f_1, f_2, \dots, f_n\}$ with one as main procedure.
- Variable: simple processor with read and write being the procedure calls.
- Execution of F : procedures executing sequentially, starting with main and that calls other procedures; only one is active at a time

Problem description: Model

- Functional partitioning creates a partition P consisting of a set of parts $\{p_1, p_2, \dots, p_m\}$, such that every procedure f_i is assigned to exactly one part p_j , i.e. $p_1 \cup p_2 \cup \dots \cup p_m = F$ and $p_i \cap p_j = \emptyset$ for all $i, j, i \neq j$.
- Each p_j represents the function to be implemented on a single processor. The processors are mutually exclusive.
- Each part p_j is converted to a single process before synthesis; this process consists of a loop that *detects a request for one of the part's procedures, receive input parameters, calls the procedure, and sends back output parameters.*

Model contd...

- Function Bus: single bus carries parameter passing between processors
- Protocol: putting destination procedure's address, pulsing address request, putting parameter, pulsing the data request.
- Process $\xrightarrow{\text{Synthesis}}$ custom processor component C_i
- For application we target, C_i = non-trivial datapath and a complex controller with hundreds of states.
- Procedure on C_i may be implemented either as a control subroutine or datapath component.
- Synthesis may implement process's procedures in parallel if data dependencies are not violated.
 - *While procedures are not mutually exclusive after partitioning, processors are still mutually exclusive.*

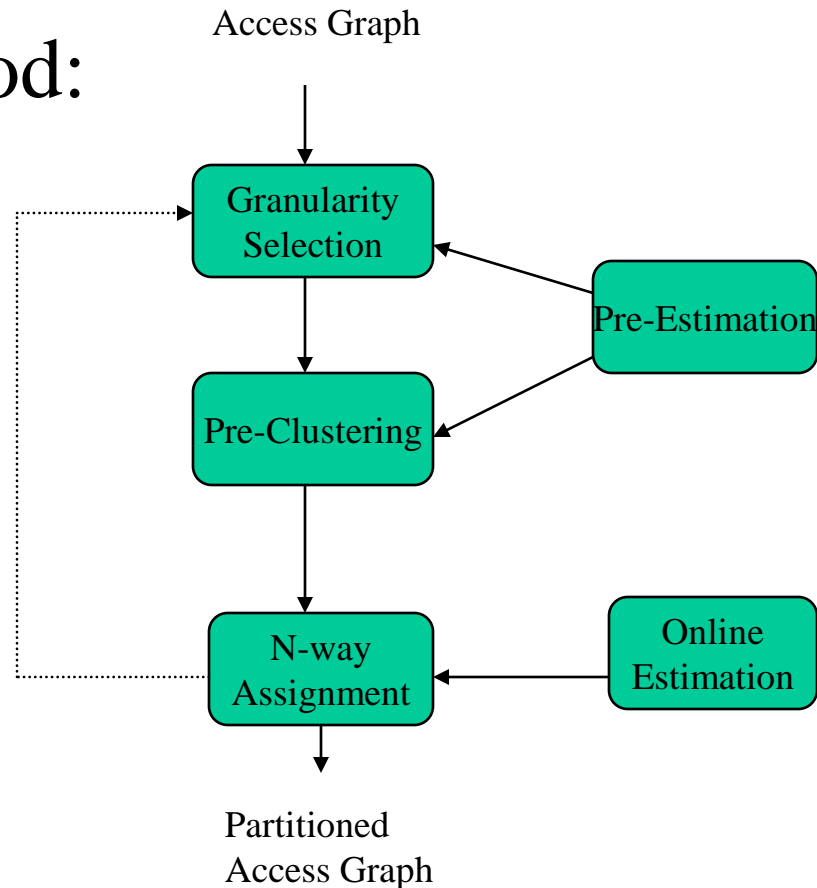
Five tasks for good partitioning

- Model creation
 - converts input to an internal model (call graph model)
- Allocation
 - Instantiating processors of varying type
- Partitioning
 - *Dividing input process among allocated processors*
- Transformation
 - modifies the input process into one with different organization but same overall functionality, leading to better partition.
- Estimation
 - provides data used to create values for design metrics. *Pre-estimation and online-estimation.*

Partitioning Methodology

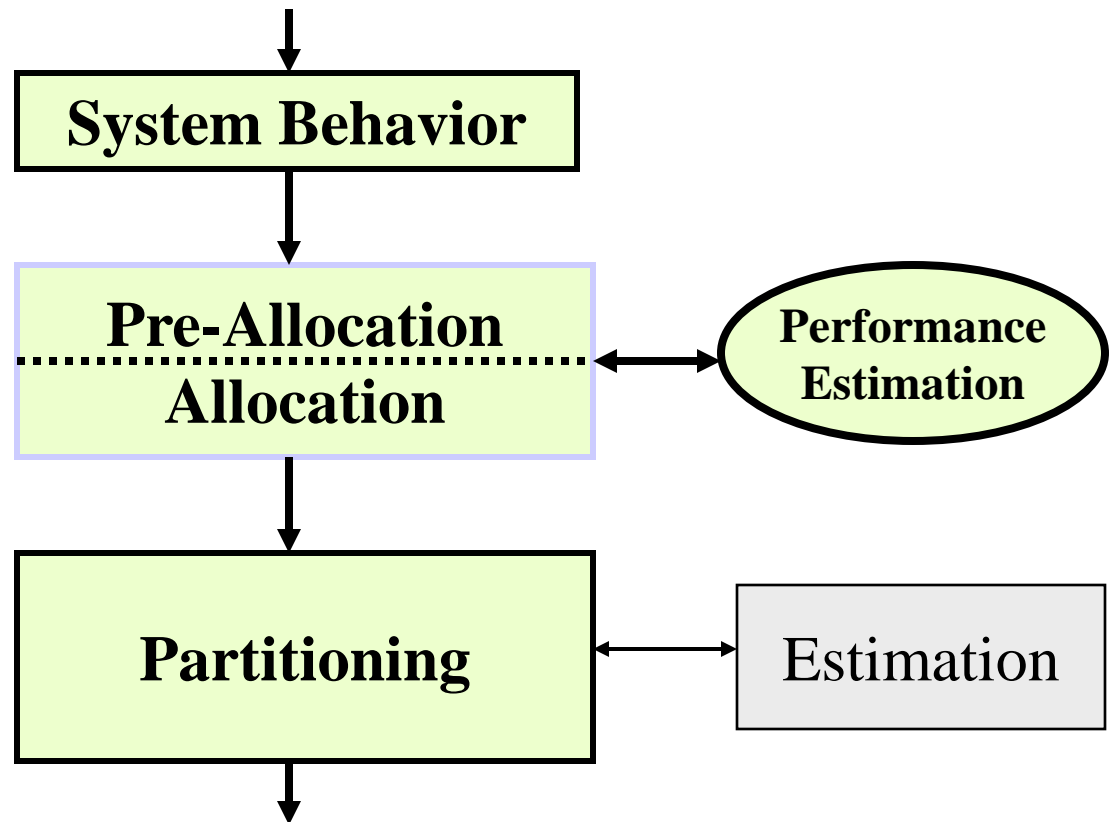
- Three-step method:

Sequence of partitioning steps proposed by Vahid



Design Space Exploration

Mohanty, Mahapatra & Choi. Proposed Pre-allocation...



Pre-Allocation

- Pre-Allocation:
 - Exploration of various design configuration before allocation
 - Embedded systems now have heterogeneous multiprocessors with ASICs, ASIPs, DSPs, Processors, Memories....etc.
 - Decide the number and type of components to be used
- Main Goal:
 - Reduced “Design Space” \Rightarrow reduced “Design Time”
 - Performance estimation for allocation

Step1: Granularity Selection

- **Goal:** Extract procedure from specification, which are to be assigned to processors during N-way assignment.
- Granularity is a measure of complexity
 - Fine: many procedures of low complexity.
 - **Little pre-estimation and online-estimation less accurate. Make online-estimation more complex to build higher accuracy.**
 - **Can be more time consuming and may prohibit the use of assignment heuristics that need many estimations.**
 - Coarse: few procedures of high complexity.
 - **many behaviors are grouped together into inseparable unit, so that any possible solution that separate those behavior is excluded.**

Granularity

- Procedures are selected very carefully to balance the above effects.
- Each statement is treated as *atomic* unit.
- Granularity Selection Problem:

Partitioning statements into procedures such that, (1) procedures are as course-grained as possible, to enable maximum pre-estimation and application of powerful N-way heuristics and (2) statements are grouped into a procedure only if their separation would yield inferior solution.

Granularity

- A straight forward heuristic: *choose a specification construct to represent a procedure. I.e. each statement or block. Also, user defined procedure for partitioning.*
- Transformations can be used to improve the above strategy
 - *Procedure Inlining*: replace procedure call by procedure's contents making granularity coarser. Inline procedure disappears.
 - *Procedure cloning*: makes a copy of a procedure for exclusive use by a particular caller. Ex: Multiply-called procedure if inlined might grow excess, and if not-inlined, might needs more communication. Cloning is a compromise.

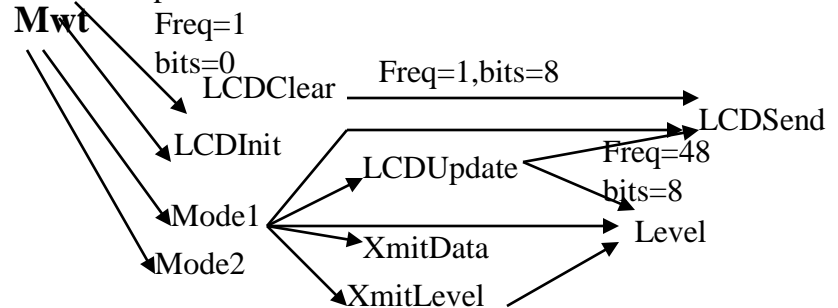
Illustration

Input specification
with many procedures

```
Mwt
bytelevel LcdSend(byte)
Mode1() LcdClear()
Mode2() LcdUpdate(byte,byte)
LcdInit() XmitLevel(byte)
          XmitData(bit)
```

```
begin
--sequence throgh modes
--which then call
--other procedures
```

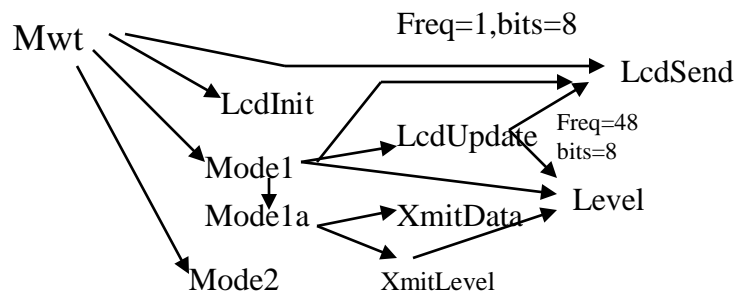
Access graph



Transformation contd..

- *Procedure Exlining*: Replaces a subsequences of a procedure's statements by a call to a new procedure containg only that subsequences. (opposite to inlining). This technique moves towards finer granularity.
 - *Redundancy exlining*: replaces two or more near-identical sequences of statements by one procedure. (use string matching method: statements are encoded characters)
 - *Distinct computation exlining*: Divide a large sequence of statements into several smaller procedures such that statements within a procedure are tightly related and would not be separated during N-way assignment solution.

Illustration of exlining



Step2: Pre-clustering

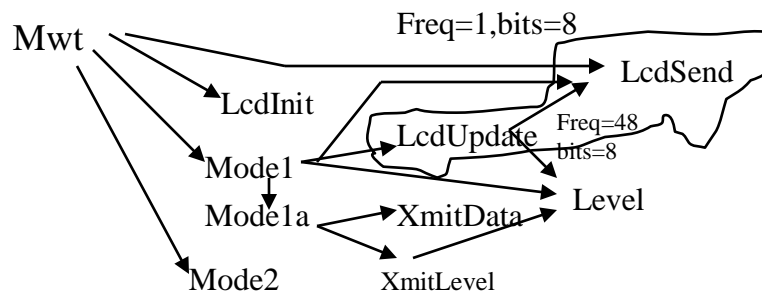
- Goal: Reduce the number of procedures for subsequent N-way assignment by merging procedures whose separation among parts would never represent good solution.
- Different from granularity step: procedures being clustered here may not be such that they could be combined into single new procedure. I.e. calls to these procedures are non-adjacent.
- Different from N-way assignment: each cluster does not represent a processor and therefore can not be guided by direct design metrics estimates.

Pre-clustering method

- Uses hierarchical clustering:
- procedures after granularity selection are converted to a graph node and edges are created between every pair weighed by the closeness of the nodes,
- closest pair of nodes are merged to a new node. This is repeated until no nodes are exceeding the threshold weight.[10]

Illustration of pre-clustering

- Two procedures *LcdUpdate* and *LcdSend* communicate heavily: 48 times per call.
- These two should never be separated. Since *LcdSend* appears 48 times inside *LcdUpdate*, inlining during granularity selection was not reasonable option.



More on pre-clustering

- Can reduce runtime of N-way assignment by 30% or more
- May look at Ethernet example in the reference.

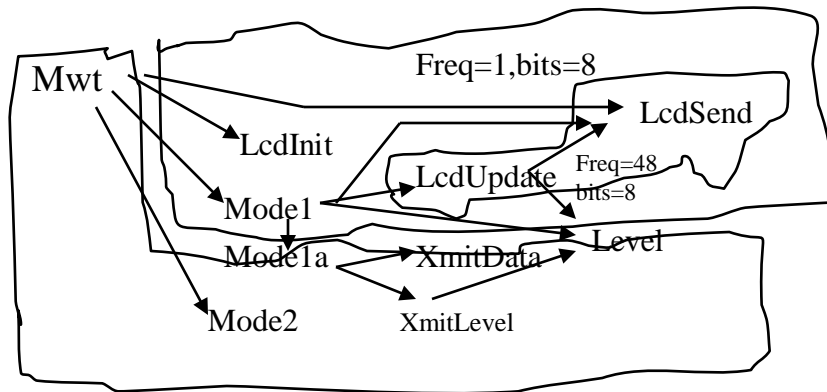
Step3: N-way assignment

- Goal: Distribute the procedure among given set of processors. Procedures are created after granularity selection and pre-clustering
- constructive heuristics are used to create initial solution and can include random distribution and clustering.
- There is an additional metric: “Balanced size” . Size of an implementation of both sets of node divided by the size of all nodes. This favors merging small sets over large ones.
- Heuristics applied: Greedy, Simulated Annealing, Hill climbing

N-way assignments

- Greedy algorithm: linear time heuristic that moves nodes that reduce the value of cost function
- Simulated annealing: randomized hill climbing to avoid local minima with long runtime
- Extended hill climbing: with some restrictions and tightly coupled data structure, $O(n \log(n))$ runtime
- *cloning* transformation can be applied selectively here
- *port-calling*, another transform: for I/O balance and ease access to shared ports. (I/O procedures are used in place of external port access that take care of send/receive etc.)

Illustration of N-way assignments



Other partitions of operations

- Aparty: among datapath modules using multi-stage clustering,
- Vulcan: among packages using iterative improvement heuristics
- Chop: among packages focusing on providing suite of feasible solutions for each package that would satisfy overall constraints
- Multipar: among packages simultaneous with scheduling and allocation, using linear programming
- SpecPart: partitioned procedures among packages using clustering and iterative improvements.

Limitation of three-step approach.

- Total hardware increase may be large for examples with small controllers and large datapaths.
- Problems that has large number of small processes - much like a scheduling problem
- parallel execution on processors
- References:
 - *Rajesh Gupta et.al. "Hardware-Software Cosynthesis for Digital Systems", IEEE Design & Test of computers, Sept 1993.*
 - *Reference: Frank Vahid, "A three-step approach to the functional partitioning of large behavioral processes".*