To appear: *Proceedings of Sixth International Workshop on Rapid Systems Prototyping, Chapel Hill, NC, June, 1995.*

The Extended Partitioning Problem: Hardware/Software Mapping and Implementation-Bin Selection

Asawaree Kalavade and Edward A. Lee

Dept. of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720, USA
{kalavade,eal}@eecs.berkeley.edu

Abstract

The extended partitioning problem is the joint problem of mapping nodes in a precedence graph to hardware or software, and within each mapping, selecting an appropriate implementation for each node. The end-goal is to minimize the hardware area, subject to architectural and performance constraints. This is an NP-complete problem; we present an efficient heuristic called MIBS to solve it. The MIBS algorithm solves the extended partitioning problem by decomposing it into an iterative process consisting of two steps: mapping and implementation-bin selection. The GCLP algorithm computes a mapping by using an adaptive optimization objective at each iteration. This objective is selected on the basis of a global time criticality measure and local optimality measures. The IBS algorithm solves the implementation-bin selection problem. It uses a bin sensitivity measure, which correlates the implementation-bin motion with the overall hardware area reduction, to determine the implementation bin of a node for a given mapping. Experimental results indicate that the added dimension of design flexibility (offered by implementation bins) can be used effectively in partitioning to reduce the overall area. The MIBS algorithm has $O(|N|^3)$ *complexity, with a solution quality comparable to that of ILP.*

1.0: Introduction

Traditional partitioning approaches make a binary choice between hardware and software (mapping) for each node. However, a node can have different implementation alternatives (called implementation bins) within a given mapping. These implementations differ in area and time characteristics. The extended partitioning problem selects the appropriate implementation for a node, in addition to deciding whether it is in hardware or software.

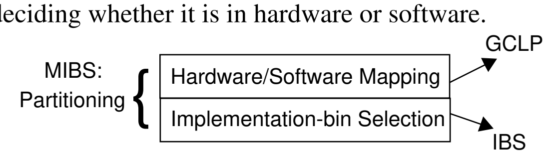

Figure 1: The Extended Partitioning Problem

Partitioning problems in general are difficult. Parameters of the design can be approximated to formulate a linear optimization problem that can be solved exactly. Exact solutions (typically using ILP) are intractable for even moderately small problems. This paper proposes and evaluates a heuristic solution for the extended partitioning problem.

Our approach to solving this problem is to decompose it into two sub-problems as shown in Figure 1. The GCLP algorithm, first described in [1], is used to select the mapping, and the IBS algorithm embedded under it selects the implementation bin.

The outline of this paper is as follows: The extended partitioning problem is motivated in Section 2.0. The GCLP algorithm is summarized in Section 3.0. The IBS algorithm is proposed in Section 4.0. The MIBS algorithm for the extended partitioning problem is described in Section 5.0, and experimental results are presented in Section 6.0.

2.0: Motivation and problem definition

The **hardware/software mapping problem (*P1*)**: Given a DAG (where nodes represent computations and arcs represent data and control precedences between nodes), area and time estimates for software and hardware implementations of all nodes, communication costs, and a desired latency, determine a mapping (M) of nodes to hardware and software, and the start time for each node (schedule t), such that the area occupied by the nodes mapped to hardware is minimum.

The mapping problem is combinatorial in the order of nodes ($O(2^{|N|})$ by enumeration). Heuristics have been reported to solve this more efficiently [1][2][3]. In addition to determining the mapping, there is yet another dimension of design flexibility — a node in a given mapping can have many implementation alternatives. Figure 2 shows the pareto-optimal points (implementation bins) in the area/time trade-off curves for the hardware mapping of typical nodes. Each of these curves was obtained by running a behavioral description (in Silage) of the node through the high-level synthesis system, Hyper [4]. The sample period

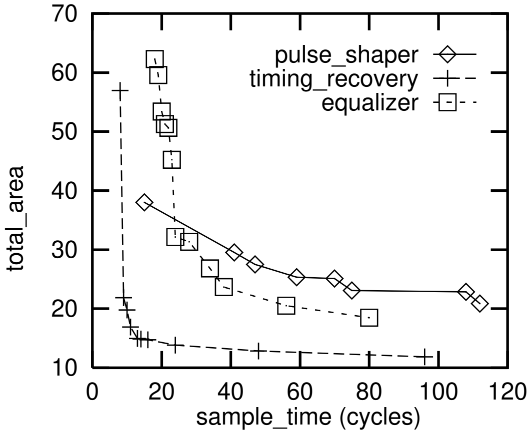

Figure 2: Typical area/time trade-off curves.

Notation	Interpretation
G	DAG G = (N,A), N: nodes, A: arcs
D	Latency constraint (deadline)
AH(AS)	Total hardware (software) capacity constraint
ah_i (as_i)	Default area estimate for node i in hardware (software)
th_i(ts_i)	Default execution time estimate for node i in hardware (software)
CH_i(CS_i)	Hardware (software) implementation curve

Table 1: Summary of notation

is shown on the X axis, and the corresponding hardware area required to implement the node is shown on the Y axis. The left-most point on the X axis for each curve corresponds to the critical path of that node; it represents the fastest possible implementation of the node. As the sample period increases, the same node can be implemented in a smaller hardware area. The right-most point on the X axis for each curve corresponds to the smallest possible area; it represents the point where only one resource of each type is used, the design cannot get any smaller. Thus, the curve represents the design space for a node.

Similarly, different software synthesis strategies can be used to implement a given node in software, giving rise to similar area/time curves for software implementations. For instance, inlined code is faster than code using subroutine calls, but with a larger code size.

In general, associated with every node i are: a hardware implementation curve CH_i, and a software implementation curve CS_i. $CH_i = \{(ah_i^j, th_i^j), j \in NH_i\}$, where ah_i^j and th_i^j represent the area and execution time when node i is implemented in hardware using implementation-bin j, and NH_i is the set of hardware implementation bins for node i (Figure 3). CS_i can be similarly defined. The notation is summarized in Table 1.

The **implementation-bin selection problem (*P2*):** Given a node i with a pre-specified hardware or software mapping, determine the implementation bin (B_i*) for the node.

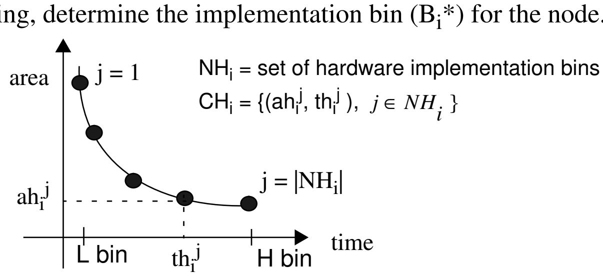

Figure 3: Hardware implementation curve (CH_i) for node i

The extended partitioning problem seeks to jointly optimize the selection of the implementation bin and the mapping.

The **extended partitioning problem (*P3*)** is stated as: Given a DAG, hardware and software implementation curves for all the nodes, communication costs, and a desired latency, find a mapping (*M*), the optimal implementation bin (B*), and the start time for each node (schedule *t*), subject to the latency and architectural constraints and taking communication costs into consideration, such that the total area of the nodes mapped to hardware is minimum.

The motivation in solving *P3* is a possibly reduced overall hardware area (relative to *P1*), while meeting the timing constraints. It is obvious that *P3* is a much harder problem than *P1* ($k^{|N|}$ alternatives, for k implementation bins per node, for a given mapping and $|N|$ nodes). In this paper we present an efficient algorithm to solve *P3* approximately. We also investigate, with case studies, the pay-off in using extended partitioning over just mapping (binary partitioning).

Solution Methodology

We propose an algorithm that solves *P3* by decomposing it into an iterative process, consisting of two steps, as shown in Figure 4. A node is defined to be *fixed* if its hardware/software mapping as well as implementation bin have been determined. A node is defined to be *free* if both its mapping and implementation bin are yet to be determined. A node is defined to be *tagged* if it is taken up for implementation-bin selection after having been mapped to hardware or software. In Figure 4, the GCLP algorithm is first

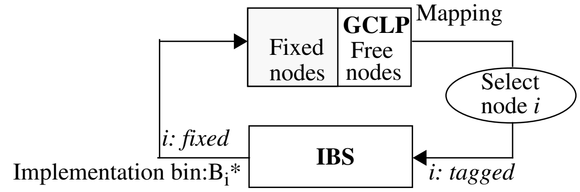

Figure 4: Proposed algorithm for *P3*

applied to get an initial hardware/software mapping for free nodes. A tagged node is then selected from the set of mapped nodes. The IBS algorithm determines the appropriate implementation bin for this tagged node. The tagged node becomes a fixed node once its implementation bin is determined. The GCLP is then applied to compute the revised mapping for the remaining free nodes. This GCLP-IBS sequence is repeated till there are no free nodes left.

3.0: Hardware/software mapping: GCLP algorithm revisited

In this section we briefly summarize the key ideas of the Global Criticality/Local Phase (GCLP) driven algorithm for the mapping problem. Refer to [1] for further details.

3.1: Adaptive selection of the objective function

The GCLP algorithm traverses the DAG and maps each node to either hardware or software, such that an objective function is minimized. Two possible objective functions could be used: minimize the *finish time* of the node (governed by the execution time on the selected mapping and the communication between the node and its predecessors) or minimize the *percentage resource consumed* by the node (hardware area or software size). However, meeting timing constraints and minimizing hardware area are contradictory goals. For example, an objective function that minimizes the finish time is more likely to be feasible, but selecting the faster mapping could drive it away from optimality (minimum hardware area). On the other hand, if a node is assigned to a mapping that minimizes hardware area, it is likely to be infeasible. To overcome this problem, the GCLP algorithm *selects an appropriate optimization objective at each step,* instead of using a *hardwired* objective function. As shown in Figure 5, the objective function is selected by a threshold-based comparison of a global time-criticality measure, called *global criticality*. If time is critical, an objective function that minimizes finish time is selected, otherwise one that minimizes area is selected.

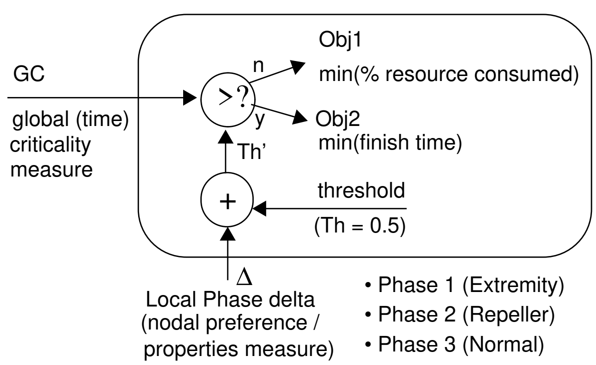

Figure 5: GCLP: Objective function selection per step

3.2: Global Criticality (GC)

GC is a global lookahead measure of time criticality at each step of the algorithm. Figure 6 illustrates the computation of GC. At each step, GC is the fraction of unassigned nodes that have to be moved to hardware from software so as to meet feasibility.

3.3: Local Phase

Mapping based on just GC is unlikely to be globally optimal because the node-invariant nature of GC at each step does not capture the local characteristics of individual nodes. To address this, nodes are classified into three types: local phase 1(extremity), local phase 2(repeller), and local phase 3 (normal).

Extremities are resource hogs and have an obvious preferred mapping. For instance, a hardware extremity requires a large area when implemented in hardware, but could be implemented inexpensively in software. The local preference of such a node i is quantified by an *extremity measure* E_i.

On the other hand, some nodes are better suited to hardware or software based on their structure [5]. Such nodes are classified as **repellers**. For instance, a node with a large number of bit manipulations is better suited for hardware (software repeller), while a node with many memory accesses is better matched to software (hardware repeller). For such nodes, this behavior is quantified by a *repeller measure* R_i.

Nodes that are neither extremities nor repellers are classified as **normal** nodes.

3.4: GCLP Algorithm

The GCLP algorithm is outlined below. In each iteration, the global criticality is first computed. Next, a node is selected for mapping from among ready nodes (nodes whose predecessors have been scheduled) based on an *urgency* criteria (Steps 2-5). Having selected a ready node, the mapping objective is determined by comparing GC to a threshold. The threshold is modified by an amount Δ, where $\Delta = E_i$ if *i* is an extremity node, $\Delta = R_i$ if *i* is a repeller node, and $\Delta = 0$ if *i* is a normal node. If GC is

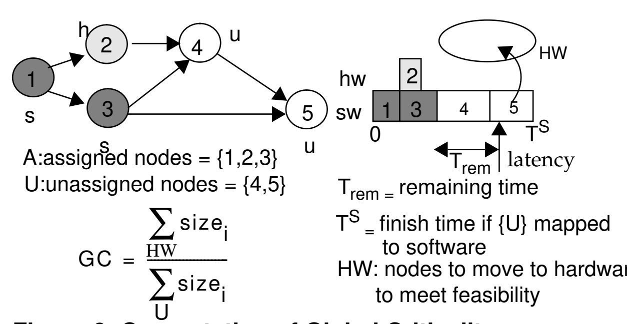

Figure 6: Computation of Global Criticality

greater than the threshold, time is critical; an objective that minimizes time is selected, otherwise one that minimizes area is chosen. Based on the selected objective, the ready node *i* is assigned a mapping (M_i) and is scheduled in a time slot (starting time t_i). The algorithm has quadratic complexity in the number of nodes.

Algorithm: **GCLP**

Input: $G = (N, A)$; AH, AS, and D.
$\forall i \in N$: ah_i, as_i, th_i, ts_i, E_i (extremity measure), and R_i, (repeller measure).

Output: $\forall i \in N$, mapping M_i and start time t_i.

Initialize: U = {unassigned nodes} = N, A= {assigned nodes} = ϕ .

Procedure:

while ($|U| > 0$){

1.Compute GC

2.Determine the set of *ready* nodes R

3.Compute the effective execution time $t_{exec}(i)$ for each node i
  If $i \in U$ $t_{exec}(i) = GC.th_i + (1-GC).ts_i$
  else if $i \in A$ $t_{exec}(i) = th_i \bullet I(M_i = hw) + ts_i \bullet I(M_i = sw)$

4.Compute the longest path $\forall i \in R$ using $t_{exec}(i)$

5.Select node i, $i \in R$, for mapping: max(*longestPath(i)*)

6.Determine mapping M_i for i:

 6.1.if ($E_i \neq 0$) $\Delta = E_i$ (extremity node)
  else if ($R_i \neq 0$) $\Delta = R_i$ (repeller node)
  else $\Delta = 0$; (normal node)

 6.2. $\text{Threshold} = 0.5 + \Delta$, $0 \le \text{Threshold} \le 1$

 6.3.If ($GC \ge$ Threshold) p: min(finish time);
  else p: min(resource consumed);

 6.4.M_i = p; Set(t_i); U = U\{i}; $A \leftarrow \{i\}$
 }

4.0: Implementation-bin selection

4.1: Problem definition

Consider the nodal implementation bin curve as shown in Figure 2. Denote L to be the fastest (leftmost) implementation bin, and H to be the slowest (rightmost) implementation bin. As the nodal implementation bin curve is traversed from L to H, the hardware area required to implement the node decreases. From the view point of minimizing hardware area, it is desirable for all nodes mapped to hardware to be set at their H bins. This might, however, be infeasible from the point of view of desired latency. Similarly, all nodes mapped to software should likely be at their L bins, but program/data memory capacity constraints might restrict this. The implementation-bin selection problem (*P2*) is to determine the implementation bin for a tagged node. An *Implementation Bin Selection* (IBS) heuristic for this problem is presented in this section.

For the sake of simplicity, in the discussions that follow, we restrict the implementation-bin selection to hardware only. Note that the concepts apply to software implementation-bin selection as well.

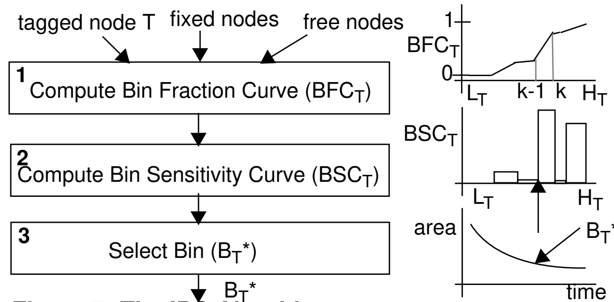

Figure 7: The IBS Algorithm

4.2: IBS algorithm overview

Figure 7 shows the key concepts of the algorithm. Given a tagged node (T) and a set of free and fixed nodes, the IBS algorithm selects an implementation bin for the tagged node (B_T*). The bin is selected so that the timing constraints are met, while assigning fewest possible free nodes to their L bins. This has the effect of reducing the overall hardware area, while maintaining feasibility.

For each bin *j* of a tagged node T, we estimate the fraction of free nodes that need to be mapped to their L bins, in order to meet timing constraints. This estimate is defined as the *bin fraction* (BF_T^j). A high value of BF_T^j indicates that if the tagged node T were to be implemented in bin *j*, a large fraction of free nodes may subsequently get mapped to their fast implementations, increasing the overall area. The *bin fraction curve* (BFC_T) is the collection of bin fraction values for each bin *j* of the tagged node T (Step 1 in Figure 7).

Next, the *bin sensitivity curve* (BSC_T) for the tagged node T is determined by computing the slope of the BFC_T (Step 2 in Figure 7). The bin sensitivity curve reflects the responsiveness of the bin fraction. Suppose that the maximum slope of the bin fraction curve is between bins k-1 and k (in Figure 7). This implies that moving the tagged node from bin k-1 to bin k shifts the largest fraction of free nodes to their L bins. The k to k-1 bin-motion for the tagged node thus results in the largest reduction of the area of free nodes.

Hence, bin (k-1) is selected as the implementation bin (B_T*) for the tagged node, in Step 3 in Figure 7. The key idea here is to correlate the motion of the tagged node along its implementation bin curve from its H to L bin, with the motion of free nodes from their L to H bins.

The notation used in the IBS algorithm is summarized in Table 2.

4.3: Bin Fraction Curve (BFC)

The bin fraction curve BFC_T is plotted by computing, for each bin *j* of the tagged node T, the bin fraction BF_T^j. Assume that each free node can be either in L or H bin.

BF_T^j is computed as the fraction of free nodes that have

Notation	Interpretation
T	tagged node
B_T*	Selected bin for node T
BF_T^j	Bin Fraction for bin j of node T
BFC_T	Bin Fraction Curve for node T
BSC_T	Bin Sensitivity Curve for node T
SL_{max}	Maximum value of BSC_T

Table 2: Summary of notation in IBS algorithm

to be mapped to L bins in order to meet feasibility, if node T were implemented in bin j. The algorithm to compute the BFC is as follows:

Algorithm: **compute_BFC**

Input: A = {fixed nodes}, U = {free nodes}, T = tagged node, with mapping M_T, CH_T, L_k and H_k for all nodes $k \in U$

Output: $BFC_T = \{(BF_T^j, j),\ j \in NH_T$

Initialize: $F = \phi$, for $p \in A$, set $t_{exec}(p)$ based on fixed bin

Procedure:

for (j = 1; $j \le |NH_T|$, j++) {

1. Set $t_{exec}(T) = th_T^j$
2. For all $k \in U$, set $t_{exec}(k) = th_k^H$
3. Compute T_{finish}, given the mapping and t_{exec} for all nodes
4. while (T_{finish} > latency){
   $F \leftarrow$ next(U)
   for all $f \in F$, $t_{exec}(f) = th_k^L$
   Update(T_{finish})}
5. $BF_T^j = \dfrac{\sum_{i \in F} size_i}{\sum_{i \in U} size_i}$

}

Figure 8 describes this mechanism to compute BFC with the help of an example. Suppose that nodes 1, 2, and 3 are fixed, node 4 is tagged, and node 5 is free. Further, suppose that a GCLP run at this point mapped nodes 4 and 5 to hardware.

The bin fraction curve for the tagged node (BFC_4) is plotted by determining the values of the bin fraction for all the bins of node 4. Consider the computation of BF_4^H. Node 4 is set to its H bin. All free nodes (node 5) are first set at their H bins. $T_{finish\text{-}4\text{-}H\text{-}5\text{-}H}$ is the finish time for the DAG for this particular bin selection. Suppose that this exceeds the timing constraint as shown in Figure 8(b). Some

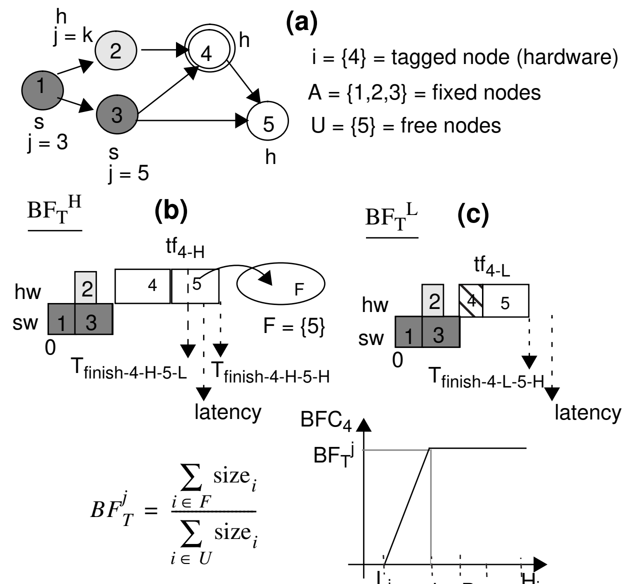

Figure 8: BFC Computation

of the free nodes hence have to be moved to their L bins to meet the timing constraint. A free node (5) is selected and moved to its L bin. The set F contains the free nodes that are at their L bins (F = {5}). The new finish time ($T_{finish\text{-}4\text{-}H\text{-}5\text{-}L}$) is computed. Suppose that this satisfies timing constraint. BF_4^H is then computed as the ratio of nodes in F to nodes in U. BF_4^H evaluates to 1 in this case. The procedure is repeated for all $|NH_4|$ bins for node 4. Consider the computation of BF_4^L. The free nodes (5) are again set at their H bin. $T_{finish\text{-}4\text{-}L\text{-}5\text{-}H}$ is the finish time for the DAG for this particular bin selection. Assuming this is feasible, BF_4^L evaluates to 0. BFC_4 is shown in Figure 8(d).

Note that the BFC is a monotonically non-decreasing function. A high value of the bin fraction for a bin j indicates that selecting the jth implementation bin for the tagged node is likely to result in a large fraction of free nodes being subsequently assigned to their L bins. The free nodes to be moved to L bins can be selected by using different ranking functions such as th_i^H or ah_i^L.

4.4: Bin Sensitivity Curve (BSC)

Figure 9 plots the BF values for various bins of a tagged node T. How is the optimal bin (B_T*) for this node selected? As ah_T^H is the smallest possible area for the tagged node in hardware, an intuitive choice for its bin is to set B_T* = H_T. At H_T, however, BF_H is high and the overall hardware area might not be optimal. As the tagged node shifts from bin H_T to H_{T-1}, the resulting decrease in BF implies that the fraction of free nodes at their H bin increases, and consequently the allocated hardware area of free nodes reduces. In general, the slope of the BFC represents the free-node area reduction gradient with respect to the (leftward) bin motion of the tagged node.

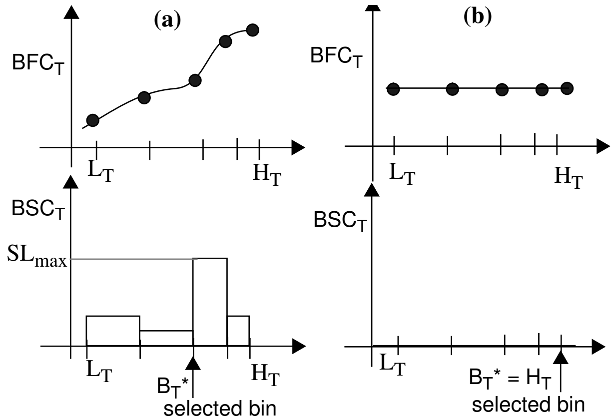

Figure 9: Using BFC and BSC to determine B_T*

The method to compute the optimal implementation bin is now described with the help of a typical BFC, in Figure 9(a). The slope of the BFC is first plotted. This is called the bin sensitivity curve, since it reflects the correlation between bin motion and free-node area reduction. Let the maximum slope of BFC be SL_{max}. The bin (B_T*) for the tagged node is selected to be the leftmost bin with the maximum slope SL_{max}, if $SL_{max} > 0$. If BFC is constant (Figure 9(b)), $SL_{max} = 0$, and the tagged node is mapped to its H bin, since moving it from its slowest to fastest implementations does not affect subsequent nodes.

Consider the BSC_T in Figure 10(a) where the two regions marked S1 and S2 have identical slopes. In this case, the bin B1, which is closer to the H_T bin is preferred over bin B2, as this reduces the area of the tagged node. To incorporate the effect of area in general the BSC_T is weighted by the area of the tagged node. Figure 10(c) indicates the normalized area curve for the tagged node. Figure 10(d) illustrates the *weighted BSC* obtained by weighting Figure 10(b) with Figure 10(c). B_T* is then selected to be the left most bin with the maximum weighted slope.

Algorithm **IBS**

Input: A = {fixed nodes}, U = {free nodes}, T = tagged node, with mapping M_T, CH_T, L_k and H_k for all $k \in U$

Output: B_T*

Procedure

1. Compute BFC_T (Section 4.3)
2. Compute BSC_T
3. Compute weighted BSC_T
4. Determine bin B_T* corresponding to leftmost bin with the maximum slope for the weighted BSC_T

5.0: The Extended Partitioning Problem: MIBS algorithm

The MIBS algorithm is described below. Note that the mapping of all the nodes is not finalized at one shot in MIBS. Instead, future mappings of the remaining free

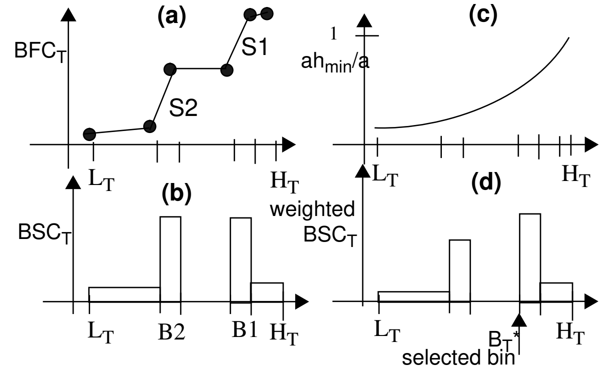

Figure 10: Weighted Bin Sensitivity Curve.

nodes are allowed to change depending on the implementation bin selected for a tagged node. At any iteration, the known mappings and implementation bins of the fixed nodes affect the mappings of the free nodes. Also note that the GCLP uses the median values. The MIBS algorithm has complexity $O(|N|^3)$.

Algorithm: **MIBS**

Input: $G = (N, A)$, AH, AS, and D. $\forall i \in N$: CH_i, CS_i, E_i (extremity measure), and R_i, (repeller measure).

Output: $\forall i \in N$, M_i, B_i*, and t_i.

Initialization: A = {fixed nodes} = ϕ, U = {free nodes} = N. Compute median values for area and time in software and hardware

Procedure

while {|U| > 0} {

1. Determine M_i for all $i \in U$
   - 1.1. For $i \in U$ set hardware and software area and time values to their median values.
   - 1.2. Run GCLP to compute M_i and t_i for $i \in U$.
2. Determine set of ready nodes R
3. Select tagged node T using urgency measures ($T \in R$)
4. Determine the implementation bin for node T, given M_T
   - 4.1. Run IBS to determine bin B_T* (Section 4.4)
5. $U = U \backslash \{T\}$; $A \leftarrow \{T\}$, Update t_i based on selected B_T*}

6.0: Results

The performance of the MIBS algorithm is next examined. Our focus is on real-time applications with periodic timing constraints. Two examples are selected: a 32 KHz, 2-PSK modem with 27 nodes and a 8KHz bidirectional telephone channel simulator with 15 nodes.

6.1: Estimation of area/time values

The applications are described at a *task* level of granularity (typical nodes include: pulse shaper, carrier recovery, timing recovery, etc.) in Ptolemy [6]. The underlying target architecture is assumed to consist of a programmable processor, custom hardware, and self-timed memory-mapped communication. Hardware and software for the nodes is synthesized using techniques reported in [7]. On an average, there are 7-8 hardware implementation bins per node. A single software implementation is assumed.

6.2: Comparison with ILP

The solution obtained by the MIBS algorithm is compared to an exact (integer linear program) solution. In the ILP formulation, a 15 node problem required 718 constraints and 396 variables (381 integer variables). The ILP was solved using CPLEX on a Sun SPARCstation10. Table 3 summarizes these results. The closeness of the solutions is encouraging, especially since ILP becomes formidable for even small problems.

Scenario	hardware area	run time
ILP	158	3.5 hours
MIBS	181	3 minutes
Comparison	1.1456 times bigger	70 times faster

Table 3: Comparison of ILP and MIBS.

6.3: Mapping vs. Extended Partitioning

In the following examples we show that the ability to select an implementation bin, rather than just use a single bin for all the nodes, significantly reduces the overall hardware area.

Three cases are considered. In the first case, the GCLP algorithm is applied to the graph, where the execution times and areas for all the nodes correspond to their L bins values. In the second case, the GCLP algorithm is applied to the graph where the area and execution time values correspond to the median implementation bins. In the third case, the MIBS algorithm is applied. Table 4 shows the results for the three cases for the modem example. Case 2 gave a better solution (28% smaller hardware area) than case 1 as expected. The MIBS solution is far superior to that obtained with just GCLP (50% less hardware compared to the fastest case, and 32% less than the median case). This proves that the implementation flexibility can be used in partitioning to reduce the overall hardware area.

	Scenario	HW area	area gain
1	GCLP, fastest implementation	736	1
2	GCLP, median implementation	530	0.7201
3	MIBS	362	0.4918

Table 4: Area improvement using the MIBS.

7.0: Conclusions

The extended partitioning problem seeks to jointly optimize the mapping of nodes to hardware or software, and the selection of implementation bins within a mapping. The MIBS heuristic is presented to solve this problem efficiently ($O(|N|^3)$). It solves the extended partitioning problem by decomposing it into an iterative process consisting of two steps: mapping and implementation-bin selection. The GCLP algorithm computes the mapping by using an adaptive optimization objective at each step. This objective is selected on the basis of a global time criticality measure and local optimality measures. The IBS algorithm solves the implementation-bin selection problem. It uses a bin sensitivity measure, which correlates the implementation-bin motion with the overall hardware area reduction, to select an implementation bin for a node for a given mapping. This GCLP-IBS sequence is repeated for |N| nodes in the DAG. Experimental results indicate that the added dimension of design flexibility (offered by implementation bins) can be used effectively in partitioning to reduce the overall area. The solution obtained by the MIBS algorithm is close to the ILP solution.

8.0: Acknowledgements

This research is part of the Ptolemy project, which is supported by ARPA and the U.S. Air Force (under the RASSP program F33615-93-C-1317), SRC (95-DC-324-016), NSF (MIP-9201605), Office of Naval Technology (via NRL), the State of California MICRO program, and the following companies: Bell Northern Research, Dolby, Hitachi, Mentor Graphics, Mitsubishi, NEC, Pacific Bell, Philips, Rockwell, Sony, and Synopsys.

9.0: References

[1] A. Kalavade, E. A. Lee, “A Global Criticality/ Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning Problem”, *Proc. of CODES/CASHE, Third Intl. Workshop on Hardware/Software Codesign*, Grenoble, France, Sept. 22-24, 1994, pp 42-48.

[2] R. Gupta, G. Micheli, “System-level Synthesis Using Re-programmable Components”, *Proc. of European Conf. on Design Automation*, Brussels, Belgium, Feb.1992, pp 2-7.

[3] R. Ernest, J. Henkel, “Hardware/software Codesign of Embedded Controllers based on Hardware Extraction”, *Handouts of the 1st Intl. Workshop on Hardware/Software Codesign*, Estes Park, Colorado, Sept. 1992.

[4] J. M. Rabaey et al. “Fast Prototyping of datapath-intensive Architectures”, *IEEE Design & Test,* June 1991, pp. 40-51.

[5] D. E. Thomas, J. K. Adams, H. Schmit, “A Model and Methodology for Hardware/Software Codesign”, *IEEE Design and Test of Computers*, Sept. 1993, pp 6-15.

[6] J. Buck, et al. “Ptolemy: a Framework for Simulating and Prototyping Heterogeneous Systems”, *Intl. Journal of Computer Simulation*, special issue “Simulation Software Development,” Apr. 1994, v4, pp 155-182.

[7] A. Kalavade, E. A. Lee, “A Hardware/Software Codesign Methodology for DSP applications”, *IEEE Design and Test of Computers*, Sept. 1993, pp 16-28.