Pattern Selection: Customized Block Allocation for Domain-Specific Programmable Systems

Elaheh Bozorgzadeh Seda Ogrenci Memik Ryan Kastner Majid Sarrafzadeh

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095-1596, USA
{elib,seda,kastner,majid}@cs.ucla.edu

ABSTRACT

The increase in complexity of integrated circuits results in the need to develop programmable hardware platforms shared among a set of applications. Patterns are application specific computational units. The patterns embedded on the target system are selected by exploiting regularity among the applications in the same domain. The number of patterns nominated by applications to be embedded on systems can be large. There can also exist overlap between the patterns. We present a gain model which can represent different characteristics of patterns. Using this gain model, we propose our algorithm, pattern selection, to select a set of patterns such that the objective is maximized. The experimental results show that our method chooses different sets of patterns when area limit for embedded cores on the system changes. We used our method to select a set of embedded modules on the SPS architecture [3] for multimedia applications. Comparing the results obtained by our algorithm with the method in which only common patterns are chosen to be embedded, latency and utilization of embedded patterns can be improved by 11% *and* 25%*, respectively.*

Keywords: Programmable System Architecture, Highlevel Synthesis, CAD and Optimization.

1. INTRODUCTION

In the next generation of electronic systems, it is expected that the conventional embedded systems are unlikely to be sufficient to meet the timing, power, and cost demanded by the applications. The diversity and increasing number of applications do not allow the fully customized system design methodology for each application like conventional ASIC designs. One of the fundamental keys is integrating programmability and reconfiguration in the systems [1]. Another important issue which supports programmability and re-use in future system is the high cost of system design and manufacturing.

A platform-based system is shared among a set of applications belonging to the same domain such as multimedia, networking, encryption, etc. In order to implement an application in particular domain on a platform-based system, partial reconfiguration of the system is needed for remaining customization. In addition, the system has to be capable of handling future applications of that domain only by minor effort and modification in the system [1]. Such a system consists of fixed cores and reconfigurable (or flexible) components. The fixed components are optimized for speed, performance and density. The reconfigurable components enable the implementation of different resources needed by multiple applications. Therefore, the increased complexity in integrated circuits of SoC (System-on-a-Chip) design can be handled by integration of programmability into system (Hybrid ASIC). In this methodology, there is a combination of IP cores, programmable logic cores, and memory blocks on a chip. Several companies already provide new design methodologies integrating programmability.

Reconfiguration is mostly integrated into the computational abstraction of a design where repetitive high-volume operations are executed. As mentioned earlier, new design methodologies are based on the fact that domain-specific system design is required to deliver the benefits of programmability while having almost same performance. The cost of programmability is in power and timing. Exploiting regularity in applications leads to decrease in cost of programmability and lower total cost and development time.

FPGAs are programmable hardwares. There have been several contributions in development and design of FPGAs towards reducing the gap in density and performance between ASIC and FPGA implementation. Hierarchical features have been added into logic and routing architecture of FPGAs. New generation of FPGAs have a trend towards embedding coarse grain units. New architectures with memory blocks, hard IPs, and even CPUs are being integrated into FPGAs. As an example, $Xilinx^{TM}$ Virtex-II architecture provides dedicated high performance multipliers for DSP applications.

We have introduced a new programmable system design methodology called SPS, which is referred as Strategically Programmable System [3, 4]. The SPS architecture includes parameterized functional blocks (called *VPBs*) that are pre-placed in a fully programmable logic array. Our motivation is to generate a programmable system suited for a set of applications. SPS architecture is generated such that for a given set of applications the suitably selected fixed blocks provide the best performance. The angle taken in SPS project [3] is that both fine grain and coarse grain reconfiguration are required. Since the applications belong to the same domain, exploiting regularity would be effective in choosing IP Cores, random logic and other components of the chip. Each application demands different embedded modules. For the final design, we need to select a set of the modules demanded by a set of applications such that the implementation is cost effective while meeting performance requirements for all the given applications. It is important to come up with a platform, which meets most of the demands of all applications. The idea of having dedicated data-path according to the demand of a specific application is not a new methodology. This paradigm has been studied in DSP architecture design in [5]. An architecture-driven high level synthesis of DSP applications, called Cathedral III, is presented. In this work, reconfigurability is not considered and it cannot easily handle multiple applications. The main focus in

Cathedral III architecture design methodology is on generating and merging operation clusters and assignment to minimize the area.

Our contribution in this paper is how to select the best set of modules demanded by multiple applications to be embedded as fixed blocks (VPBs) in SPS architectures. We have implemented our method at data-path level. The embedded modules demanded by each application are already given as input to our tool. Data-path structure retains the regularity and reduces the problem size compared to work on gate-level of the circuits. Demanded modules (or patterns) are already extracted from data flow graphs of applications. The output is a set of patterns suggested to be embedded as fixed cores on the system by our algorithm. Our experimental results show that we are able to improve the latency depending on how much area is assigned for IP cores and random logic on the chip.

The paper proceeds as follows: In Section 2, programmable hybrid systems are briefly described as well as SPS design flow. In Section 3, the pattern selection problem is formulated followed by our pattern selection algorithm. The experimental setup and results are explained in Section 4. Conclusion and some suggested possible future work are given in Section 5.

2. GLOBAL APPROACH

An IP core is a module which is custom-designed to realize one or more functions. In a reconfigurable hybrid structure, each operation in a data flow graph of an application may be realized by reconfigurable logic units or a customized functional unit. We refer to the application specific units other than reconfigurable blocks as *pattern candidates*.[1]. Therefore, there is a gain and a cost associated with each candidate module. The gain comes from the better performance of embedded patterns compared to reconfigurable units. The cost comes from the non-flexibility of hard-wired IP blocks by which not every function can be realized compared to reconfigurable cores. Utilization of silicon is an important target. If the pattern has not already been customized for target architecture (or similar customized design is not available in other libraries), the effort to custom-design a pattern is more costly due to design effort. Customized blocks are mostly designed manually in today's design.

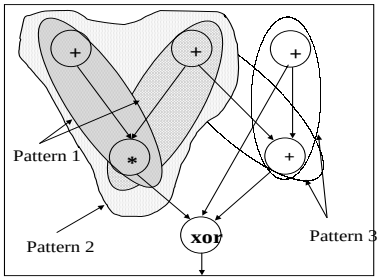

Figure 1: Different Embedded Pattern Candidates on a Data Flow Graph.

Different set of pattern candidates are demanded by each application. Most of patterns are suggested either due to high frequency of occurrences in application or due to their high performance compared to other components. Therefore, considering multiple applications, IP utilization and contribution of embedded customized blocks in performance of applications implemented on target architecture requires to develop an optimization technique to select the patterns. In [**?**], it is suggested that the intersection of different sets of constraints demanded by different applications defines the hardware platform. However, intersection is not necessarily a good way to decide on the functionality of embedded components of hardware.

As mentioned in Section 1, the dedicated data path blocks on target systems are not taken from a fixed given library but are extracted from each application and synthesized. A good way to extract such pattern candidates is profiling. Profiling is mostly applied to DAG representation of systems. For example, we extract control data flow graphs (CDFG) from a compiler [4]. Pattern generation via profiling have been done in several previous works. In [8], sub-graph matching is applied to extract the patterns in each application, which are critical in performance or any other objective function such as power consumption. In [7], data path synthesis is based on pattern extraction from data-path of an application.

Another important issue in pattern selection is when different patterns *overlap* on some DFGs. Figure 1 shows profiling on a data flow graph (DFG). In Figure 1, different extracted modules (or patterns) on the data flow graphs are shown. Pattern 1 has been observed two times in Figure 1. There are overlaps between Patterns 1, 2, and 3. If Pattern 1 and Pattern 2 overlap in application C, only one of them would be used by application C. If both patterns exist on the chip, the chip would not be fully utilized when application C is implemented.

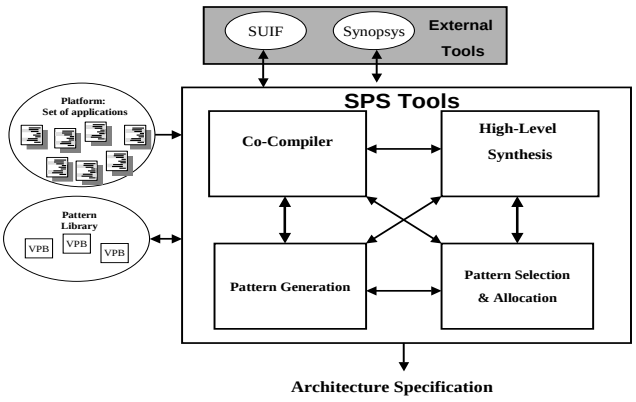

Figure 2: Pattern Generation in Platform-based System Design.

Figure 2 shows the SPS design flow. Data flow graphs of applications are extracted using a compiler tool. Then a pattern generator profiles different patterns demanded by different applications (See [4] for more details). Since the patterns are generated among a set of applications in the same domain, regularity among applications results many frequently-visited patterns to be nominated. The objective is to select a set of pattern candidates such that highest gain in performance is obtained, utilization of the chip is maximized, and maximum covering by patterns is achieved. In overlap cases, pattern selection would guide the co-compiler where and when to generate the patterns in the DFGs. After that, the pattern-based DFGs are scheduled. Before scheduling, the real effect of selected modules are not known yet. Scheduling can guide the pattern selection tool to give higher priority to more critical modules. Pattern Selection would then select new sets of patterns.

[1] In this paper, the terms "pattern" and "module" have been used interchangeably.

3. PATTERN SELECTION PROBLEM

We assume that the following data is provided for pattern selection tool.

1. There is a gain associated with each pattern. We assume that the gain is the difference between implementation on embedded fixed blocks and reconfigurable (e.g. FPGA) implementation of each pattern in terms of area and delay. A possible gain function could be the difference in area-delay product of components.

2. The area assigned for embedded basic blocks on the chip is restricted. In order to obtain maximum utilization, all pattern candidates cannot be chosen to be embedded as hard-wired ASIC blocks on the chip.

3. The frequency of occurrences of each pattern candidate in DFGs is an important factor. The number of occurrences does not imply the number of resources for implementation of pattern candidate required since the embedded fixed blocks associated with a pattern can be re-used later. In Figure 2, the number of demanded patterns is given to pattern selection tool by ASAP Scheduler. ASAP scheduler returns the maximum number of instances of each pattern used in scheduled DFGs.

4. Overlap is an important issue, which needs to be reported by pattern generator.

The problem is similar to resource allocation on scheduled data flow graph in high level synthesis. Resource allocation problem is resolved by solving graph coloring problem in conflict graph. However, this solution cannot be applied to our defined problem. There are two main differences. First is that overlap does not mean that two resources cannot be chosen to be embedded on the chip. However, it leads to less utilization. The other problem is due to decision problem on the number of instances of each candidates required in target architecture. It is not easy to handle this in conflict graph.

3.1 Gain and Overlap Model

Assume that there is an overlap between two patterns where they are both observed in a given data flow graph. If a resource for each pattern is embedded on the chip, both will not be used by the application. Assume g_i and g_j are gains associated with pattern i and pattern j. During scheduling, either of the two patterns will be used when overlap occurs. We assume that any of the two patterns b_i and b_j can be used with probability of p_i and p_j, respectively. The total gain would not be only the summation of both gains. We define the total gain as follows:

$$g_{total}(i,j) = p_i \times g_i + p_j \times g_j, \quad (1)$$

where $p_i + p_j = 1$. The Equation 1 can be rephrased as follows:

$$g_{total}(i,j) = g_i + g_j - overlap_{unit}(i,j), \quad (2)$$

where

$$overlap_{unit}(i,j) = (1-p_i) \times g_i + (1-p_j) \times g_j. \quad (3)$$

Here we assume that in case of overlap each of the pattern can be selected with equal probability ($p_i = p_j = 0.5$). In addition, the frequency of occurrences of each pattern has to be considered while computing gain of the pattern for an application. Assume the number of times pattern i and j have been observed in an application are occ_i and occ_j respectively. c_{ij} is the number of times pattern i and j have overlap in a given application. Assume d_i and d_j are the number of resources demanded by an application. If there are lesser number of embedded resources for pattern i and j, the overlap and gain can be approximately computed as follow respectively:

$$Overlap(i,j) = \frac{r_i}{d_i} \times \frac{r_j}{d_j} \times c_{ij} \times overlap_{unit}(i,j), \quad (4)$$

$$gain_i = \frac{r_i}{d_i} \times occ_i \times g_i, \quad (5)$$

$$g_{total}(i,j) = gain_i + gain_j - Overlap(i,j), \quad (6)$$

where $r_i (0 \leq r_i \leq d_i)$ and and r_j $(0 \leq r_j \leq d_j)$ are the number of available resources of pattern i and j. Equation 6 can be extended for all applications by summing the gain function over all applications as shown in Equation 7.

$$Gain = \sum_{k=0}^{app} (\sum_{i=0}^{n} f(i,k) \times occ_{i,k} \times g_i - \frac{1}{2} \times \sum_{j=0, i \neq j}^{n} Overlap(i,j,k)) \quad (7)$$

where

$$f(i,k) = \begin{cases} \frac{r_{i,k}}{d_{i,k}} & r_{i,k} \leq d_{i,k} \text{ and } d_{i,k} \neq 0 \\ 0 & d_{i,k} = 0 \\ 1 & \text{otherwise} \end{cases}$$

The subscript k in the coefficients and variables of Equation 7 shows their corresponding value in application k.

3.2 Problem Formulation

Pattern selection problem can be formulated as follows:

- Given a set of pattern candidates, $P = \{p_1, p_2, ..., p_n\}$ with corresponding Gain set ($G = \{g_1, g_2, ..., g_n\}$), Area set ($A = \{a_1, a_2, ..., a_n\}$, Occurrence of each pattern i in each application j (o_{ij}), and Demand Set in each application j for each pattern i (d_{ij}),

- The objective is to choose $R = \{r_1, r_2, ...r_n\}$ such that the total gain function in Equation 7 is maximized.

- Subject to: $0 \leq r_i \leq max(d_{ij})$ and $\sum_{i=0}^{n} r_i \times a_i \leq A_{max}$, where $j = 0, 1, ..., k$ and $i = 0, 1, ..., n$ and where A_{max} is the maximum area assigned for embedded pattern blocks.

The objective function is a non-linear function. If only one application is considered, the problem can be solved by a quadratic programming solver. However, considering multiple applications cause the objective to be non-linear and hard to be solved by any of linear/quadratic programming tools.

In the next sub-section, we introduce *overlap graph*. We show that any instance of pattern selection problem can be transformed into overlap graph. Due to space constraints, the corollaries and lemmas stated without proof; please see the technical report for further details [2].

3.3 Overlap Graph

Overlap Graph is an undirected weighted Graph $G = (V, E)$. Weight of edges can be negative. Each node corresponds to a pattern candidate. The label of the node is the area associated with pattern candidate (original nodes). An edge between two nodes corresponds to overlap between the two pattern candidates. Since overlap is a cost, the weight is negative of the value returned by function $Overlap(i, j)$ in Equation 4. For each node in the graph (or called *original* node), a dummy node is added. The dummy node is connected to original node with an edge. The weight of the edge is the gain of the node (Equation 5). The labels of the dummy nodes are zero. Figure 3 shows an overlap graph. Pattern selection problem is transformed into problem of extracting an induced subgraph of G such that the summation over weights of edges inside the subgraph is maximized subject to summation over labels of nodes inside the subgraph does not exceed a given limit (see Figure 3). Lemma 3.1 mentions the properties required for an induced subgraph in order to be a feasible solution of the corresponding pattern selection problem. The first property is the area constraint. The other property says that correct number of dummy nodes has to exist in the subgraph.

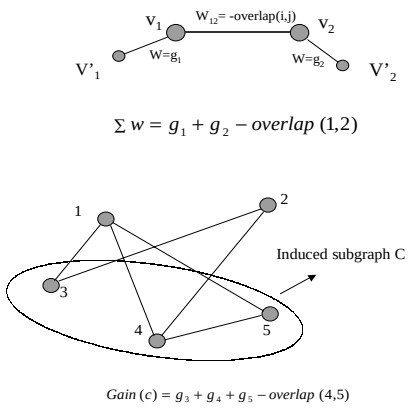

Figure 3: Overlap Graph.

Lemma 3.1. *Summation over weights of edges in an induced subgraph of overlap graph corresponding to pattern selection problem returns objective value for pattern selection problem if the conditions below are satisfied:*

- *the summation over the labels of the nodes insides the subgraph is less than a given limit, which is equivalent to area limit for embedded patterns in the system. This is referred as area constraint.*
- *Dummy node associated with any original node is included in the subgraph iff the original node is itself inside the subgraph. This is referred as gain constraint.[2]*

We modify the graph in order to be able to handle the multiple instances of patterns and multiple application demands. . Figure 4 shows two nodes of overlap graph with corresponding labels and weight of the edge between two end nodes. Assume v_1 represents Pattern 1 and v_2 represents pattern 2. There are two number of resources for pattern 1 ($d_1 = 2$) and three number of resources for pattern 2. The edge between the two nodes shows the overlap between the patterns. According to Equation 6, the weight of the edge is:

$$w_{i,j} = -Overlap(i, j) \tag{8}$$

$$w'_{i,j} = -\frac{1}{d_i} \times \frac{1}{d_j} \times c_{ij} \times overlap_{unit}(i, j) \tag{9}$$

$$w_{i,j} = r_i \times r_j \times w'_{i,j} \tag{10}$$

$$w'_{d_i} = \frac{1}{d_i} \times occ_i \times g_i \tag{11}$$

$$w_i^{dummy} = r_i \times w'_{d_i} \tag{12}$$

The weight of edge depends on the number of instances selected from node v_1 and v_2. In overlap graph, node v_1 and node v_2 are divided into d_1 and d_2 sub-nodes, respectively, shown in Figure 4. There is an edge with weight w' between any two instances of node v_1 and v_2. We refer to w' as *unit overlap weight*. Each instance of a super node is represented by a subnode connected to the graph through sub-edges. As shown in Equation 9 and 10, overlap between two nodes can be defined in terms of unit overlap weight between the two supernodes and number of subnodes of the two overlapping supernodes in overlap graph. Equation 11 and 12 similarly show that individual gain of each supernode can be expressed in terms of number of subnodes and unit gain. The subgraph shown in Figure 4 includes two instances of v_1 and two instances of v_2. The total summation over weights of subedges is equivalent to objective function for pattern selection problem. The label of each subnode is the area of the instance.

Overlap graph representation can be easily extended to handle multiple applications. In this case, the number of occurrences and demands for each pattern candidate is different. We consider multiple edges between super nodes. Each edge corresponds to overlap between the two end supernodes of the edge in one application. Similarly there would be multiple subegdes between subnodes for each supernode. In addition, we define multiple occurrences for each supernode. We duplicate each supernode to subnodes as many as maximum number of resources demanded by all applications for the corresponding pattern candidate.

As shown in Figure 5, each subnode corresponds to an instance of a supernode. Assume that application A demands 3 resources for pattern node I in overlap Graph. Application B demands two instances of supenode I. Supernode I will be duplicated three times. Therefore, it consists of three subnodes connected through subedges to supernode II. The first set of subedges is according to overlaps between node I and supernode II in application A. Overlap graph is a multigraph, i.e. a graph with more than one edge between two nodes in the graph. Since two instances of node I were demanded by application A, there exist a subedge between the first two instances of node II and the first two instances of supernode II .In order to handle this issue, we need to index the subnodes of each supernode. Index i of pattern I corresponds to i^{th} resource of pattern I.

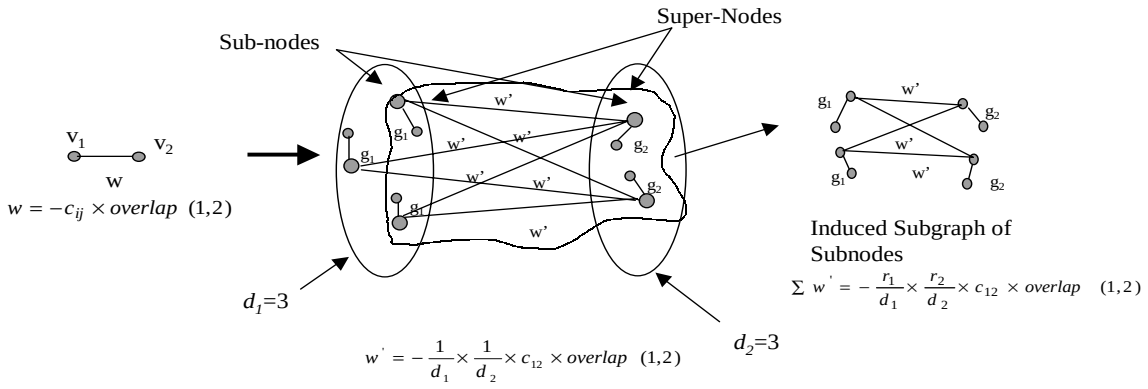

Figure 4: Supernodes and Subnodes in Overlap Graph

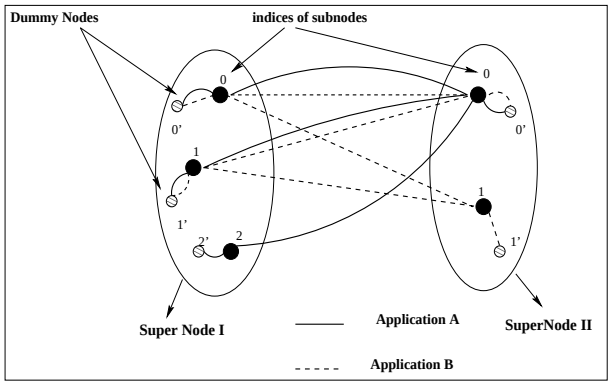

Figure 5: Overlap MultiGraph

3.4 Pattern Selection Algorithm

In this section, we propose an algorithm for pattern selection problem formulated in Section 3.2. Problem of finding an induced subgraph in a graph is equivalent to clustering a subset of nodes in the graph. In pattern selection problem, we generate a single cluster on overlap graph.

Theorem 3.1. *Problem of generating a single cluster on overlap graph including edges with negative weights such the total summation of the weights of the edges inside the cluster (induced subgraph) is maximized, is NP-Complete [2]*

According to Theorem 3.1, the problem is an intractable problem. Therefore, a heuristic has to be developed to solve the problem. We propose a simple and fast heuristic to solve this problem. We support sequential clustering approach. In sequential approach, one node is chosen at a time to be added to the cluster. As mentioned in Lemma 3.1, the cluster must include two properties. Our clustering algorithm has to satisfy both area constraint and gain constraint. Area constraint can be easily handled. When a node is being added to the cluster, it is checked if the area constraint is satisfied. This means to see if the label of the node added to the total labels of nodes in cluster is less that a given limit. Otherwise the node is not a feasible choice and cannot be added to the cluster. In order to satisfy gain constraint the clustering algorithm has to deal with two cases. While constructing the cluster we avoid choosing any dummy node before selecting the node associated with the dummy node. Therefore, no redundant dummy node would be added. The other issue is that we have to guarantee that subgraph includes each dummy node associated with nodes inside the subgraph.

In sequential clustering, nodes are selected to be added to the cluster iteratively. We refer to the node being selected as *candidate* node. There are constraints which need to be satisfied by this candidate. We choose each candidate based on local closeness to optimality. Each candidate node is the one increasing the objective value the most. However, it might not be feasible. In terms of feasibility, the node should satisfy the two constraints in Lemma 3.1. Otherwise the next best candidate is chosen. A function called *potential* function is used to judge among the candidates.

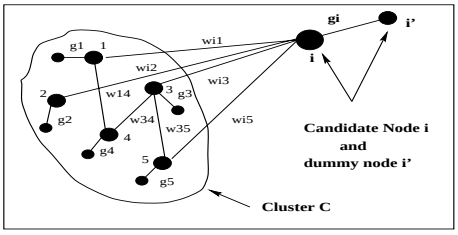

Figure 6: Candidate Node i Being Added to Cluster C.

Figure 6 shows a candidate node (node i) while being added to a cluster C. Node i' is the dummy node associated with node i in overlap graph. The objective value for current cluster is the summation over the weights of edges inside the cluster. If node i is added to cluster C, the change in the value of the objective function would be the total weights of edges connecting node i to the nodes inside cluster.

We define potential function $p(i, C)$ for each candidate i as shown in Equation 13, which is related to the amount of change in objective value. Since there is an area constraint, we consider utilization in our potential function by dividing to the area of the embedded pattern candidate, i.e. the label of the node in overlap graph.

$$p(i, C) = \frac{\sum_{j \in C} w_{ij} + g_i}{l_i} \quad (13)$$

Lemma 3.2. *A required condition for a candidate node to be*

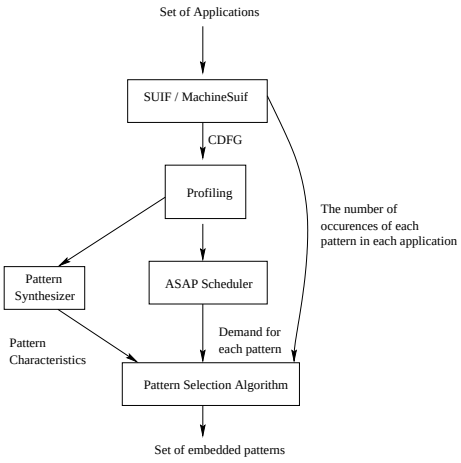

Figure 7: Experimental Flow.

added to cluster in sequential clustering is that $\sum_{j \in C} w_{ij} + g_i > 0$*, if the current nodes in cluster are assumed not to be pruned later [2].*

This condition avoids selecting a dummy node if the original node associated with the dummy node is not already in cluster. The associated dummy node is connected to the graph only through the corresponding original node. Hence, if the node is inside the cluster, the dummy node has a positive potential function value towards the cluster. Otherwise the potential function value for such dummy node is zero. Therefore this condition satisfies the gain constraint mentioned in Lemma 3.1.

Another important issue is that potential gain of each node i inside cluster C should not become negative after adding the candidate.

Lemma 3.3. *The complexity of Pattern_Selection_Algorithm is* $O(r_{max}^2 \times n^2 \times k)$*, where* n *is the number of supernodes and* r_{max} *is the maximum number of subnodes belonging to supernodes in overlap multigraph.* k *is the number of applications [2].*

4. EXPERIMENTS

In this section, we perform experiments which attempt to show the impact of pattern selection algorithm both on utilization of embedded patterns and performance.

The flow of experimental setup is shown in Figure 7. The data flow graph for each application is generated. We have used the SUIF compiler to extract the data flow graphs of applications [4]. Pattern "MA" means a multiplier DFG node followed by an adder DFG node. In Table 1, the number of occurrences of each pattern in each application is reported in the first column. This data is provided by profiling through DFG nodes of applications as described in [4]. In order to obtain the number of instances of each pattern required by each application, we ran a simple and fast ASAP scheduling to get the data dependency between the DFG nodes. The number of instances used by ASAP scheduler gives the maximum number of patterns in each application. The second column in Table 1 shows the demand for each pattern in each application.

The gain function used in our experiments is the difference of area-delay product between the FPGA implementation of each pattern and ASIC implementation of each pattern. The area and delay corresponding to FPGA implementation of each pattern is obtained by Xilinx CoreGen IP blocks. Basically we used Xilinx XC4000XL as the target reconfigurable architecture in our experiments. Depending on ASIC core for each embedded pattern candidate, the gain can vary significantly. ASIC cores are synthesized with Synopsys Synthesis tool. The delay is reported in terms of number of clock cycles. We assume the speed clock of 80 MHz which is applicable to XC4000XL series. The area is reported in terms of number of CLBs in XC4000XL. Choosing the best ASIC and FPGA implementation for each pattern is an important research required in programmable system design. Pattern candidates are given as input to pattern selection tool. According to the area limit for embedded pattern on the target system, a set of patterns are selected.

After the embedded patterns are selected by the pattern selection algorithm, each application is scheduled on the target architecture. The target architecture consists of embedded patterns and reconfigurable units (here XC4000XL architecture is used). We have used a set of programs from the MediaBench application suite [10] in our experiments. Multimedia applications are suitable for reconfigurable systems due to their high level of parallelism. We have modified scheduling tool in [9] to accept the embedded cores. We set the reconfigurable area large enough so that no loop blocks are rejected for implementation on system. In this scheduling, the selected loops are implemented on reconfigurable system independently and there is no reuse of resources between any loops.

The performance and utilization of the embedded patterns in the system are reported. Utilization is the average number of embedded patterns used by scheduler while mapping the applications on the system.

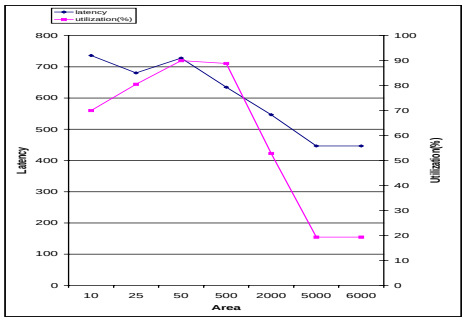

Figure 8: Utilization(%) and Latency vs. Area Limit for Embedded Patterns.

We ran the pattern selection algorithm with different area limits for embedded patterns. Table 2 shows the corresponding results. If we further increase the area limit above 5000, the selected patterns do not change. The plot in Figure 8 shows the average latency and utilization of embedded patterns while the area for embedded patterns on system changes. Initially the area constraint is strict and not necessarily best patterns can be chosen. After we increase the area limit to some extent, where area is large enough to fit several embedded patterns, the utilization and performance improve. In our method, we do not consider high cost for low utilization. That is we do not consider cost if a pattern is not used by an application; hence only gain is not added. Therefore, with no area limit, more number of resources are selected by algorithm until no gain can be obtained. Hence, for large area limit, utilization decreases. It should be noted that the scheduling algorithm is not optimized for combined fixed and programmable cores in the system. Therefore, hard-wired embedded patterns are not efficiently used in scheduling to improve

Benchmark C File	Number of Occurrence of Patterns						Number of Resources Demanded per Pattern					
	A	M	AA	AM	MM	MA	A	M	AA	AM	MM	MA
adpcm.c	29	6	0	0	0	6	26	6	0	0	0	6
convolve.c	239	82	0	2	0	74	193	82	0	2	0	74
fft.c	209	171	6	5	4	144	148	54	5	2	4	68
getblk.c	141	76	10	2	4	68	120	70	10	2	4	68
jctrans.c	214	35	23	1	0	33	180	35	23	1	0	31
jdmerge.c	229	52	37	2	1	48	168	50	29	2	1	46
motion.c	97	70	28	0	0	70	56	70	27	0	0	34
noise_est.c	315	154	12	0	3	144	239	143	12	0	3	135

Table 1: Occurrence and Demanded Number of Embedded Resources for Pattern Candidates in MediaBench.

the latency. In spite of this fact, we still observe reasonably good improvement in performance when number of embedded patterns increases. In order to support our idea that in reconfigurable system design more sophisticated methods are required to exploit the regularity among the applications, we performed another set of experiment. We selected only the common pattern candidates among the applications as embedded patterns in the system (intersection method). We compared the latency and utilization of embedded patterns after scheduling obtained from both methods. For instance, we experimented both methods for MediaBench applications when area limit for embedded cores is 500. Patterns selected by intersection method are 26 "A", 6 "MA", and 6 "M". Pattern Selection algorithm chooses 12 resources of pattern "MA". When the pattern selection algorithm is used, latency and utilization are improved by 11% and 25% respectively in average over all the MediaBench applications.

Area Limit for Embedded Patterns	Patterns
10	1 A, 1 M
25	1 A, 3 AA
50	1 A, 1 MA
100	2 AA, 2 MA
200	5 MA
500	12 MA
1000	25 MA
2000	48 MA, 1 MM
6000	135 MA, 4 MM

Table 2: Patterns vs. Area limit for Patterns resulted by Pattern Selection Algorithm for MediaBench.

5. CONCLUSION AND FUTURE WORK

In this work, we specifically introduce pattern selection problem for programmable platform-based systems. We propose an algorithm to select the best set of patterns demanded by different applications to be embedded on target system. We propose a general gain model which can accept different characteristics of each pattern such as delay, area and power. The experimental results show that our method chooses different set of patterns when the area limit for embedded core on system changes. We believe for future design systems we need more sophisticated methodologies for embedded module selection in order to be able to get the most out of such platform-based systems.

6. REFERENCES

[1] P. Schaumont, I. Verbauwhede, K. Keutzer, and M. Sarrafzadeh, "A Quick Safari Through the Reconfiguration Jungle", In the *38th Conference on Design Automation Conference* , June 2001.

[2] E. Bozorgzadeh, S. Ogrenci-Memik, R. Kastner, and M. Sarrafzadeh, "Pattern Selection for Programmable Systems", *Technical Report, Computer Science Department, UCLA*, Nov. 2001.

[3] S. Ogrenci-Memik, E. Bozorgzadeh, R. Kastner, and M. Sarrafzadeh, "SPS: A Strategically Programmable System", Proc. of *Reconfigurable Architecture Workshop*, April 2001.

[4] R. Kastner, E. Bozorgzadeh, S. Ogrenci-Memik, and M. Sarrafzadeh, "Instruction Generation for Hybrid Reconfigurable Systems", in the Proc. of *International Conference on Computer-Aided Design (ICCAD)*, Nov. 2001.

[5] W. Geurts, F. Catthoor, and H. De Man, "Time Constrained Allocation and Assignment Techniques for High Throughput Signal Processing", Proc. of *29th ACM/IEEE Design Automation Conference*, 1992.

[6] M. R. Garey, and D. S. Johanson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", *W. H. Freeman and Company*, 1999.

[7] T. J. Challahan, P. Chong, A. Dehon, and J. Wawrzynek, "Fast Module Mapping and Placement for datapaths in FPGAs", In *Proc. of International Symp. of Field Programmable Gate Arrays*, 1998.

[8] S. Cadambi, S. C. Goldstein, "CPR:A Configuration Profiling Tool", In *IEEE Symposium on FPGAs for Custom Computing Machines*, 1999.

[9] K. Bazargan, M. Sarrafzadeh "Fast Scheduling and Placement Methods for C to Hardware/Software Compilation", In *SPIE International Symposium on Information Technologies*, Vol. 4212, November 2000.

[10] C. Lee, M. Potkonjak, W. H. Maggione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing MultiMedia and Communication Systems", *Technical Report, Computer Science Department, University of California, Los Angeles*, 2001.