Efficient Algorithms for Finding Highly Acceptable Designs Based on Module-Utility Selections

Chantana Chantarapornchai  Edwin H.-M. Sha  Xiaobo (Sharon) Hu
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
email: {cchantra, esha, shu}@cse.nd.edu
Phone number: (219) 631-8803
Fax number: (219) 631-9260

Abstract

In this paper, we present an iterative framework to solve module selection problem under resource, latency, and power constraints. The framework associates a utility measure with each module. This measurement reflects the usefulness of the module for a given a design goal. Using modules with high utility values will result in superior designs. We propose a heuristic which iteratively perturbs module utility values until they lead to good module selections. Our experiments show that the module selections formed by combinations of modules with high utility values are superior solutions. Further, by keeping modules with high utility values, the module exploration space can drastically be reduced.

1 Introduction

Module selection in high-level synthesis is a complex problem due to its interaction with the scheduling and binding processes. Most existing high-level synthesis tools assume there is only one resource type of each functional unit during the scheduling phase. However, in some cases, functional units with the same functionality may use different modules to implement. This assumption can lead to an incomplete characterization of design space. Therefore, to ensure the fully design space exploration, module selection process needs to be integrated into the high-level synthesis tools.

In this paper, we present an efficient module selection framework which takes the effect of scheduling into account for a constrained environment. The success of the approach lies on the use of the module utility and inclusion scheduling. We called the approach utility selection. Moreover, we allow the constraint to be modeled as a fuzzy constraint, namely acceptability function, which takes arguments as design attributes. In return, the acceptability value describes how good the given design characteristics are. Such a proposed model can express a design objective, trade-off between conflicting design criteria, as well as hard (fixed) or soft (fuzzy) constraint boundary.

Many researchers have been studying scheduling, binding and module selection problems. Torby and Knight proposed a method which used a genetic algorithm to solve the scheduling and storage optimization problems [7]. However, their method excludes the fact that functional units with the same function may be implemented by different modules. Ahmad et.al. also used a genetic algorithm to solve the integrated problem of scheduling, binding, and module selection [1]. Their formulation, nonetheless, does not include fuzzy constraints. Ishikawa and De Micheli proposed a heuristic to find a module selection under latency and area constraints [4]. Their approach is neither applicable under fuzzy environment nor easily expandable to consider other criteria. Several works presented heuristics to completely characterize the design space [6, 3]. However, for a large application and module set, exploring all designs are still an expensive approach.

2 Models and Problem Descriptions

Operations and their dependencies in an application are modeled by a vertex-weighted directed acyclic graph, called a Data Flow Graph, $G = (V, E, \beta)$, where each vertex in the vertex set $V$ corresponds to an operation and $E$ is the set of edges representing data flow between two vertices. Function $\beta$ defines the type of operation for node $v \in V$. Operations in a data flow graph can be mapped to different functional units which in turn can be implemented by different modules. Such a system must also satisfy certain design constraints, for instance, power and cost limitation. These specifications are characterized by a tuple $S = (N, \mathcal{F}, M, A, Q)$, where $N$ is the number of functional units allowed in the system, $\mathcal{F} = \{f_i : \forall i \in [1,N]\}$ is the set of functional units allowed in the system, e.g., $\{\text{add, mul}\}$, $M = \{M_1, \forall f \in \mathcal{F}, \forall i \in [1,N]\}$, where each $M_1$ contains a set of eligible modules for functional unit $f_i$, e.g., $M_1 = \{\text{ripple_adder, carry-look-ahead_adder}\}$. A is a function mapping from $M_1 \in M$ to a set of tuples $\{a_1, \ldots, a_k\}$, where $a_1$ to $a_k$ represent attributes of a particular module. In this paper, we are interested in synthesizing a system under latency and power constraints. Hence, $A(m) = (a_1, a_2)$ where $a_1$ refers to the latency attribute of module $m$ while $a_2$ refers to the power consumption of module $m$. In this paper, we are interested in the average power consumption per time unit, using the formula in

---

*This work was supported in part the NSF under grant number MIP 95-01006, MIP-9612298 and MIP-9701416.
\[ P = \sum_{k} n_k p_k t_k \]  is the schedule length, \( p_k \) and \( t_k \) are the power consumption as well as execution time of functional unit \( f_k \), and \( n_k \) is the number of operations executed by functional unit \( f_k \). Finally, \( Q \) is a function that defines the degree of a system being acceptable for different system attributes. If \( Q(a_1, \ldots, a_k) = 0 \) the corresponding design is totally unacceptable while \( Q(a_1, \ldots, a_k) = 1 \), the corresponding design is definitely acceptable.

Using a function \( Q \) to define the acceptability of a system is a very powerful model. It can not only define certain constraints but also express design goals. In Figure 1(a), an example of z-curve shape fuzzy constraints is shown. The boundary is (latency \( t = 150 \), power \( p = 120 \)) while the objective is to optimize the weighted sum of latency and power \( 2t + p \) (tradeoff ratio 2:1).

Figure 1(b) presents the projection of the function in Figure 1(a) into a latency and acceptability plane. An inner z-curve (tighter latency) corresponds to a looser power constraint.

\[ \text{Figure 1. Example of of acceptability functions} \]

In this paper, the combined scheduling/binding and module selection problem we intend to solve can be formulated as follows: Given a specification containing \( S = (N, F, M, A, Q) \), and \( G = (V, E, \beta) \), select modules \( m \in M_f \) for a functional unit \( f \), \( V_f \), based on the resulting module utility values while maximizing the acceptability degree of the solutions executing graph \( G \).

\section{Utility Selection Framework}

We use the concept of utility values to compute the utility selection. Each module is associated with a utility value, which represents the usefulness of a module. The module may be present in good designs which optimize a certain goal and/or bad designs that do not satisfy the design goal. We allow the utility value of a module to be any real number between 0 and 1 to represent this ambiguity. Note that the design using those modules with utility value of 1 should be of the highest quality.

The operations of the framework at a high level is straightforward. First, a designer give initial utility values. Then, the framework improves them until they lead to good module selections. Figure 2 depicts the utility selection framework in details. It consists of two main steps, scheduling and utility value improvement. The first phase, inclusion scheduling, takes a data flow graph, the number of functional units required in a target system, and a module set as well as their associated utility values as inputs and constructs a general schedule. Rather than creating all possible schedules, inclusion scheduling creates only one schedule for the entire module combinations for efficiency. More importantly, it produces varying latencies and powers which are close approximation of the latencies and powers generated by the schedules of all module combinations [2]. Such informations are useful for future assessment of modules’ usefulness.

Particularly, while calculating a schedule, latency, power as well as the corresponding module usages are recorded. These data are then used as inputs to the second phase: the utility value improvement. In this step, the utility value of each module is adjusted. For a given module, the acceptability of every latency and power pair that the module contributes to is analyzed. Intuitively, if a module contributes to a lot of unacceptable latency and power values, the utility of the module should be decreased. On the other hand, if a module contributes to a lot of high-acceptability latency and power values, the module’s utility value should be increased. The statistics of a module usage for each latency and power value are used as a scaling factor to the acceptability value for the module for signifying the module’s contribution. Based on this idea, we have developed a heuristic to compute the relative adjustment of a utility value. The adjustment is then applied to update the previous utility value. The 2-step process is repeated until the adjustment values converge to zero. The experimental results show that the average number of iterations is approximately eleven.

\section{Inclusion Scheduling}

In order to construct an inclusion schedule based on a utility assignment, we borrow some techniques from the fuzzy theory [8]. In particular, we model modules and their respective utility values as a fuzzy set with respect to the corresponding functional unit. For functional unit \( f \), and its eligible module set \( M_f \), \( \mu(m) \in [0,1], \forall m \in M_f \), describes a utility value of module \( m \). This
The key information here is $\mathsf{freq}_{\alpha,p}(f,m)$, the number of module references for each latency and power value pair, which will lead to appropriate module utility assignments. This value is obtained by modifying inclusion scheduling to also tally the module contributing to each pair.

Then, we apply $\mathsf{adj}_{\alpha}(m)$ value in the following manner: if $\mathsf{adj}_{\alpha}(m)$ equals 1, we double the value of $\mu_{\alpha}(m)$ and if $\mathsf{adj}_{\alpha}(m)$ equals $-1$, $\mu_{\alpha}(m)$ is reduced by half. If $\mathsf{adj}_{\alpha}(m)$ is between $(-1,0]$, the change of $\mu_{\alpha}(m)$ is proportional to half of $\mu_{\alpha}(m)$ and if $\mathsf{adj}_{\alpha}(m)$ is between $[0,1]$, the change of $\mu_{\alpha}(m)$ is proportional to $\mu_{\alpha}(m)$. After the adjustment for all modules is made, $\mu_{\alpha}(m)$ are normalized with respect to the highest one, i.e., $\mu_{\alpha}(m) = \frac{\mu_{\alpha}(m)}{\max_{m \in M_{(\alpha)}} \mu_{\alpha}(m)}$, for all $m \in M_{(\alpha)}$. If $\mu_{\alpha}(m)$ is the same as $\mu_{\alpha}(m)$ from the previous iteration for every $m$, the adjustment is no longer needed.

### 4 Experimental Results

We performed several experiments on well-known benchmarks. Table 2 presents some of the results of the experiments from elliptic filter and discrete cosine transform. The module set used for these tests are shown in Figure 1.

In Table 2, Column “Spec” displays a specification of a target system. The linear acceptability functions used in these tests are described in Column “Acceptability”. Fields “latency” and “power” display two vital points $(x_L, x_U)$ in which any acceptability is in between $[0,1]$. Column “$w_1 : w_2$” displays a tradeoff ratio between latency and power of each respectively. Column “Selected modules” of Table 2 shows the results generated by our method.

<table>
<thead>
<tr>
<th>Modules</th>
<th>Time</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>5</td>
<td>60</td>
</tr>
<tr>
<td>a1</td>
<td>10</td>
<td>38</td>
</tr>
<tr>
<td>a2</td>
<td>20</td>
<td>23</td>
</tr>
<tr>
<td>a3</td>
<td>35</td>
<td>20</td>
</tr>
<tr>
<td>a4</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>a5</td>
<td>70</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modules</th>
<th>Time</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>m0</td>
<td>100</td>
<td>296</td>
</tr>
<tr>
<td>m1</td>
<td>160</td>
<td>84</td>
</tr>
<tr>
<td>m2</td>
<td>170</td>
<td>70</td>
</tr>
<tr>
<td>m3</td>
<td>300</td>
<td>55</td>
</tr>
<tr>
<td>m4</td>
<td>640</td>
<td>29</td>
</tr>
<tr>
<td>m5</td>
<td>770</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 1. Example module set

The following:

\[
\begin{align*}
\mathsf{adj}_{\alpha}(m) &= \sigma_+(f,m) - \sigma_-(f,m) \\
\sigma_{\pm}(f,m) &= \sum_{v \in \{1, \ldots, t\}} u_{\alpha}((t,v) \rightarrow (t,r_m)) \mathsf{acc}(1,1) \\
\mathsf{adj}_{\alpha}(m) &= \sigma_{\pm}(f,m) + \sigma_{\pm}(f,m) \\
\end{align*}
\]

Utility Adjustment Heuristic

Recall that the utility values of modules should reflect the usefulness of the modules towards a design goal. In our heuristic, we compute the positive contribution $\sigma_+(f,m)$ and negative contribution $\sigma_-(f,m)$ of module $m$ if it were to implement functional unit $f$, and use them to compute the relative adjustment value for each iteration. The adjustment value is computed using the following:

\[
\begin{align*}
\text{adj}_{\alpha}(m) &= \sigma_+(f,m) - \sigma_-(f,m) \\
\sigma_{\pm}(f,m) &= \sum_{v \in \{1, \ldots, t\}} u_{\alpha}((t,v) \rightarrow (t,r_m)) \mathsf{acc}(1,1) \\
\end{align*}
\]
and powers are between [418,498]. Both of our selections of Table 2, (a2,a2,a2,m0,m0) and (a2,a2,a1,m0,m0), resulting in the latency and power (980, 498) and (1010,481) respectively, also lies in this region.

Figure 3. Enumerated solutions for the dct experiment in the second last row in Table 2

Table 3. (a) # elite set and #enumerated set ratio (b) Average distribution per rank

<table>
<thead>
<tr>
<th>Ben.</th>
<th>Spec.</th>
<th>Sel-Enum</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>elf 1a 2m</td>
<td>1300,1700</td>
<td>75,150</td>
<td>5:1</td>
</tr>
<tr>
<td>elf 1a 2m</td>
<td>1300,1700</td>
<td>75,150</td>
<td>2a 2m</td>
</tr>
<tr>
<td>89</td>
<td>1a 2m</td>
<td>4:18</td>
<td>91%</td>
</tr>
<tr>
<td>elf 1a 2m</td>
<td>1300,1700</td>
<td>75,150</td>
<td>2a 2m</td>
</tr>
<tr>
<td>91</td>
<td>1a 2m</td>
<td>4.5:2085</td>
<td>99%</td>
</tr>
<tr>
<td>elf 1a 2m</td>
<td>5:216</td>
<td>97%</td>
<td></td>
</tr>
<tr>
<td>elf 1a 2m</td>
<td>5:216</td>
<td>97%</td>
<td></td>
</tr>
<tr>
<td>elf 1a 2m</td>
<td>1010,481</td>
<td>11:7776</td>
<td>99%</td>
</tr>
<tr>
<td>elf 1a 2m</td>
<td>1010,481</td>
<td>11:7776</td>
<td>7th</td>
</tr>
</tbody>
</table>

5 Conclusion

We have presented a module selection framework that takes into account of scheduling effect as well as resource, latency, and power constraints. This approach uses of the utility measure to model the degree of usefulness of a module. The scheduling and binding method called inclusion scheduling is used exclusively as a basis for deriving fuzzy latency and power values, approximating latency and power enumerated exhaustively, in order to improve module utility to reflect real module goodness. Experiments show that module utilities are good pointers to module selections. By using module utility, designers also have alternative in selecting initial designs. Our current approach can be integrated in an iterative design process varying the number of functional units for complete design exploration.

References