Effective Graph Theoretic Techniques for the Generalized Low Power Binding Problem

Azadeh Davoodi
Azade@eng.umd.edu

Ankur Srivastava
Ankurs@eng.umd.edu

Department of Electrical and Computer Engineering
University of Maryland, College Park
MD, USA, 20742

ABSTRACT

This paper proposes two very fast graph theoretic heuristics for the low power binding problem given fixed number of resources and multiple architectures for the resources. First the generalized low power binding problem is formulated as an Integer Linear Programming(ILP) problem which happens to be an NP-complete task to solve. Then two polynomial-time heuristics are proposed that provide a speedup of up to 13.7 with an extremely low penalty for power when compared to the optimal ILP solution for our selected benchmarks .

Categories and Subject Descriptors

B.5.2 [**RTL-Implementation**]: Design Aids—*Automatic synthesis, optimization*

General Terms

Algorithms, Performance, Design, Theory

Keywords

Low-power Binding, Graph Theory, High Level Synthesis

1. INTRODUCTION

A portable device today is expected to perform high speed complex tasks consuming extremely low power. Complex functionality corresponds to more transistors on a chip and consequently more power consumption. Also portability demands lower power dissipation. Power consumption is so important that it should be optimized at all the steps of the design flow.

Binding is the process of ordering the operations on available resources such that the computation could be done successfully. Low power binding has been an active topic of interest during the past decade. The way binding is done drastically affects the power dissipation. Among the optimization techniques at different design levels, architectural and behavioral decisions influences the design the most.

ISLPED'03, August 25–27, 2003, Seoul, Korea.

Even though so much work has been done at the behavioral level, few of them have considered the general low power binding problem.

Given a set of operations in a scheduled DFG, the generalized low power binding problem, binds operations to R resources and picks the architecture for each resource such that the overall power is minimized. This is under the assumption that different implementations or architectures of a resource type are available.

It is shown in [5] that the problem is NP-complete. Our contribution in this paper is two polynomial-time heuristics for the generalized low power binding problem that result in very high quality solutions for our selected benchmarks.

In this paper, after initially discussing relevant work in the literature, the low power binding problem for fixed number of resources and single architecture of the resources is formulated as a min-cost flow problem that can be optimally solved [4]. The problem is further extended to the general case of having multiple architectures for the resources. The general binding problem is then formulated as an ILP problem that can be optimally solved but happens to be an NP-complete task [1]. Two polynomial-time heuristics are presented in this paper that result in very fast high quality solutions.

The first graph-based technique iteratively runs the single-architecture flow formulation for each architecture and then chooses the least power consuming assignment from the set of resulting candidates. Afterwards, it assigns the possible unassigned operations through a node coverage algorithm that follows another flow formulation. The node coverage algorithm runs iteratively until all the unassigned operations are covered.

The second technique assigns the operations to the resources of multiple architectures in incremental clock steps similar to an approach in the left edge algorithm.

Experimental results for 12 selected benchmarks shows a speed up of up to 13.7 with a maximum penalty of 2.5% compared to the optimal ILP solution.

Section 2 discusses previous work in the literature. Section 3 formulates the low power binding problem for the case of single and multiple architectures. Sections 4 and 5 describe the two proposed graph theoretic techniques. Experimental results are presented in section 6 and finally conclusion is made in the last section.

2. PRELIMINARIES

The initial steps of the design flow are extremely important as they impact the final implementation significantly.

In particular, architectural decisions during High Level Synthesis(HLS) have a profound impact on the power consumption of the final design. Therefore, power consumption is considered in high level synthesis tools as another optimization objective.

There are three phases in the behavioral synthesis. Scheduling takes a Data Flow Graph (DFG)and assigns clock cycle to each operation. Allocation determines the number of required resources and assignment or resource binding maps the operations to the resources. Scheduling is usually done prior to the other two tasks. Resource binding can be considered for functional units, registers and multiplexers. Binding of the functional units specially data intensive ones contributes substantially to power dissipation.

In a CMOS circuit, the major source of power dissipation is switching power which is directly proportional to the switching capacitance. Switching capacitance is defined as the product of the physical gate output capacitance and transition activity at the gate. Here for a fixed supply voltage, switched capacitance is an important optimization metric.

Most of the previous work in HLS for low power propose estimation and optimization of power consumption at algorithmic and architectural level. In [1], allocation has been formulated as an Integer Linear Programming(ILP) problem. The objective function is to minimize the overall switched capacitance in the data paths. In the ILP formulation, functional units, registers and multiplexers have been considered. In [4], external switching activity of a set of registers has been calculated following a statistical approach for single architecture of the resources. Using these values, register allocation and binding problem has been formulated as a minimum cost clique covering of an appropriately defined compatibility graph. In [5], the importance of computing the lower and upper bounds on power consumption are considered. Low power allocation and binding with and without constraint on the number of resources has been formulated. Based on the calculated power cost information of the functional units, efficient heuristics are proposed to measure these bounds. Here, the resources can be considered from different architectures. Allocation and binding of a scheduled DFG for fixed number of resources and architectures is an NP-complete task[5]. Hence, polynomial time algorithms that efficiently solve the problem of binding the operations to the resources with minimum power consumption need to get investigated. The ILP formulation in [1], can be considered for different architectures but the solution is not efficient. Register binding formulation in [4] is for single architecture only and in [5] the main concern is to calculate lower and upper bounds on power consumption of data paths.

In this paper, the low power binding problem of a scheduled DFG for fixed number of resources is extended considering different architectures of the resources. Since the extended problem is NP-complete, our contribution is polynomial-time heuristics that provide great speedup up to 13.7 with very little loss of quality from the optimal ILP solution.

3. LOW POWER BINDING PROBLEM

Given a scheduled DFG and fixed number of resources, the low power binding problem, assigns each of the DFG operations to a resource such that the overall switched capacitance of the resources is minimized. The problem can have two variations if the architecture of the functional unit is known or not. The generalized low power binding problem binds operations to the given R resources and picks up the architecture for each resource such that the overall power is minimized.

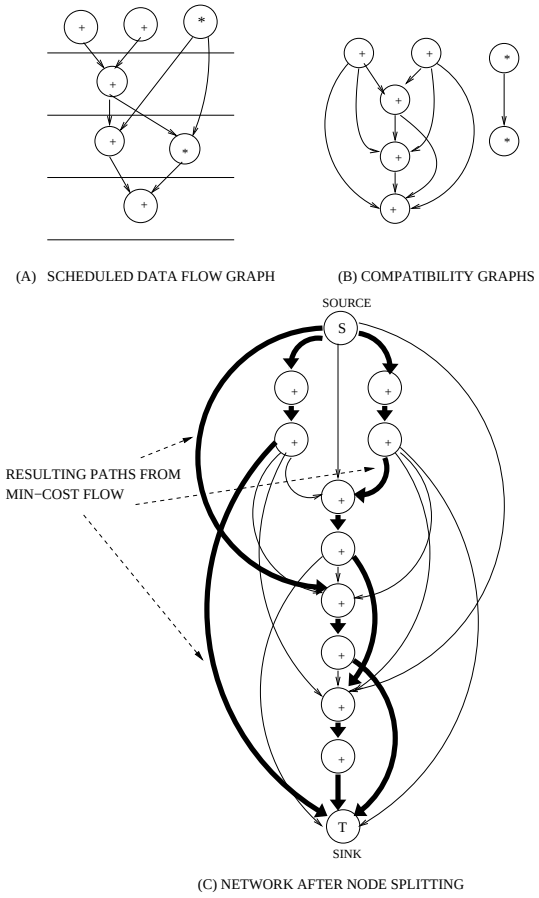


Figure 1: Single Architecture Low Power Binding

Assuming a single architecture for the time being for better insight, the formulation of the binding problem is described as follows based on work done in [4].

As illustrated in figure 1(A) and (B), the compatibility graph for a single architecture and operation type is generated from a given scheduled DFG. Each vertex of the compatibility graph corresponds to an operation. An edge is added between any two compatible nodes. Compatible nodes are operations that can be bound on the same resource.

Given the number of resources of a certain type as the resource constraint, the problem of binding the operations of the compatibility graph onto these resources is usually modelled as a min-cost flow formulation developed in [4] for a single architecture. The edge weights or cost values indicate the estimated switched capacitance on the given module if the source and destination of the edge form a predecessor and successor pair of operations on that resource. The cost values are calculated for the specific single architecture that is considered. Precise definition and calculation of the edge costs have been previously investigated in [5] and [1]. Similar to the register binding formulation described in [4], the node list is augmented by a sink(t) and source(s) with directed edges from the source to all the nodes and from all the nodes to the sink.

The non-sink/source nodes are further duplicated resulting in a network with vertex splitting as figure 1(C) illustrates. Considering any duplicated pair, any incoming edge to the original node enters the top one and any outgoing edge from the original node leaves the bottom one.

An edge is added between any pair of duplicated nodes. Here, the edge weight between each pair of duplicated nodes is -**W** where **W** is the summation of all the edge costs. Each edge has capacitance of unity.

Resource binding for a specific operation and architecture given **R** resources is solved by sending **R** units of flow from the source node to the sink such that each node is visited exactly once, all the nodes are covered and the overall cost is minimized. Note that it is assumed that **R**, the number of resources of a specific type is atleast the minimum number required for scheduling which is specified by the maximum number of operations scheduled on the same clock cycle. Figure 1(C) illustrates the sets of operations that are bound together. The operations on the same bold path in figure 1(C) are assigned to the same resource.

More formally the objective and constraints can be represented as:

$$Minimize \sum_{ij \in E} X_{ij} C_{ij} \tag{1}$$

$$\sum_{\forall i \, such \, that \, si \in E} X_{si} = R \tag{2}$$

$$\sum_{\forall i \, such \, that \, it \in E} X_{it} = R \tag{3}$$

$$\sum_{\forall j \, such \, that \, ij \in E,} X_{ij} = 1 \; \forall i, i \neq s, t \tag{4}$$

$$\sum_{\forall i \, such \, that \, ij \in E, i \neq s} X_{ij} = 1 \; \forall j, j \neq s, t \tag{5}$$

$$X_{ij} should \, be \, positive \, integer \tag{6}$$

Where
R: Number of Resources
C_{ij}: Switching Cost of an Edge ij
($Cost = 0$ if either $i \in s, t$ or $j \in s, t$)
($Cost = -W$ if(i, j) is a pair of duplicated nodes)

The above formulation can be implemented as a min-cost flow problem and then solved optimally under the assumption of single architecture.

As illustrated in figure 1(C), the algorithm results in exactly **R** discrete paths from sink to source. These paths are verified by bold lines in figure 1(C). Each path corresponds to a unit of flow. Each unit of flow represents a resource and the edges that this flow traverses decide the switching activity of this resource. Note that duplicated nodes with a connecting edge of capacity 1 ensure discrete paths in the final solution. Also having an edge weight of -**W** between each pair of duplicated nodes ensures coverage of all the nodes, as **W** will be more than the weight of any possible path.

Note that the proposed min-cost flow formulation does not consider re-iterative effects and focuses on a single architecture. An extension of this problem finds the best architecture from a library of architectures available for a resource type such that the overall switched capacitance is minimized.

In order to solve this problem, the formulation shown in figure 1(C) needs to be extended to consider multiple architectures as described in [2] and [1].

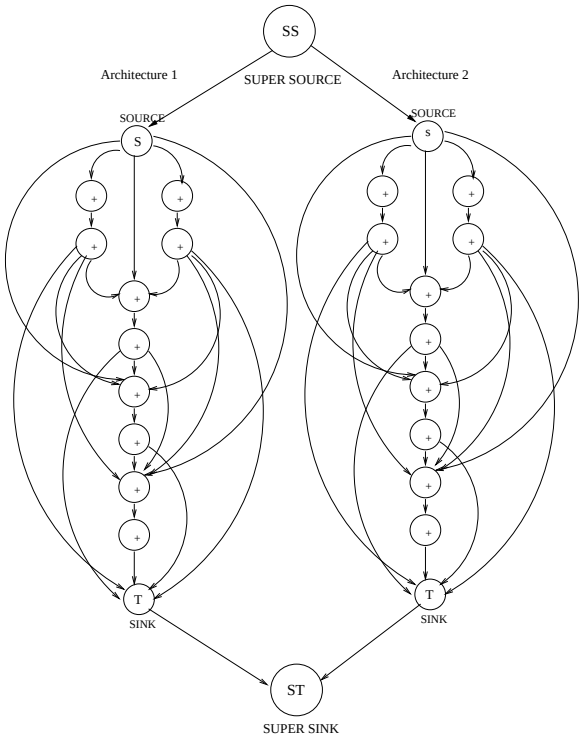


Figure 2: Multiple Architecture Low Power Binding

As figure 2 shows, let us suppose there are **M** different architectures and as before we also have a resource constraint (total number of resources available) of **R**. The problem is binding of operations together on **R** resources and deciding the architecture for each resource such that the overall switched capacitance is minimized. To address the issue of different architectures, the graph shown in 1(C) is replicated **M** times (figure 2). Hence a node i in the original compatibility graph occurs **M** times in this graph. Each s node in the graph is connected to a node called Super-source and each t node is connected to a Super-sink. We need to send **R** units of flow from the Super-source to Super-sink such that the overall cost is minimized. The constraints force us to use an operation in exactly one resource. Each of the replicated sub-graphs represent a particular architecture. The cost values of an edge in that sub-graph represent the estimated switched capacitance of having the source and sink of the edge as predecessors and successors on that particular architecture. The cost of the same edge in some other sub-graph will be decided by the corresponding architecture. The objective and constraints are formally described below. In order to better understand the formulation, we introduce the following notation. If n is a node in the original compatibility graph (refer to figure 1(B)), then it gets replicated M times. We denote each of the replicates as $n^1, n^2 .....n^M$.

$$Minimize \sum_{ij \in E} X_{ij} C_{ij} \tag{7}$$

$$\sum_{\forall j \, Super-source j \in E} X_{Super-source j} = R \tag{8}$$

$$\sum_{\forall j \, j Super-sink \in E} X_{j \, Super-sink} = R \tag{9}$$

$$0 \leq X_{Super-sourcej} \leq R \; \forall j Super - sourcej \in E \qquad (10)$$

$$0 \leq X_{j\,Super-sink} \leq R \; \forall j j Super - sink \in E \qquad (11)$$

$$\sum_{k:1..M} \sum_{\forall in^k \in E} X_{in^k} = 1 \forall n \qquad (12)$$

$$\sum_{k:1..M} \sum_{\forall n^k j \in E} X_{n^k j} = 1 \forall n \qquad (13)$$

$$\sum_{\forall in^k \in E} X_{in^k} = \sum_{\forall n^k i \in E} X_{n^k i} \; (k : 1..M, \forall n) \qquad (14)$$

$$X_{ij} \textit{should be positive integer} \qquad (15)$$

Equations 8, 9, 10, 11 represent the resource constraints. Equations 12, 13, 14 force each operation in the scheduled DFG to be in exactly one resource. Again this formulation does not consider re-iterative effects. The problem is NP-complete as a general case of the problem discussed in [5]. It could be solved optimally using any ILP solver.

Even though some previous papers have considered the extended version of the problem, no work has been done to provide fast solutions with negligible error for the generalized low power binding problem. In sections 4 a very high speed and accurate algorithm motivated by the min-cost flow formulation in [4] is proposed to solve the generalized resource binding problem. Section 5 presents another graph-based alternative. Our results are compared to the ILP solution resulted from the formulation above for multiple architectures.

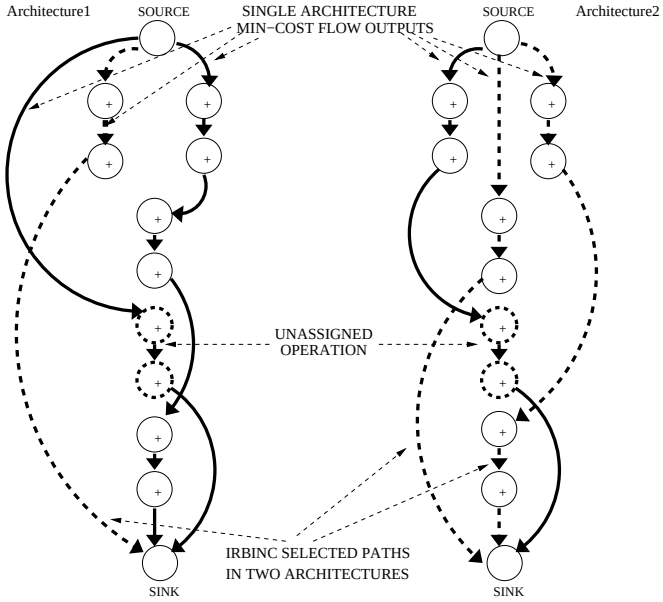
4. IRBINC: ITERATIVE RESOURCE BINDING ITERATIVE NODE COVERAGE

Given **M** different architectures and **R** different resources of the same type, consider running the min-cost flow algorithm iteratively for **M** times for different architectures. The edge costs in each graph relates to its corresponding architecture. From each graph, **R** discrete paths will be obtained.

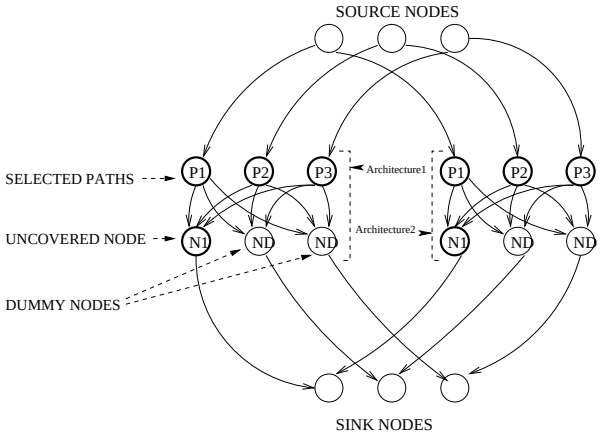
The goal here is to select up to **R** disjoint paths that would cover all the nodes with the least cost from the available **R**×**M** set. IRBINC iteratively selects the path with the smallest weight from the set of available paths. Whenever a path is selected and added to the solution set, all of the overlapping paths from the available set will be removed. IRBINC iterates until the available set becomes empty meaning that each path that is not selected has at least one overlap with a path in the final solution set.

Consider figure 3(A). The single architecture flow formulation of the previous section results in 3×2 paths here for 2 architectures and 3 resources (6 bold and dashed paths). After the iterative path selection, 3 final paths are obtained illustrated by the dashed lines. As can be seen the first path is from the first architecture and the second and third ones are from the second architecture for this example.

At this point up to **R** paths are obtained but there is still possibility of some uncovered nodes which are the operations that have not yet been assigned to any resource as the example of figure 3(A) shows.



(A) ITERATIVE SINGLE ARCHITECTURE MIN−COST FLOW SOLUTION



(B) COVERING UNASSIGNED OPERATIONS

Figure 3: Node Coverage Algorithm

We proposed a node coverage algorithm that simultaneously assigns the uncovered nodes of the same clock cycle in the DFG to the resources obtained before. It runs iteratively for each group of unassigned operations until all the nodes are covered. Each group contains unassigned operations in the same clock cycle.

The strategy of the node coverage algorithm is to assign the uncovered nodes into these **R** paths with the smallest overall cost following another flow formulation. The node coverage algorithm described below runs iteratively until all the uncovered operations are assigned.

Node coverage algorithm: Given up to **R** selected paths (assigned resources) and **P** uncovered nodes (unassigned operations) of the same clock cycle in the DFG, a new graph is constructed as follows. Consider a node for each path and a node for each uncovered operation. If the number of uncovered nodes is less than the number of resources(**P**<**R**), dummy nodes will be added to make it equal.

An edge goes from a "path-vertex" to an "operation-vertex",

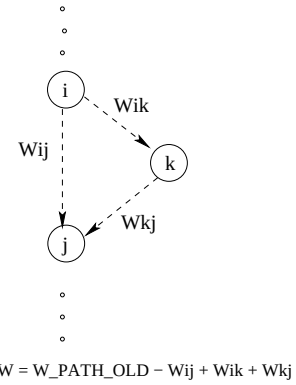


Figure 4: Updating a path cost

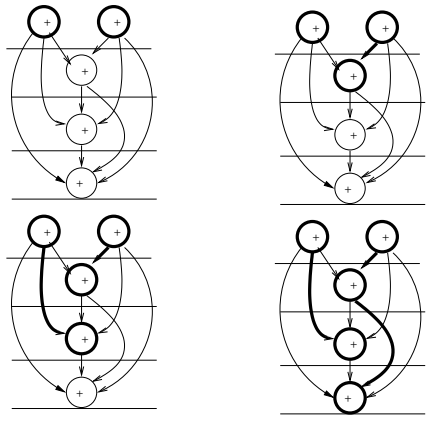


Figure 5: Development of paths in CTDA

if the node can be augmented in the corresponding path [there is no operation scheduled on the path in the related clock cycle]. As figure 3(B) illustrates, the described structure is repeated **M** times. **R** source and **R** sink nodes are added. Each source node corresponds to a path and each sink node corresponds to an uncovered operation. Directed edges of 0 cost goes from each source node to its corresponding path-vertex and from each of the **R**×**M** operation-vertices to the corresponding sink node. All the edge capacities are unity to ensure disjoint solutions.

The edge costs are modified correspondingly for each architecture. The cost of an edge from a path-vertex to an operation-vertex is the cost of the new path obtained after augmenting the node as figure 4 illustrates. The cost of an edge to a dummy node is simply the path cost.

By sending one unit of flow from each source node to any of the sink nodes such that all the costs are minimized, up to **R** new paths are obtained that cover the **P** uncovered operations simultaneously. The edge from a path-vertex to an operation-vertex having a positive flow decides the coverage of the corresponding operation in the corresponding path. The subgraph that this edge occurs at, decides the architecture.

Note that if the actual number of paths from the iterative resource binding is smaller than the number of resources, some of the path-vertices will be "dummy paths". A dummy path covering an operation-vertex corresponds to a new resource that will be assigned to an unassigned operation. The node coverage algorithm considers the set of unassigned operations simultaneously for different architectures and resources. It runs iteratively for each set of unassigned operations of the same clock cycle in the DFG until all the nodes are covered.

Complexity Analysis: The single architecture min-cost flow is of $O(n^3)$ for n operations and runs iteratively M times for each architecture. Iterative path selection is $O(RMlog(RM))$ for $R \times M$ paths. Each iteration of the node coverage algorithm is of $O((2R + 2RM)^3) = O((RM)^3)$, as figure 3(B) illustrates, the total number of nodes for any such graph is $2R$ for the source and sink nodes and $2R$ nodes for each architecture. The node coverage algorithm iterates at most T clock steps. Therefore the running time of the IRBINC algorithm is described as $O(Mn^3 + RMlog(RM) + TR^3M^3) = O(Mn^3 + TR^3M^3)$ which is a polynomial expression.

IRBINC considers selection of different architectures of each resource at the time of resource binding. Our experimental results show a very high quality solution and great speed up compared to the optimal ILP discussed in section 3. In section 5 another heuristic with similar characteristics is presented.

5. CTDA: CONSTRUCTIVE TOP DOWN ASSIGNMENT

We propose another approach in which the solution is generated in a constructive manner. The scheduled DFG is traversed in a top down fashion from clock step 1 to clock step T, hence the name *Constructive Top Down Assignment*. Starting from clock-1, we have a set of operations in clock-1 that need to be bound onto **R** resources. We arbitrarily make this assignment to the available resources. The clock steps are traversed iteratively in a top down fashion. Let us suppose we have traversed the first i clock steps. At this stage we have an intermediate solution in which a set of operations belonging to the first i clock steps have already been bound to **R** resources. Clock $i + 1$ contains a set of operations that need to be bound onto these **R** resources. This can be done using the **node coverage algorithm** presented in the previous section. That formulation assigned operations in the same clock step to a set of **R** resources on which some operations have already been bound. There is one key difference though. The cost of an edge connecting a path-vertex to an operation-vertex will be decided by the cost of the compatibility edge between the operation and the last operation bound (in terms of clock step) onto that resource. This formulation simultaneously assigns operations in the same clock step to available resources and also selects the architecture with power as the optimization objective. Then we proceed to the next clock step, $i + 2$ and the same thing is repeated.

Figure 5 illustrates running of the algorithm. The operations in bold are the ones that have been assigned, the bold edges signify the paths or sequence of operations on the corresponding resource. It can be seen that operations are considered one clock at a time in a top down fashion. Please note that this strategy is similar to the left edge algorithm on interval graphs. The scheduled operations could be considered as intervals in a temporal sense with finite lifetimes (starting time to ending time) indicating lengths or intervals of nodes. We proceed on this so-called interval graph in a left edge (or top down) fashion, solving a local binding problem optimally at each step. The global algorithm is of course a heuristic.

Complexity Analysis: At most R operations are covered at each clock cycle. Assuming T clock steps, the node coverage algorithm will be called at most T times. Each iteration of the node coverage algorithm is of $O(R^3M^3)$ as described in the previous section. Therefore the time complexity of CTDA is $O(T \times R^3M^3)$.

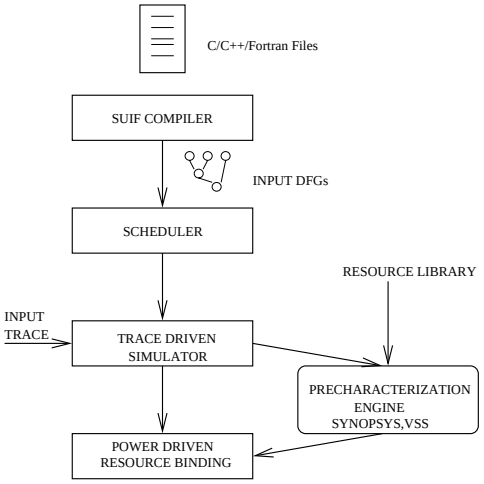


Figure 6: Experimental Flow

Bench	IRBINC		CTDA		ILP
	Time	Speedup	Time	Speedup	Time
fft-2	1.00	3.87	0.73	5.30	3.87
motion-2	0.76	10.70	1.20	6.75	8.13
motion-3	0.80	10.12	1.73	4.68	8.10
noisest-2	0.73	5.89	4.48	0.96	4.30
ecb-enc-4	1.17	13.7	1.30	12.33	16.03
dct	1.03	1.52	0.70	2.24	1.57
ellipt	1.77	2.56	1.57	2.88	4.53
jdmerge-1	0.73	7.85	1.00	5.73	5.73
jdmerge-3	0.87	11.49	2.03	4.93	10.00
jdmerge-4	0.63	9.36	0.80	7.37	5.90
fir	0.80	1.25	0.40	2.50	1.00
jctrans2	0.70	5.71	0.60	2.40	4.00

Table 1: Performance of the Proposed Techniques

6. RESULTS

The experimental environment was setup similar to [2]. The design flow is shown in figure 6. We take functions written in C and extract data flow graphs from them using SUIF/Machine-SUIF. These DFGs are then scheduled using a path based scheduler [6]. We also take an input trace which is a representative of the input statistics for the DFG. Based on this trace, the DFG is simulated and the values for all the internal variables of the DFG are computed. This information along with the scheduling information is given to synopsys design compiler and the VSS simulator which generate the switched capacitance information. These values correspond to the edge costs for the compatibility graph of the corresponding schedule.

We used the media bench suite [3] together with some HLS benchmarks to extract the data flow graphs. Our experiments are done on a Solaris 2.7 operating system (Sparc edition) with 256M memory. Here, power optimization for only additions are considered from different operation types in the DFG. Table 1 illustrates the performance of the two proposed methods compared to the ILP case for multiple architectures formulated in section 3. Both the IRBINC and the CTDA result in great speedups up to 13.7. Even though the speedup of the two methods are similar, in the case of motion-3, noisest-2 and jdmerge-3, IRBINC results in greater speedups. The proposed methods have better performance on large benchmarks. As can be seen in the case of dct which represents a small benchmark, the speedup is really small.

Bench	IRBINC		CTDA		ILP
	Pnlty	SwitchCap	Pnlty	SwitchCap	SwitchCap
fft-2	2.49	130.78	0.93	128.79	127.60
motion-2	0.01	163.33	2.19	166.91	163.32
motion-3	0	158.11	2.48	162.04	158.11
noisest-2	0.47	107.98	1.09	108.64	107.47
ecb-enc-4	1.45	196.76	0.57	195.04	193.94
dct	0.2	79.78	0.64	80.13	79.62
ellipt	1.19	179.54	2.28	181.48	177.43
jdmerge-1	0	161.07	0	161.07	161.07
jdmerge-3	0.46	219.00	0.70	219.52	217.99
jdmerge-4	2.50	138.53	2.58	138.64	138.15
fir	0.67	59.75	0	59.35	59.35
jctrans2	0	66.46	0	66.46	66.46

Table 2: Penalty of the Proposed Techniques

The average speedup from the 12 benchmarks is 7.00 and 4.84 for IRBINC and CTDA respectively.

Table 2 shows the power cost penalty of both of the schemes to be really small. The maximum penalty is only 2.58% in the case of jdmerge-4. Average penalties are 0.78% and 1.12% for IRBINC and CTDA respectively.

As the results show, simple and fast heuristics make enormous difference in the speedup with a negligible deviation from the optimal solution.

7. CONCLUSION

This paper formally discusses the low-power binding problem for multiple resources and architectures. It introduces two fast polynomial-time alternatives to the optimal ILP approach. They result in a maximum speedup of 13.7 with a maximum deviation of only 2.5% from the optimal solution. The proposed algorithms provide much better speedups for larger benchmarks which should make them very suitable in real-world applications. In the end, we would like to point out that the above formulation can be useful in similar optimization problems that more than one choice/implementation exist for the inputs.

8. REFERENCES

[1] A. Raghunathan and N. Jha. "An ILP Formulation for Low Power Based on Minimizing Switched Capacitance During Datapath Allocation". In *Procs of IEEE Symposium on Circuits and Systems*, 1995.

[2] A. Srivastava. "Predictability: Definition, Analysis and Optimization". In *Procs of International Conference on Computer Aided Design*, Nov 2002.

[3] C. Lee, M. Potkonjak and W.H. Mangione-Smith. "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems". In *International Symposium on Microarchitecture*, 1997.

[4] J.M. Chang and M. Pedram. "Low Power Register Allocation and Binding". In *Proc. Design Automation Conference*, pages 29–35, June 1995.

[5] L. Kruse, E. Schmidt, G. Jochens, A. Stammermann, A. Schulz, E. Macii and W. Nebel. "Estimation of Lower and Upper Bounds on the Power Consumption from Scheduled Data Flow Graphs". In *IEEE Trans. on VLSI Systems*, pages 3–14, Feb 2001.

[6] S. Ogrenci-Memik, E. Bozorgzadeh, R. Kastner and M. Sarrafzadeh. "A Super Scheduler for Embedded Reconfigurable Systems". In *Procs of International Conference on Computer Aided Design*, pages 391–394, Nov 2001.