Pattern Selection in Programmable Systems

Elaheh Bozorgzadeh Ryan Kastner Seda Ogrenci-Memik Majid Sarrafzadeh

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

ABSTRACT

The increase in complexity of integrated circuits results in the need to develop hardware platforms shared among a set of applications in the same domain. Today's general purpose processors cannot satisfy the future aggressive timing and power constraints for a specific application. On the other hand, conventional ASIC design methodologies are costly and require a long time-to-market for today's complex designs. We need a platform based system optimized for a set of applications in a same domain. Reconfiguration has to be integrated into system design. We must exploit the regularity (or similarity) among applications in target system design. This regularity depends on the domain of applications . For example, Each application can demand different set of modules to be embedded as fixed cores in the target system.

In this work, we specifically study one of the important issues in domain-specific programmable design methodologies. We introduce the pattern selection problem. 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 an algorithm to select a set of patterns such that the objective is maximized. Our model and proposed algorithm can be applied at different levels of design hierarchy. Our method also considers the overlap between the patterns. 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 [6] 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.

1. INTRODUCTION

The increase in the complexity of integrated circuits and shorter time-to-market requirement results in the need to develop hardware platforms shared across multiple applications [1, 2, 3]. 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, 2]. On the other hand, the current general purpose fully programmable solutions cannot satisfy the future aggressive timing and power constraints. Therefore, new design methodology has to be developed to combine reconfiguration in system design for future applications. 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. A platform-based system is a generic system designed for a specific domain. In order to implement an application in a 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, 3]. 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) [4, 5]. In this methodology, there is a combination of IP cores, programmable logic cores, and memory blocks on a chip (see Figure 1). Several companies already provide new design methodologies [4] integrating programmability.

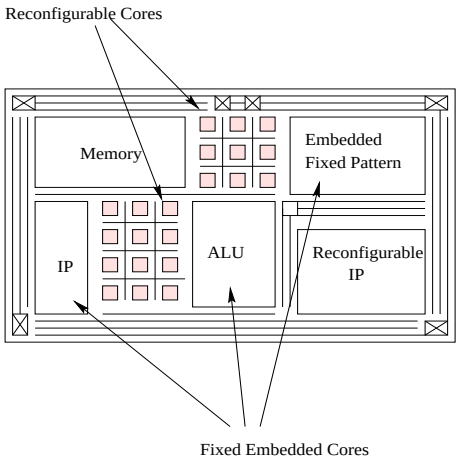

Figure 1: Integration of Programmable and Hard-wired Cores in System Design in Micro-architecture Level

System design can be viewed in a variety of levels of granularity from the architecture level to logic/interconnection level (Please refer to [2] for more details). Reconfiguration can also be applied in different hierarchy levels of a design [1]. For instance, programmability in gate level of system is realized via a programmable module, e.g. FPGA by programming the functionality of logic blocks

and connectivity of routing switches. In instruction-set architectural level, the instruction set can include custom-designed instructions other than a set of general instructions, such as the specific instructions (e.g. MAC) in DSP domain. Either an operation can be realized by some generic instruction separately or group of them can be realized by a customized instruction. Micro-architectural level of design defines the hardware implementation of components that is used to implement a function [1, 2]. An example is shown in Figure 1. The micro-architecture of a system can be fixed or partially fixed. It is important to consider programmability in different level of design. Considering reconfiguration in higher level removes (or reduces) the small variation of regularity among the applications, which can be observed in implementation (lower) level. Therefore, in a higher level, the designer would not be misguided by those variations when extracting regularity among the applications.

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. Applications are all given from a specific application domain. Exploiting regularity in applications leads to decrease in cost of programmability and lower total cost and development time.

FPGAs are programmable hardwares. FPGAs can provide the programmability and flexibility required for future embedded systems. Although a design implemented on FPGA cannot be as fast or as dense as the design implemented on an ASIC, the FPGA can be programmed by software tools to implement different hardware systems. On the other hand, original FPGAs cannot provide the requirement for high volume and complex applications. 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. Fine grain FPGA architectures are shifting towards new architectures where 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 [6, 7]. The basic building blocks of SPS architecture are parameterized functional blocks (called *VPBs*) that are pre-placed in a fully programmable logic array. Since VPBs are custom made and fixed, they do not require configuration, hence lesser number of switches are required to program VPBs. 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 [6] is that both fine grain and soft grain reconfiguration are required. The target architecture is optimized for applications in a given specific domain. Coarse grain reconfigurable blocks are most likely specialized for the applications. 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 our 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 [8, 9]. An architecture-driven high level synthesis of DSP applications, called Cathedral III, is presented. Similar to SPS project, the application specific units are extracted and synthesized from data flow graph of the given application instead of selecting dedicated data-paths from predefined libraries. In this work, reconfigurability is not considered and it is assumed all the operations are being implemented on Application specific units (ASUs). In addition, the target architecture is optimized for a single application. Their optimization problem cannot easily handle multiple applications. The main focus in Cathedral III architecture design methodology is on generating and merging operation clusters and assignment to ASUs in order to minimize the area. Any of clusters on the data flow graph are being implemented on a dedicated functional unit. The goal is that whether two different clusters should be implemented each by a dedicated data-path or by one partially-programmable functional unit capable of implementing the two clusters. That is where similarity or regularity of clusters comes into the picture.

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. Since we focus on reconfigurable systems, we assume that if a module candidate is not selected to be embedded as a fixed block, the corresponding function in each application can be realized by the reconfigurable units of the system. 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.

Our method can be applied in different level of abstractions as long as the constraints of our model is satisfied. In instruction set level, application specific instructions can be demanded by each application. The set of instructions for target architecture can be defined by our method. In pattern selection problem, the patterns would be the application specific instructions.

The paper proceeds as follows: In the following section, the pattern selection problem is formulated. In Section 3, we present 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. PATTERN SELECTION PROBLEM

In a reconfigurable hybrid structure, each operation in a data flow graph of an application may be realized by reconfigurable logic units, ALU, or a customized functional unit. In a hybrid structure, there are some embedded IP cores on system. The main focus in this paper is to choose a set of demanded modules to be embedded on the chip. We want to have a maximum covering and make the most benefits from custom designed blocks (or IP Blocks) and reconfigurable modules. Each module is able to cover a part of system. We assume if the demanded module is rejected to be embedded on the system, the corresponding operations would be mapped onto reconfigurable cores of the system. 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. It is not cost effective to have such costly customized modules that either are not used in many applications or do not yield a significant gain in overall compared to implementation on other components of system like reconfigurable units. We refer to the application specific units other than reconfigurable blocks as *pattern candidates* [1]. In the SPS architecture, pattern candidates are VPB blocks. In SoC systems, where there are different components such as IPs, ALUs and different computational units, we can have a different set of pattern candidates 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.

A good way to extract such pattern candidates is profiling. Profiling is mostly applied to DAG representation of systems. For example, we can easily extract control data flow graphs (CDFG) from a compiler. Pattern Generation via profiling have been done in several previous works. In [14], sub-graph matching is applied to extract the patterns in each application, which are critical in performance. In [13], datapath synthesis is based on pattern extraction from data-path of an application. DFG nodes are mapped into simple and compound components on a FPGA chip. In hardware-platform design, each application demands different functionality-based constraints on the hardware platform [1]. In [1], 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. The intersection set may only have the highest probability to be chosen as the set of embedded components in the system. In our approach we try to choose a set of patterns to be embedded while trying to meet maximum requirement of applications.

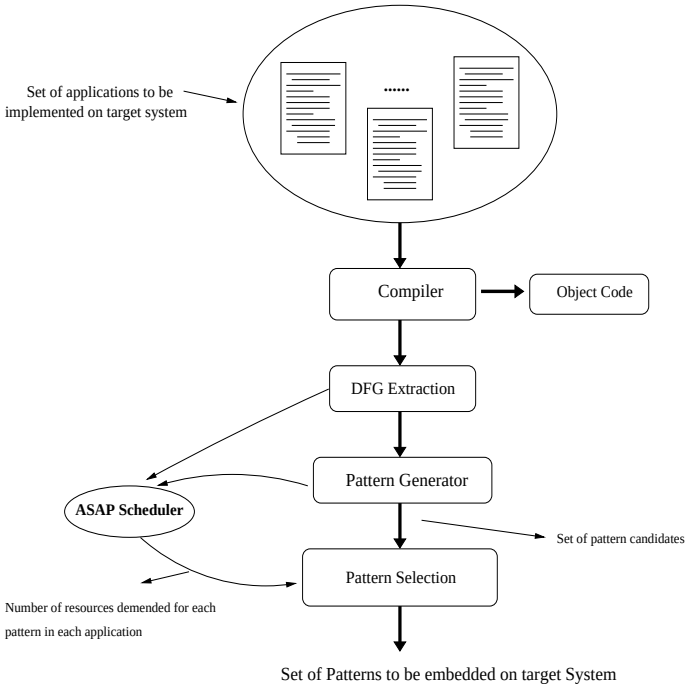

Figure 2: Pattern Selection in SPS Design Flow.

Figure 2 shows SPS design flow for choosing a set of patterns to be embedded on the system. The input to the design flow is a set of applications belonging to a specific domain. Data flow graphs of applications are extracted using a compiler tool such as SUIF [17, 18]. Then a pattern generator profiles different patterns demanded by different applications (See [7] 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 this work, we are focusing mostly on selection of computational units. We are assuming that given patterns are customized for the set of computations. They outperform the reconfiguration units. However, they are more costly due to inflexibility and low utilization compared to programmable units. The pattern candidates can overlap with each other, which means that they may cover the same nodes of data flow graphs.

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

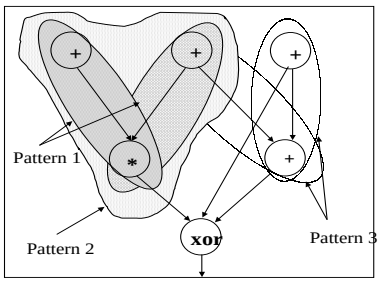

Figure 3: Different Embedded Pattern Candidates on a Data Flow Graph.

Figure 3 shows profiling on a data flow graph (DFG). In Figure 3, different extracted modules (or patterns) on the data flow graphs are shown. Pattern 1 has been observed two times in Figure 3. 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. We cannot afford having a fully utilized system for a set of applications. In the definition for pattern selection problem, we assume that patterns are already generated from the data flow graph of applications. Each application suggests a set of patterns.

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. Other characteristic objectives such as minimizing power consumption can be added to the gain function.

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. On the other hand, having only one resource for each pattern may underestimate

the demand for the pattern. We assume that the approximate demand for each pattern candidate is given in each application. In Figure 2, the number of demanded patterns is given to pattern selection tool by ASAP Scheduler. ASAP scheduler schedules the data flow graphs and 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. If all the pattern candidates are embedded as fixed blocks and there are many overlaps between candidates, maximum gain and utilization still are not likely to be achieved.

The problem is similar to resource allocation on scheduled data flow graph in high level synthesis [10]. 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.

2.1 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. The total gain would not be only the summation of both gains. We define the total gain as follows:

$$g_{total}(i,j) = g_i + g_j - overlap_{unit}(i,j), \tag{1}$$

$$0 \leq overlap_{unit}(i,j) \leq max(g_i, g_j). \tag{2}$$

A possible function representing overlap can be:

$$overlap_{unit}(i,j) = \frac{g_i + g_j}{2}. \tag{3}$$

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 embedded resources for pattern i and j as many as demanded, the total gain would be:

$$g_{total}(i,j) = occ_i \times g_i + occ_j \times g_j - c_{ij} \times overlap_{unit}(i,j). \tag{4}$$

If there are lesser number of embedded resources for pattern i and j, the overlap and gain can be approximately computed as follow respectively:

$$Overlap(i,j) = \frac{r_i}{d_i} \times \frac{r_j}{d_j} \times c_{ij} \times overlap_{unit}(i,j), \tag{5}$$

$$gain_i = \frac{r_i}{d_i} \times occ_i \times g_i, \tag{6}$$

$$g_{total}(i,j) = gain_i + gain_j - Overlap(i,j), \tag{7}$$

$$0 \leq r_i \leq d_i, \tag{8}$$

$$0 \leq r_j \leq d_j, \tag{9}$$

where r_i and r_j are the number of available resources of pattern i and j.

As mentioned before, the area assigned for embedded pattern modules are limited. Therefore, the total areas of selected patterns should not violate the area constraint. Recall that there are more than one application demanding different patterns for our target system. Equation 7 can be extended for all applications by summing the gain function over all applications as shown in Equation 10.

$$Gain = \sum_{k=0}^{app}(\sum_{i=0}^{n} f(i,k) \times occ_{i,k} \times g_i - \frac{1}{2} \times \sum_{j=0, i \neq j}^{n} Overlap(i,j,k)). \tag{10}$$

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 10 shows their corresponding value in application k.

2.2 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 Set in application j, $j = 1, ..., k$: $O_j = \{o_1, o_2, ..., o_n\}$,
  - Demand Set in each application j, $j = 1, ..., k$: $D_j = \{d_{1j}, d_{2j}, ..., d_{nj}\}$.
- The objective is to choose $R = \{r_1, r_2, ...r_n\}$ such that the total gain function in Equation 10 is maximized.
- Subject to:

$$0 \leq r_i \leq max(d_{ij}), \quad j = 0, 1, ..., k, i = 0, 1, ..., n \tag{11}$$

$$\sum_{i=0}^{n} r_i \times a_i \leq A_{max}, \tag{12}$$

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.

2.3 Overlap Graph

Overlap Graph is an undirected weighted Graph $G = (V, E)$. Weight of edges can be negative. There is a label associated with each node v. 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 5. 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 6). The labels of the dummy nodes are zero. Figure 4 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, which corresponds to area constraint in pattern selection problem (see Figure 4). Lemma 2.1 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.

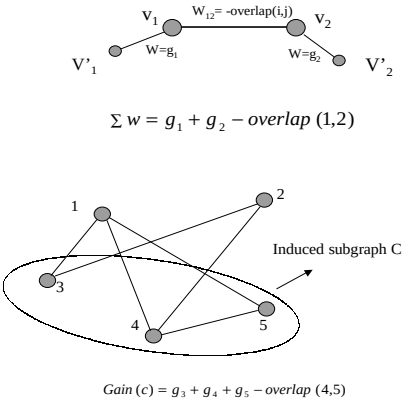

Figure 4: Overlap Graph.

Lemma 2.1. *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.*

Proof 2.1. *It can be easily proven by contradiction. If any of the two conditions mentioned in Lemma 2.1 is removed, the summation of weights of edges inside the subgraph is not the objective value for pattern selection problem.*

As mentioned before, there can exist more than one instance of pattern candidates demanded by applications. Therefore, we need to decide on the correct number of instances of each pattern candidate. We modify the graph in order to be able to handle the multiple instances of patterns and multiple application demands. . Figure 5 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 7, the weight of the edge is:

$$w_{i,j} = -Overlap(i, j) \tag{13}$$

$$w'_{i,j} = -\frac{1}{d_i} \times \frac{1}{d_j} \times c_{ij} \times overlap_{unit}(i, j) \tag{14}$$

$$w_{i,j} = r_i \times r_j \times w'_{i,j} \tag{15}$$

$$w_i^{dummy} = gain_i \tag{16}$$

$$w'_{d_i} = \frac{1}{d_i} \times occ_i \times g_i \tag{17}$$

$$w_i^{dummy} = r_i \times w'_{d_i} \tag{18}$$

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 5. 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. The gain of each subnode can be transformed into weight of subedges be adding dummy nodes as mentioned above. We add a dummy node to each subnode connected to subnode with a sub-edge. The weight of edge would be the unit gain of supernode. As shown in Equation 14 and 15, 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 17 and 18 similarly show that individual gain of each supernode can be expressed in terms of number of subnodes and unit gain (which is the weight of the edge connected to corresponding dummy subnode). The subgraph shown in Figure 5 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

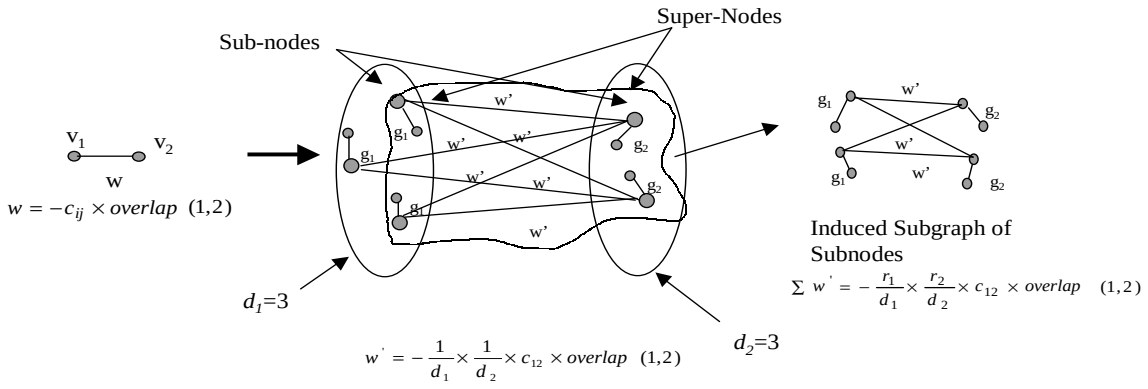

Figure 5: Supernodes and Subnodes in Overlap Graph

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.

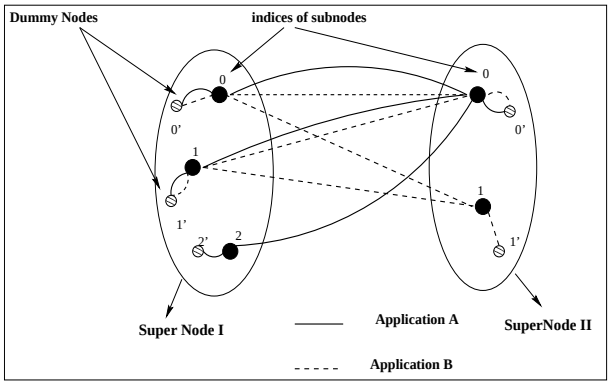

Figure 6: Overlap MultiGraph

As shown in Figure 6, 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 two instances of node II and two instances of supernode II . Since application B only demanded one resource for pattern II, the subedges belonging to application B will be connected only to one subnode 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.

3. PATTERN SELECTION ALGORITHM

In this section, we propose an algorithm for pattern selection problem formulated in Section 2. As described earlier, the problem is transformed into finding an induced subgraph in overlap graph such that the total weights of edges in the subgraph is maximized while the total summation of labels of nodes in subgraph do not exceed a given limit. Problem of finding an induced subgraph in a graph is equivalent to clustering a subset of nodes in a given graph while optimizing an objective function value corresponding to that cluster. In pattern selection problem, we generate a single cluster on overlap graph. The size of cluster is not fixed. It depends on label constraint (equivalent to area constraint) and maximizing the gain function.

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.*

Proof 3.1. *Assume that there is an overlap graph with n supernodes. Assume that there is only one instance of each supernode. Therefore, there is one subnode for each supernode. The gain of each supernode in pattern selection problem is* 1 *and the weight of edges between the supernodes are all $-n$. We can transform this graph into graph G' in linear time. In G', each node represents a supernode, label of each node is set to* 1 *and weight of edges are equal to $-n$. The problem in Theorem 3.1 on such a graph is a problem of finding an induced subgraph S such that the summation of edges and labels of nodes inside the cluster S is maximum. It can be easily seen that this problem is as hard as finding maximum independent set in undirected graph G'. Finding Maximum independent set in a graph G' is NP-Complete [12]. Therefore if subgraph S can be found in polynomial time, the problem of maximum independent set in G' can be solved in polynomial time as well and this contradicts the fact that the problem of MIS on G' is NP-complete.*□

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 2.1, 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. We will see later how our method resolve these two issues.

In sequential clustering, nodes are selected to be added to the cluster iteratively. We refer to the node being selected as *candidate* node. A Candidate node is not necessarily the acceptable choice. There are constraints which need to be satisfied by this candidate. We choose each candidate based on local closeness to optimality. Therefore, each candidate node is the one increasing the objective value the most among other candidates. However, it might not be feasible. In terms of feasibility, the node should satisfy the two constraints in Lemma 2.1. Otherwise the next best candidate is chosen.

Each time a node is being added to cluster, we need to express the candidate as a best choice. Therefore, a function called *potential* function is used to judge among the candidates. This judgment has to be towards optimizing the objective function, i.e. maximizing the total weight of edges inside the cluster.

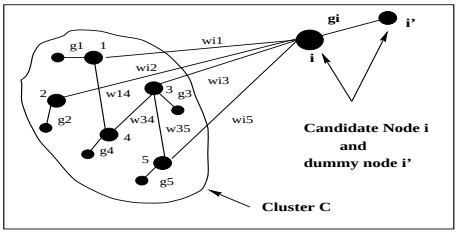

Figure 7: Candidate Node i Being Added to Cluster C.

Figure 7 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. Recall that weight of edges are negative. In addition, when a node is added to cluster its associated gain should also be included. No redundant dummy node should be inserted into the cluster.

We define potential function $p(i, C)$ for each candidate i as shown in Equation 19, which is related to the amount of change in objective value by adding the candidate to cluster C. 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.

$$p(i, C) = \frac{\sum_{j \in C} w_{ij} + g_i}{l_i} \tag{19}$$

Lemma 3.1. *A required condition for a candidate node to be 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.*

Proof 3.1. *If a node with negative potential is added to cluster, it decrease the gain of cluster itself. Never would the gain of that node become positive since the nodes already inside clusters are fixed.*□

A node which does not have positive potential gain, is not added to the cluster. In addition, 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 2.1.

Another important issue is that potential gain of each node i inside cluster C should not become negative after adding the candidate as shown in Equation 20.

$$g_i + \sum_{j \in C} w_{ij} > 0 \tag{20}$$

The pseudo-code of our proposed clustering method is shown in Figure 8. In the first step, potential function is computed for the first subnode (subnode with index 0) of each supernode. They are all inserted in a priority queue. When the first subnode with highest potential value is selected to be inserted into the cluster, there is no edge in computation of potential function. After that the potential value for neighbors of the recently selected subnode is updated. In addition, the potential value for the next index of that supernode is computed and inserted into the priority queue. At each time the best candidate is inserted inside the cluster if the constraints are satisfied. The constraints are area constraint and keeping the potential gain of each node inside the cluster positive. In addition, a candidate with non-positive potential cannot be added to cluster. This process continues until either there is no candidate with positive potential gain or the total labels summed over the subnodes inside the cluster has reached the limit. At each sequence of clustering, only one subnode (one index) of each supernode exists in the queue.

ALGORITHM 1. Pattern Selection Algorithm

input*: Overlap MultiGraph G*
output*: induced subgraph C*

$index = 0$
for *each supernode* $v_s \in G$

Compute_Potential_Gain(index)
Insert_Candidate_Queue(index)

While Priority_Queue_NonEmpty

candidate ¡- Queue_front
if $(Area\_Constraint(candidate)$ *and* $Potential\_Constraint(candidate))$
Insert_into_Cluster(C, candidate)
Update_Neighbors(candidate)
Update_SuperNode(candidate)
Increase_supernode_index(candidate)
Remove_from_Cluster(candidate,index)
Update_Queue

Figure 8: Algorithm *Pattern_Selection* Pseudo-code.

Lemma 3.2. *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.*

Proof 3.2. *In this algorithm each node is visited once. At each time a node is selected, the number of nodes needed to be updated is equal to edge degree of the node. Therefore the clustering algorithm*

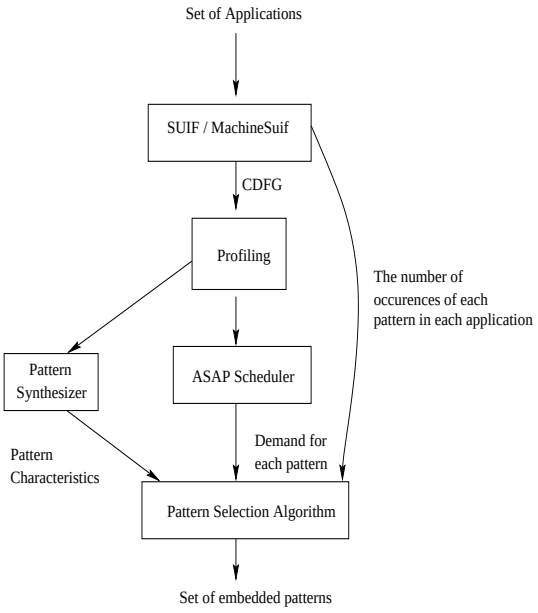

Figure 9: Experimental Flow.

process takes at most as long as visiting all the multi-edges on overlap multi-graph, i.e $O(E)$. The number of subnodes, N_s, is equal to $\sum_{i=0}^{n} r_i \times n_i$, where r_i is the number of subnodes for supernode n_i. Therefore the complexity of algorithm would be $O(N_s^2)$. It can be easily shown that $O(N_s^2) \leq O((r_{max}^2 \times n^2 \times k)$.□

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. First, we explain our experimental setup. Then we discuss the experimental results.

We focus on high level synthesis of reconfigurable system design. The flow of experimental setup is shown in Figure 9. The data flow graph for each application is generated. We have used the SUIF compiler [17] to extract the data flow graphs of applications [18]. The patterns shown in Figure 10 are extracted from DFGs. Pattern "AA" means two add operations followed after another in a DFG. Similarly, 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 [7]. 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 which can be used by each application. The number of resources used by ASAP scheduler for each pattern candidate is used as the demand for each pattern candidate in each application. The second column in Table 1 shows the demand for each pattern in each application. The pattern selection algorithm generates the overlap graph as shown in Figure 10. The edges in overlap graph shows the overlap between the pattern candidates.

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. We assumed that some ASIC core for each pattern with reasonably better delay and density is given. 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. The Pattern synthesizer engine in Figure 9 provides the characteristics of patterns for the pattern selection algorithm. 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.

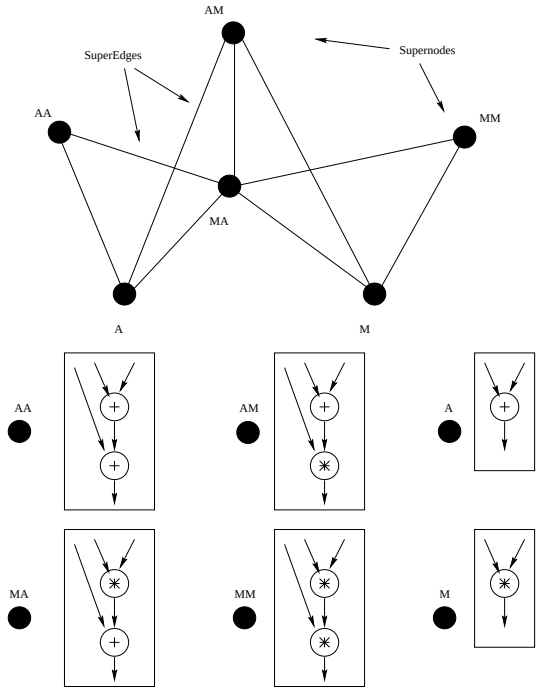

Figure 10: Overlap Graph of Pattern Candidates used in our Experiments.

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 [16] in our experiments. Multimedia applications are suitable for reconfigurable systems due to their high level of parallelism. We have modified scheduling tool in [15] 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.

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 11 shows the average latency and utilization of embedded patterns while the area for embedded pat-

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.

terns 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 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 studied one of the improtant issues in domain-specific programmable design methodologies. We specifically introduce pattern selection problem for such systems. Embedded cores

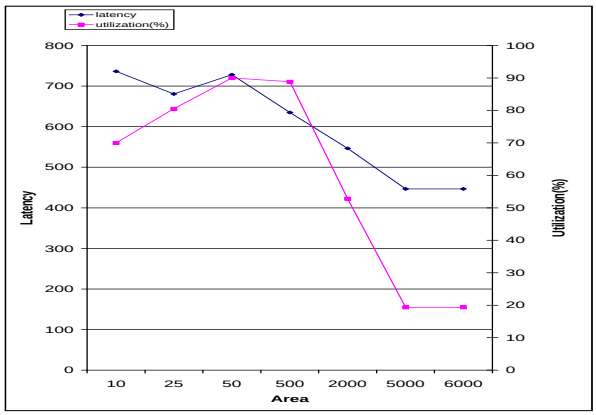

Figure 11: Utilization(%) and Latency vs. Area Limit for Embedded Patterns.

are selected by exploiting regularity among the applications in same domain. Therefore the number of patterns nominated by applications to be embedded on system can be large. There can exist overlap between the patterns. 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. We transform the pattern selection problem into clustering problem in a multi-graph. Our method also considers the overlap between the patterns. The experimental results show that our method chooses different set of patterns when the area limit for embedded core on system changes. Although our scheduler cannot efficiently exploit the embedded cores to improve the latency, we still outperform compared to the method in which only common patterns are chosen to be embedded. We believe for future design systems we need more sophisticated mothodologies for embedded module selection in order to be able to get the most out of such platform-based systems.

6. REFERENCES

[1] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System Level Design: Orthogonalization of Concerns and Platform-Based Design", In *IEEE trans on Computer-Aided Design*, Vol. 19, No. 12, December 2000.

[2] 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.

[3] H. Chang, et al., "Surviving the SOC Revolution: A Guide to Platform-Based Design", *Kluwer Academic Publishers*, 1999.

[4] A. Cataldo, "Actel and LSI Logic ignite drive to ASIC/PLD hybrids", *EE Times*,pg. 166, Feb. 19, 2001.

[5] S.J.E. Wilton, R. Saleh, "Programmable Logic Cores in SoC Design: Opportunities and Challenges", In *IEEE Custom Integrated Circuits Conference*, May 2001.

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

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

[8] T. Note, W. Geurts, F. Catthoor, and H. De Man, "Cathedral-III : Architecture-Driven High-level Synthesis for High Throughput DSP Applications", Proc. of *28th ACM/IEEE Design Automation Conference*, 1991.

[9] 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.

[10] G. D. Micheli, " Synthesis and Optimization of Digital Circuits", *McGraw-Hill*, 1994.

[11] D. S. Hochbaum, "Approximation algorithms for NP-Hard problems", *PWS Publishing Company*, 1995.

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

[13] 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.

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

[15] 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.

[16] 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.

[17] "SUIF 2 Compiler", *Stanford University*, available at: http://www.suif.stanford.edu/suif/suif2.

[18] G. Holloway, M. D. Smith, "The Machine SUIF SUIFvm Library", *Technical report, Harvard University*, 2000.