A Rough Programming Approach to Power-Balanced Instruction Scheduling for VLIW Digital Signal Processors

Shu Xiao and Edmund M-K. Lai

Abstract—The focus of this paper is on VLIW instruction scheduling that minimizes the variation of power consumed by the processor during the execution of a target program. We use rough set theory to characterize the imprecision inherent in the instruction-level power model that is obtained through empirical measurements. The optimal instruction scheduling problem based on such a power model is formulated as a chance-constrained rough program which is solved by a problem-specific genetic algorithm. Efficiency of the algorithm is greatly improved through a novel rule-based approach to rank the intermediate candidate schedules. Experimental results using the MediaBench and Trimaran benchmarks show that the near-optimal schedules obtained are significantly better than those obtained through the mixed-integer programming approach. Computational requirements are low enough for the technique to be adopted by practical compilers.

Index Terms—VLIW instruction scheduling, power-balanced scheduling, rough programming.

I. INTRODUCTION

Multimedia digital signal processing software typically exhibits a high degree of instruction-level parallelism (ILP) [1]. This parallelism can be efficiently exploited through appropriate instruction scheduling by the compiler for Very Long Instruction Word (VLIW) processors [2]. The number and type of parallel instructions being executed in a typical schedule vary substantially from clock cycle to clock cycle. As a result, there are potentially large processor current fluctuations. These fluctuations have a number of undesirable side effects. Firstly, they increase power supply voltage noise, which may result in timing and logic errors, especially for high performance processors [3], [4]. Secondly, large current variation is usually correlated with a large number of high peak currents which adversely affect the chip temperature [5] to which chip reliability and sub-threshold leakage power are exponentially related [6]. Besides, battery efficiency, for the same average discharge current, may change by as much as 25% as a result of the discharge current profile over time. In particular, the maximum battery life is achieved when the variance of the discharge current distribution is minimized [7]. It is therefore highly desirable to minimize supply power/current fluctuations. The problem of producing VLIW instruction schedules that minimize supply power fluctuation without degrading the speed performance of a program will be referred to as power-balanced VLIW instruction scheduling in this paper.

Conventional ways of controlling power variation between clock cycles are mainly by means of hardware mechanisms [4], [8]–[10]. More recently, the instruction scheduling approach to the problem of inductive noise reduction is gaining importance. In [11], [12] instructions are scheduled using heuristics. A more formal approach is used in [13]–[16] where the problem is formulated as a mixed-integer program (MIP) which is solved by standard techniques or a problem-specific branch-and-bound algorithm. The advantage of the MIP formulation is that the solutions obtained are guaranteed to be optimal based on the instruction-level power model of the VLIW processor used.

Instruction-level power models for VLIW processors are mostly constructed by making power measurements when certain instructions are being executed. The classical method involves empirical power measurements of the current drawn by the processor during repeated execution of each instruction as well as pairs of consecutive instructions [17]–[20]. It is perhaps the only practical way to derive a power model without having details of the microarchitecture design. For practical processors, however, the model constructed using the measurement method becomes too complex as there are too many possible instruction pairs to consider. Simplifications are proposed in [18], [21], trading off accuracy for reduction in model complexity. Furthermore, empirical measurements will always introduce some degree of imprecision or uncertainty. The standard practice is to use the mean or median values of the measurements in the power models. Thus the instruction schedules obtained using such power models can only achieve power balance in an average sense. It is therefore highly desirable to have knowledge of the level of confidence on the power performance of the instruction schedule in real situations.

Common methods for analysis and reasoning using models with uncertainty include Bayesian statistical techniques, Dempster-Schafer theory [22] and fuzzy sets [23]. In the case that the values of power model coefficients are obtained through empirical measurements, it is difficult to assign probability distribution functions or fuzzy membership functions for them because the precision of measurements is affected by nuisance factors. The best one can do is to estimate the intervals to which the values belong. This leads naturally to the use of rough set theory [24]–[26].

A framework of applying rough set theory to this problem
was first proposed by the authors in [27]–[29]. It consists of two main components. The first one is a rough power model to characterize the imprecision inherent in measurements. The rough power model consists of coefficients that are represented by rough sets defined by their lower/upper approximations. These approximations are derived purely from the power measurement data. However, the methodology of obtaining them was rather ad hoc. A theoretical description of the generic methodology of computing the rough set quantities from measurement data has not been provided. The second component is a chance-constrained rough programming formulation of the power-balanced instruction scheduling problem. The advantage of the resulting optimal schedule is that it is guaranteed to produce minimal power variation with the specified level of confidence. In order to solve this rough program, a novel rule-based technique was proposed for ranking candidate schedules. This technique, combined with a problem-specific genetic algorithm (GA), is able to efficiently produce near-optimal solutions to the rough program. However, the complete rough program is not provided. The papers only described a way to obtain the rough program based on the mixed integer program. Moreover, the solution techniques have only been tested on a few small programs.

This paper continues and extends our previous works [27]–[29] and addresses the shortcomings discussed above. In Section II, some basic concepts of rough set theory are explained according to their logical relations while being applied for computing rough set quantities from measurement data. Then using these concepts the generic way model coefficient uncertainty is characterized using rough sets is described in Section III. In Section IV, the complete rough program is provided so that it can be readily used in practical compilers. The theoretical basis of the rule-based technique for ranking candidate schedules is now fully established through rigorous mathematical analysis as presented in Section V. Through this technique, the problem-specific GA presented in Section VI can obtain near-optimal solutions very efficiently. Extensive experimental evaluation of this solution technique conducted using the extensive MediaBench [30] and Trimaran benchmarks [31] with the C6711 [32] as the target VLIW processor. Results presented in Section VII show that the schedules obtained exhibit substantially guaranteed reduction in power variation. Furthermore, the computational requirement of the rule-based GA is low enough for our technique to potentially be incorporated into production compilers.

II. A BRIEF INTRODUCTION TO RELEVANT CONCEPTS

Rough set theory has been developed to deal with imprecise concepts. We shall briefly introduce the ideas of approximation space, discretization, imprecise concepts, ordering of rough values and rough simulation which are relevant to our application. A more comprehensive introduction to rough set theory can be found in [24]–[26].

A. Approximation Space

Given a universe \( U \) of objects, each possessing a set of attributes \( S \), two objects \( x, y \in U \) are said to have an indiscernible relation \( I(S) \) if

\[
s(x) = s(y), \quad \forall s \in S
\]

where \( s(x) \) and \( s(y) \) denote the values of attribute \( s \) for objects \( x \) and \( y \) respectively. \( I(S) \) is an equivalence relation that partitions \( U \) into equivalence classes. The set of objects in an equivalence class is called an elementary set. The collection of all the elementary sets constitutes an approximation space. Thus an approximation space \( A \) is defined by its universe and an equivalence relation.

\[
A = (U, I(S))
\]

B. Discretization

The attributes that define an approximation space can either be continuous or discrete. If attribute values \( s(x) \) are continuous and real, it is quite possible that there will be few objects in \( U \) which have an indiscernible relation with each other according to (1). Consequently the number of equivalence classes and hence the number of elementary sets will be very large with each elementary set having very few elements. One way to overcome this problem is to note that the concepts are imprecise and thus we can partition the domain of \( s(x) \) into a finite number of intervals. Let the domain of \( s(x) \) be the interval \([l, r] \in \mathbb{R}\). A set \( \{b_1, b_2, \ldots, b_m\} \) partition \([l, r]\) into subintervals \([b_0, b_1), [b_1, b_2), \ldots, [b_m, b_{m+1}]\), where \( b_0 = l \) and \( b_{m+1} = r \). After this discretization, \( s(x) \) can be replaced by the subinterval it belongs to. The real-valued attribute \( s \) is now replaced by a discrete one \( s' \). The new indiscernibility relation \( I(S') \) between objects in \( U \) can then be used to partition \( U \) into the equivalence classes, resulting in a new approximation space

\[
A' = (U, I(S'))
\]

Finding the optimal subintervals for a given domain is known to be NP-hard. Some algorithms that has been proposed for this discretization process can be found in [33]–[35]. Sub-optimal solutions can be obtained by using the entropy/MDL and the Boolean reasoning algorithms with the latter one being the more popular choice. It translates the original problem into one that computes the minimal prime implicants of a Boolean function [34].

C. Imprecise Concepts

Any subset \( Y \subseteq U \) is called a concept. If \( Y \) is not exactly a union of a few elementary sets in an approximation space defined by either (2) or (3), \( Y \) is called an imprecise concept in this approximation space. Rough set theory defines \( Y \) through two approximations – the lower approximation \( I_*(Y) \) which is the greatest union of the elementary sets contained in \( Y \) and the upper approximation \( I^*(Y) \) which is the least union of the elementary sets containing \( Y \). Mathematically,

\[
I_*(Y) = \bigcup \{y \in U \mid I(y) \subseteq Y\}
\]

\[
I^*(Y) = \bigcup \{y \in U \mid I(y) \cap Y \neq \emptyset\}
\]

where \( I(y) \) denotes an elementary set containing the object \( y \). Thus all the objects in the lower approximation are certain
valid for describing the concept Y. But those objects in the upper approximation are only possibly valid descriptions of Y. Through these two approximations, the vagueness of a concept is characterized.

In this paper, we shall refer to the imprecise concepts represented through these two approximations as rough values.

1) Rough Values in an Approximation Space with Only One Real-Valued Attribute: In a real-valued one-attribute approximation space, the upper and lower approximations of an imprecise concept can simply be represented as two closed intervals on the domain of attribute. Let y denote any subinterval of the discretized attribute s. Given a concept Y, the lower and upper approximations are given by

\[ [a, b] = \bigcup \{ y \mid y \subseteq Y \} \]  
\[ [c, d] = \bigcup \{ y \mid y \cap Y \neq \emptyset \} \]  

The values within [a, b] are valid descriptions of Y with certainty and the values in [c, d] are possibly valid descriptions of Y with c ≤ a ≤ b ≤ d. Rough sets in such a one-attribute space can therefore be expressed as ([a, b], [c, d]).

Basic arithmetic of rough values in the form ([a, b], [c, d]) can be found in [36].

D. Ordering Rough Values

Unlike integers or real numbers, there are more than one way to rank rough values in the form ([a, b], [c, d]). They may be ranked using their α-optimistic values, the α-pessimistic values or the expected values [36]. In this paper, we choose the α-pessimistic value because it will give us a more conservative estimation.

**Definition 1:** Let \( \xi = (I_*(Y), I^*(Y)) \) be a rough value on the approximation space defined by (2). Then the lower trust measure of the rough value \( \xi \leq r \) is defined by

\[
Tr_\xi \{ \xi \leq r \} = \frac{\text{Card}(g \in I_*(Y) \mid y \leq r)}{\text{Card}(I_*(Y))}
\]

where Card() denotes the cardinal number of a given set. Similarly the upper trust measure is defined by

\[
Tr^\xi \{ \xi \leq r \} = \frac{\text{Card}(g \in I^*(Y) \mid y \leq r)}{\text{Card}(I^*(Y))}
\]

The trust measure of the rough event is given by

\[
Tr \{ \xi \leq r \} = \frac{1}{2} (Tr_\xi \{ \xi \leq r \} + Tr^\xi \{ \xi \leq r \})
\]

The trust may be defined as any convex combination of the lower and the upper trusts.

**Definition 2:** For the rough value \( \zeta = ([a, b], [c, d]) \), the trust measure of the rough event \( \zeta \leq r \) according to Definition 1 is defined by

\[
Tr(\zeta \leq r) = \begin{cases} 
0, & r < c \\
\frac{1}{2} \left( \frac{r - c}{r - a} \right), & c \leq r \leq a \\
\frac{1}{2} \left( \frac{r - c}{r - b} + \frac{r - d}{r - c} \right), & a \leq r \leq b \\
\frac{1}{2} \left( \frac{r - d}{r - a} + 1 \right), & b \leq r \leq d \\
1, & r \geq d
\end{cases}
\]  

**Definition 3:** Given \( \zeta = ([a, b], [c, d]) \) with \( c \leq a \leq b \leq d \), and \( \alpha \in (0, 1) \). Then

\[
\zeta^\alpha_{\inf} = \inf \{ \tau | Tr \{ \zeta \leq r \} \geq \alpha \}
\]

is called the \( \alpha \)-pessimistic value of \( \zeta \).

The \( \alpha \)-pessimistic value tells us that \( \zeta \) is less than \( \zeta^\alpha_{\inf} \) with a confidence level of \( \alpha \in (0, 1) \). If two rough values \( \zeta \) and \( \eta \) are to be ranked based on their \( \alpha \)-pessimistic values, then \( \zeta > \eta \) if and only if \( \zeta^\alpha_{\inf} > \eta^\alpha_{\inf} \) with a confidence level \( \alpha \).

Combining (8) and (9), we have

\[
\zeta^\alpha_{\inf} = \begin{cases} 
(1 - 2\alpha)c + 2ad, & \text{if } 0 < \alpha \leq \frac{a - c}{2(d-c)} \\
2(1 - \alpha)c + (2\alpha - 1)d, & \text{if } \frac{a - c}{2(d-c)} < \alpha \leq 1 \\
\frac{a(b - a) + a(d - c) + 2a(b - a)(d - c)}{(b - a) + (d - c)} - \frac{a - c}{2(d - c)}, & \text{if } \frac{a(b - a) + a(d - c) + 2a(b - a)(d - c)}{(b - a) + (d - c)} - \frac{a - c}{2(d - c)} < \alpha < \frac{b + d - 2c}{2(d - c)}
\end{cases}
\]

E. Rough Simulation

The \( \alpha \)-pessimistic value of a function involving rough values can be estimated by rough simulation [36]. This process is best illustrated by an example.

**Example 1:** Given three rough values \( \zeta_1 = ([0, 1], [-1, 3]), \zeta_2 = ([1, 2], [0, 3]) \text{ and } \zeta_3 = ([2, 3], [1, 5]) \). Define a function \( g = \zeta_1 + \zeta_2 + \zeta_3 \). Suppose we need to calculate \( g(\zeta_1, \zeta_2, \zeta_3)_{0.995} = \inf (\sqrt{v} | Tr(\zeta_1 + \zeta_2 + \zeta_3 \leq v) \geq 0.995) \).

To estimate \( g(\zeta_1, \zeta_2, \zeta_3)_{0.995} \) by rough simulation, random samples \( \lambda^0_i, \lambda^2_i \) and \( \lambda^3_i \) for \( i = 1, 2, ..., N \) are generated from the intervals \([-1, 3], [0, 3] \text{ and } [1, 5]\) respectively. Let \( N \) denote the number of times \( \lambda_1^0 + \lambda_2^0 + \lambda_3^0 \leq v \). Then the upper trust \( Tr(\zeta_1 + \zeta_2 + \zeta_3 \leq v) \) is given by \( N(v)/N \).

Similarly, the lower trust is obtained from random samples \( \lambda_1^0, \lambda_2^0 \) and \( \lambda_3^0 \) generated from the intervals \([0, 1], [1, 2] \text{ and } [2, 3]\) respectively. If \( N \) denote the number of times \( \lambda_1^0 + \lambda_2^0 + \lambda_3^0 \leq v \), then the lower trust \( Tr^\star(\zeta_1 + \zeta_2 + \zeta_3 \leq v) \) is given by \( N(v)/N \).

Finally, the trust \( Tr(\zeta_1 + \zeta_2 + \zeta_3 \leq v) \) is obtained by averaging, i.e. \( (N(v) + N(v))/2N \). The value of \( g(\zeta_1, \zeta_2, \zeta_3)_{0.995} \) is given by the minimum value of \( v \) such that

\[
\frac{N(v) + N(v)}{2N} \geq 0.95
\]

Using \( N = 2000 \) we obtained \( g(\zeta_1, \zeta_2, \zeta_3)_{0.995} = 39.9997 \).

III. INSTRUCTION-LEVEL POWER MODELS FOR VLIW PROCESSORS

There are two instruction-level power models in the literature that are specifically applicable to VLIW architectures [18], [20]. We shall brieﬂy describe the power model proposed in [18]. Then we shall show how the imprecision of the model coefficients can be characterized using rough values. The resulting rough power model will subsequently be used for power-balanced instruction scheduling. This methodology of characterizing imprecision is also applicable to the model proposed in [20].
A. Conventional Power Model Estimation

The organization of a generic VLIW processor core is shown in Figure 1. It has a number of functional units that can execute individual instructions in parallel. Let $K$ denote the number of functional units in the target VLIW processor. Suppose an instruction schedule has a duration of $t$ time slots. The total power consumed by the execution of this schedule can be expressed as

$$ P = \sum_{i=1}^{t} \left( \sum_{k=1}^{K} v_{k}^{i} + m_i \cdot p_i \cdot S \right) $$

where $v_{k}^{i}$ is the power consumed in time slot $i$ by the $k$-th functional unit, $m_i$ is the average number of additional cycles (stall cycles) occurred during the execution of the $i$-th long instruction word, $p_i$ is the probability that this pipeline stall occurs, and $S$ is the power consumption of the processor modules while the pipeline stalls. So $v_{k}^{i}$ is the sum of the base power cost of the instruction executed in time slot $i$ and the cost of inter-instruction effects due to previous instructions for functional unit $k$.

1) Imprecision Due to Reducing Model Complexity: The values of the coefficients $v_{k}^{i}$ and $S$ are typically determined through current measurements since the current drawn by the CPU during the execution of a program is proportional to its power cost [17], [19]. Stable current measurements for each individual instruction are made while it is executed repeatedly for a period of time. In order to account for inter-instruction effects, the measurements have to be repeated for all possible instruction pairs. Unfortunately, the number of instruction pairs that has to be considered is prohibitively large. This is because ideally all combinations of instructions that differ in terms of functionality (i.e., opcode), addressing mode (immediate, register, indirect, etc.) and data differences (either in terms of register names or immediate values) have to be considered. Even if we can exhaustively model the inter-instruction effects, the complexity of the model will be very high.

The complexity of model can be reduced by ignoring the inter-instruction effects and by clustering instructions with roughly the same power cost. Clustering can be performed by the $C$-means algorithm [21] which minimizes the mean-squared error. The power model can be simplified by having instructions in the same cluster share the same power coefficient. Let $c_k$ be the power consumption value of the $k$-th cluster. The model given by (11) can be simplified to

$$ P = \sum_{i=1}^{t} \left( \sum_{k=1}^{C} c_k r_{k}^{i} + m_i \cdot p_i \cdot S \right) $$

where $r_{k}^{i}$ is the number of cluster $k$ instructions being executed in time slot $i$ and $C$ is the total number of clusters.

2) Imprecision Due to Physical Measurements: The only way to enhance the accuracy of (12) is to reduce the size of each cluster of instructions, thus increasing the complexity of the model. Even if clustering is not employed, the imprecision inherent in current measurements will give rise to imprecision in the estimated coefficient values. The power coefficients are usually obtained by averaging a large number of current measurements. When such power models are applied to our instruction scheduling problem, the solutions obtained are optimal only in the average sense. We have no idea by how much would the power deviate from the average when the program is actually executed.

If minimal power variation has to be guaranteed with a specified level of confidence even when a simplified power model is used, we need a way to capture the imprecision of the power model.

B. A Rough Power Model

The rough power model we propose in this paper is similar to the conventional model (12) except the imprecise model coefficients $c_k$ are rough-valued. Since power consumption is the only attribute of interest in our application, we have a one-attribute approximation space. We shall denote this real-valued attribute by $w$.

The rough power model is constructed as follows:

**Step 1:** For each imprecise coefficient $c_k$, data of attribute $w$ are collected through current measurements.

- Measurements are made with every instruction instances taken from each instruction cluster.
- Measurements for an instruction instance are repeated a number of times to encapsulate the imprecision inherent in physical measurements.
- A randomized sequence is necessary to prevent the effects of unknown nuisance variables from contaminating the results.

Each measurement is treated as an object and all of them constitute the universe $U$.

**Step 2:** Since $w$ is real-valued, it has to be discretized, resulting in a discrete attribute $w'$.

**Step 3:** In the resulting new approximation space $(U, I(w'))$, the lower and upper approximations for each coefficient $c_k$ are computed according to (6) and (7).
TABLE I
APPROXIMATION SPACE OF THE POWER MODEL IN EXAMPLE 2.

<table>
<thead>
<tr>
<th>(w^3)</th>
<th>coefficient</th>
<th>(w^2)</th>
<th>coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>[190, 191)</td>
<td>(c_4)</td>
<td>[210, 213)</td>
<td>(c_3)</td>
</tr>
<tr>
<td>[191, 192)</td>
<td>(c_4)</td>
<td>[213, 215)</td>
<td>(c_2)</td>
</tr>
<tr>
<td>[192, 193)</td>
<td>(c_4)</td>
<td>[215, 216)</td>
<td>(c_1)</td>
</tr>
<tr>
<td>[193, 204)</td>
<td>(c_3)</td>
<td>[216, 217)</td>
<td>(c_1)</td>
</tr>
<tr>
<td>[204, 205)</td>
<td>(c_3)</td>
<td>[216, 217)</td>
<td>(c_2)</td>
</tr>
<tr>
<td>[204, 205)</td>
<td>(c_4)</td>
<td>[217, 218)</td>
<td>(c_1)</td>
</tr>
<tr>
<td>[205, 207)</td>
<td>(c_2)</td>
<td>[217, 218)</td>
<td>(c_2)</td>
</tr>
<tr>
<td>[205, 207)</td>
<td>(c_3)</td>
<td>[218, 219)</td>
<td>(c_1)</td>
</tr>
<tr>
<td>[205, 207)</td>
<td>(c_4)</td>
<td>[218, 219)</td>
<td>(c_2)</td>
</tr>
<tr>
<td>[207, 208)</td>
<td>(c_2)</td>
<td>[219, 220)</td>
<td>(c_1)</td>
</tr>
<tr>
<td>[207, 208)</td>
<td>(c_3)</td>
<td>[219, 220)</td>
<td>(c_2)</td>
</tr>
<tr>
<td>[208, 210)</td>
<td>(c_2)</td>
<td>[220, 222)</td>
<td>(c_1)</td>
</tr>
<tr>
<td>[208, 210)</td>
<td>(c_3)</td>
<td>[222, 225)</td>
<td>(c_1)</td>
</tr>
<tr>
<td>[210, 211)</td>
<td>(c_3)</td>
<td>[222, 225)</td>
<td>(c_1)</td>
</tr>
<tr>
<td>[210, 211)</td>
<td>(c_3)</td>
<td>[222, 225)</td>
<td>(c_1)</td>
</tr>
<tr>
<td>[211, 213)</td>
<td>(c_2)</td>
<td>[227, 230]</td>
<td>(c_1)</td>
</tr>
</tbody>
</table>

TABLE II
LOWER AND UPPER APPROXIMATIONS FOR THE COEFFICIENTS IN EXAMPLE 2 (*10MA).

| \(c_1\) | \([22.0, 23.0], [21.5, 23.3]\) |
| \(c_2\) | \([21.0, 21.5], [20.5, 22.0]\) |
| \(c_3\) | \([19.0, 20.2], [19.0, 20.7]\) |

Note that this method is independent of the number of the instruction clusters. We shall construct a simple instruction-level power model with four coefficient for the C6711 VLIW digital signal processor [32] as an example.

Example 2: The instruction set of the C6711 is partitioned into four clusters only for illustrative purposes. These four clusters are, namely, the memory access cluster, the double-precision floating-point arithmetic function cluster, the single-precision floating-point and fixed-point arithmetic function cluster and the miscellaneous cluster. Hence this power model has four coefficients, denoted \(c_1, c_2, c_3, c_4\).

The current measurement data for every instruction instances from each cluster are discretized using the Boolean reasoning algorithm [34]. After removing the same objects in \(U\), we obtain the approximation space shown in Table I. The lower and upper approximations for each coefficient are computed according to (6) and (7) and the results are shown in Table II. These four rough values can now be used by the rough power model for estimation of power consumption.

IV. CHANCE-CONSTRAINED ROUGH PROGRAM FORMULATION

The power-balanced VLIW instruction scheduling problem has previously been formulated as a mixed-integer program (MIP) in [13], [14], [16] based on the conventional power models described in Section III-A. However, with the introduction of the rough power model, we can no longer use the MIP formulation. Instead, the problem is formulated as a chance-constrained rough program [36].

The chance-constrained rough program is similar to constrained mathematical programs except rough values are involved. It has an objective function to minimize and a set of constraints that need to be satisfied. The objective is to minimize power deviations by rescheduling the VLIW instruction. In this case, the objective function is given by a rough value according to the rough power model. The constraints include data dependency and resource constraints which are needed to ensure that the schedule produced will execute the program correctly. Furthermore, we also require that all execution deadlines must be met. In order to ensure that the execution performance of the target program is not compromised, we can set the execution deadline as the time slots available in the optimal schedule obtained by performance-oriented VLIW instruction scheduling algorithms.

The chance-constrained rough program is formally given by \(\text{RP1}\) below. Detailed discussions on the objective function, constraints and solution are given in subsequent sections. The notations used are listed in the Appendix.

\(\text{RP1}: \quad \min P(X, \xi)^{\alpha}_{\inf} \)

subject to

\[ X = \bigcup \{x^i\} \quad i = 1, \ldots, t; \quad j = 1, \ldots, n \]  

The objective is to minimize power variation for the duration of the VLIW program. Total power variation can be defined as the sum of power variance over all time slots. Mathematically, the total power deviation \(P(X, \xi)\) of the schedule \(X\) using the set of rough power model coefficients \(\xi\) is given by

\[ P(X, \xi) = \sum_{i=1}^{t} (P_i - M)^2 \]  

where the average power \(M\) over a total of \(t\) time slots is

\[ M = \frac{1}{t} \left( \sum_{i=1}^{t} P_i \right) \]  

\(P_i\) is the power consumption in time slot \(i\) which can be obtained using the rough power model derived from (12).

An instruction schedule \(X\) consists of all the instructions in the schedule \(x^i\) (constraint (13)) which are the binary decision variables (constraint (14)) in this rough program. These decision variables are related to the power model parameters \(r^i_k\) as follows. Since \(r_k^i\) is the number of the cluster \(k\) instructions
which are being executed in time slot $i$, it can be expressed as

$$r_k^i = \sum_{j=1}^{n} c_k x_j^i + \sum_{m=1}^{L-1} \sum_{j=1}^{n} \Lambda(L^j - m) c_k x_{i-m}^j, \quad L^j > 1$$  \hspace{1cm} (21)

The first term on the right-hand-side of (21) relates to the instructions that are scheduled to start execution in the current time slot. The second term are those instructions which started earlier but execution is still being completed.

Since the set of coefficients in $\xi$ are rough values, the function $P(X, \xi)$ also returns rough values. For our problem, the more conservative $\alpha$-pessimistic values described in Section II-D are used for ordering the rough values because we want to guarantee minimal power variations. Based on (9), we have

$$P(X, \xi)_{\alpha} = \inf \{ r | Tr \{ P(X, \xi) \leq r \} \geq \alpha \} \hspace{1cm} (22)$$

A rough program like $RP1$, where rough values are ranked using their $\alpha$-pessimistic values, is called a chance-constrained program.

Given a schedule $X$ and the set of model coefficients $\xi$, the value of the objective function (22) is obtained through rough simulation in a way similar as Example 1.

**B. Constraints**

The function $G$ in constraint (18) is generated through data flow dependency analysis of the instructions in the program. It also includes resource usage constraints such as those imposed by single and multi-cycle functional unit latency as well as shared resources such as registers, read/write ports and shared buses. These constraints do not involve any rough values and are therefore the same as those found in [2], [14], [16], [37].

The inequality (17) guarantees that all execution deadlines will be met by the schedule generated. In this constraint, $D^j$ is the number of delay slots for instruction $j$. Finally, (16) ensures that each instruction can only be issued once.

It should be emphasized that $RP1$ does not take into account data cache misses which could happen during the execution of a schedule. This is because cache misses are not predictable at compile time. If cache misses are to be considered, then additional information will need to be available for scheduling.

**C. Solution**

It is not easy to solve $RP1$ since its objective function is multimodal and the search space is irregular. Genetic algorithm (GA) [38] has previously been applied to find the solution to rough programs [36]. However, a generic GA does not possess problem specific knowledge and thus the time required to reach the optimal solution is long. Furthermore, in the search for the optimal solution, candidate schedules have to be ranked and ordered according to their objective function values. Since the objective function gives rough values, this ranking process requires rough simulations which are very computationally expensive.

### Table III

**SYMBOLIC POWER CONSUMPTION PROFILES TO ILLUSTRATE**

**Definition 4.**

In order to drastically reduce the time required to reach the optimal solution, two key aspects have to be tackled. First, the computational complexity of candidate schedule ranking has to be reduced. Secondly, the GA has to be enhanced with problem-specific knowledge to help focus the search. Our proposed solutions are described in the following two sections.

**V. RULE-BASED APPROACH TO RANKING SCHEDULES**

We propose a rule-based approach to rank candidate schedules produced by the GA according to their objective function values. Our approach is based on the observation that the difference between the objective function values of two schedules only depends on the difference between the two schedules themselves. We shall give a mathematical proof to our hypothesis. Then we shall show how a set of basic rules can be obtained offline to evaluate the qualitative relation between the two schedules as well as between their respective objective function values. Note that these rules will only need to be generated once for each rough power model. Thus the overhead is minimal.

**A. Theoretical Basis**

The power profile of an instruction schedule is obtained using the rough power model derive from (12). The minimal difference between power profiles of any two schedules is the ‘OneMove’ difference as defined below.

**Definition 4:** If the power consumption of two schedules $X_1$ and $X_2$ are exactly the same in every time slots except $i$ and $j$ as shown in Table III, where the rough value $c$ of a power model coefficient is moved from time slot $i$ to $j$, then we say that $X_2$ is ‘OneMove’ from $X_1$.

Our hypothesis is that for two schedules $X_1$ and $X_2$, the difference $P(X_2, \xi)_{\alpha} - P(X_1, \xi)_{\alpha}$ is dependent on the $\alpha$-pessimistic values of the rough values in time slots $i$ and $j$ only. If this is true, then the $\alpha$-pessimistic operator is linear. Lemma 5 proves this linearity property when $\alpha$ is large enough. Based on Lemma 5, Theorem 6 shows that the above hypothesis is true.

**Lemma 5:** Given two rough values $y = ([a_y, b_y], [c_y, d_y])$ and $z = ([a_z, b_z], [c_z, d_z])$, let $u = y + z$. Then

$$u_{\alpha} = y_{\alpha} + z_{\alpha} \hspace{1cm} (23)$$

provided $\alpha \in (0, 1]$ satisfies

$$\alpha \geq \max \left( \frac{b_y + d_y - 2c_y}{2(d_y - c_y)}, \frac{b_z + d_z - 2c_z}{2(d_z - c_z)} \right) \hspace{1cm} (24)$$

**Proof:** Since $u$ is the sum of $y$ and $z$, the lower and upper approximations of $u$ are computed by adding the values of the corresponding limits (see [36] for rough arithmetics).

$$u = ([a_y + a_z, b_y + b_z], [c_y + c_z, d_y + d_z])$$
Let \( u^\alpha_{\inf}, y^\alpha_{\inf} \) and \( z^\alpha_{\inf} \) be the \( \alpha \)-pessimistic values of \( u, y \) and \( z \) respectively. Based on (10), these values are given by

\[
v^\alpha_{\inf} = \begin{cases} 
(1 - 2\alpha)(c_y + c_z) + 2\alpha(d_y + d_z), \\
2(1 - \alpha)(c_y + c_z) + (2\alpha - 1)(d_y + d_z),
\end{cases}
\]

\( \text{if } 0 < \alpha \leq \frac{a_y + a_z - c_y - c_z}{2(d_y + d_z - c_y - c_z)} \)

\[
y^\alpha_{\inf} = \begin{cases} 
(1 - 2\alpha)c_y + 2\alpha d_y, \\
2(1 - \alpha)c_y + (2\alpha - 1)d_y,
\end{cases}
\]

\( \text{if } 0 < \alpha \leq \frac{a_y - c_y}{2(d_y - c_y)} \)

\[
z^\alpha_{\inf} = \begin{cases} 
(1 - 2\alpha)c_z + 2\alpha d_z, \\
2(1 - \alpha)c_z + (2\alpha - 1)d_z,
\end{cases}
\]

\( \text{if } 0 < \alpha \leq \frac{a_z - c_z}{2(d_z - c_z)} \)

Note that \( p_u < \min(p_y, p_z) \).

Hence when \( \alpha \geq \max(p_y, p_z) \), we have \( \alpha \geq p_u \). In this case, we have

\[u^\alpha_{\inf} = y^\alpha_{\inf} + z^\alpha_{\inf} \]

This completes the proof.

**Theorem 6:** Suppose schedule \( X_2 \) exhibits a ‘OneMove’ from schedule \( X_1 \) in time slots \( i \) and \( j \) as shown in Table III. The other rough values in time slots \( i \) and \( j \) are respectively denoted as \( A = a_1 + a_2 + \ldots + a_m \) and \( B = b_1 + b_2 + \ldots + b_m \). If \( \alpha \) satisfies the condition stated in (24), we have:

\[P(X_2, \xi)_{\inf}^\alpha - P(X_1, \xi)_{\inf}^\alpha = \sum_{g=1}^{m}(2b_g c)_{\inf}^\alpha - \sum_{g=1}^{n}(2a_g c)_{\inf}^\alpha \]  
(27)

**Proof:** The power variation defined by (19) for a given schedule can be expressed as

\[P(X_1, \xi) = \sum_{k=1, k \neq i, k \neq j}^{t} (P_k - M)_{\inf}^\alpha + (P_i - M)^2 + (P_j - M)^2 \]

\[P(X_2, \xi) = \sum_{k=1, k \neq i, k \neq j}^{t} (P_k - M)_{\inf}^\alpha + (P_i - M)^2 + (P_j - M)^2 \]

Hence according to Lemma 5, with \( \alpha \) large enough, we have

\[P(X_1, \xi)_{\inf}^\alpha = \left( \sum_{k=1, k \neq i, k \neq j}^{t} (P_k - M)_{\inf}^\alpha \right) + (P_i - M)^2 + (P_j - M)^2_{\inf}^\alpha \]

\[P(X_2, \xi)_{\inf}^\alpha = \left( \sum_{k=1, k \neq i, k \neq j}^{t} (P_k - M)_{\inf}^\alpha \right) + (A - M)^2 + (B - M)^2_{\inf}^\alpha \]

\[P(X_2, \xi)_{\inf}^\alpha - P(X_1, \xi)_{\inf}^\alpha = (A - M)^2 + (B - M)^2_{\inf}^\alpha - (A - M)^2 - (B - M)^2_{\inf}^\alpha \]

\[= (2Bc)_{\inf}^\alpha - (2Ac)_{\inf}^\alpha \]

\[= \sum_{g=1}^{m}(2b_g c)_{\inf}^\alpha - \sum_{g=1}^{n}(2a_g c)_{\inf}^\alpha \]

This completes the proof.

**Corollary 7:** Suppose the schedule \( X_2 \) exhibits \( K \) ‘OneMove’ from the schedule \( X_1 \) with \( K > 1 \). Then the difference between the objective function values of \( X_2 \) and \( X_1 \) is equal to the sum of the differences caused by each of the \( K \) ‘OneMove’, independent of the sequence of the \( K \) ‘OneMove’.

**Proof:** First consider \( K = 2 \). We construct an intermediate schedule \( X_3 \) which exhibits one ‘OneMove’ from \( X_1 \) and another ‘OneMove’ to \( X_2 \). Then

\[P(X_2, \xi)_{\inf}^\alpha - P(X_1, \xi)_{\inf}^\alpha = (A - M)^2 + (B - M)^2_{\inf}^\alpha \]

\[= (2Bc)_{\inf}^\alpha - (2Ac)_{\inf}^\alpha \]

\[= \sum_{g=1}^{m}(2b_g c)_{\inf}^\alpha - \sum_{g=1}^{n}(2a_g c)_{\inf}^\alpha \]

In the case that the two ‘OneMove’ from \( X_1 \) to \( X_2 \) are a rough value \( c \) is moved from time slot \( i \) to \( j \) and the \( d \) is moved from time slot \( j \) to \( k \), as illustrated Table IV. The other
rough values in time slot \( i, j \) and \( k \) are respectively denoted as \( A = a_1 + a_2 + \ldots + a_n, B = b_1 + b_2 + \ldots + b_m \) and \( E = e_1 + e_2 + \ldots + e_q \). There are two possible schedules for \( X_3 \) given as the \( X_3^1 \) and \( X_3^2 \) in Table IV. In the case that \( X_3 \) is the schedule as \( X_3^1 \), following the equation (28) we have

\[
P(X_2, \xi)^\alpha_{\text{inf}} - P(X_1, \xi)^\alpha_{\text{inf}} = \left[ \sum_{g=1}^{m} (2e_g)_{\text{inf}}^\alpha - \sum_{g=1}^{m} (2b_g)_{\text{inf}}^\alpha - (2a_{\xi})_{\text{inf}}^\alpha \right]
\]

\[
+ \left[ \sum_{g=1}^{m} (2b_g)_{\text{inf}}^\alpha - \sum_{g=1}^{m} (2a_{\xi})_{\text{inf}}^\alpha \right]
\]

(29)

In the case that \( X_3 \) is the schedule as \( X_3^2 \), following the equation (28) we have

\[
P(X_2, \xi)^\alpha_{\text{inf}} - P(X_1, \xi)^\alpha_{\text{inf}} = \left[ \sum_{g=1}^{m} (2b_g)_{\text{inf}}^\alpha - \sum_{g=1}^{m} (2a_{\xi})_{\text{inf}}^\alpha \right]
\]

\[
+ \left[ \sum_{g=1}^{m} (2e_g)_{\text{inf}}^\alpha - \sum_{g=1}^{m} (2b_g)_{\text{inf}}^\alpha \right]
\]

(30)

The result of (28) in (29) and (30) are the same. Thus this completes the proof for the case that the two ‘OneMove’ from \( X_1 \) to \( X_2 \) are as shown in Table IV. Other cases of two ‘OneMove’ can be proved similarly.

This completes the proof for \( K = 2 \). It can be extended in a similar way for \( K > 2 \).

Our optimization problem is to guarantee the minimal power variation with the confidence level \( \alpha \). Thus a large enough confidence level is generally needed in which case the \( \alpha \geq \max(p_y, p_z) \) required by Lemma 5 is generally true. In the rest of this paper, we assume that \( \alpha \) satisfies this condition.

B. Basic Rules to Evaluate Difference Between Objective Function Values of Any Two Schedules

Corollary 7 tells us that the difference between the objective function values of two schedules is only dependent on the \( \alpha \)-pessimistic values of the products of rough values in the time slots which exhibit ‘OneMove’ differences. Therefore if we pre-compute the \( \alpha \)-pessimistic values for all possible combinations of the rough model coefficients, the difference in objective function values between any two schedules can be obtained using these values without performing rough simulations. These values can be expressed as rules where the premise is a pair of rough model coefficients and the conclusion is the \( \alpha \)-pessimistic value of the product of this combination obtained by rough simulation.

Suppose there are a total of \( C \) coefficients in the rough power model. Then there will be \( \frac{1}{2}(C^2 + C) \) rules in total. This process of building the set of rules only needs to be performed once offline for the rough power model of a target VLIW processor. It is illustrated the following example.

Example 3: Consider the rough power model in Example 2. The premise of the rules are the combinations of any two of

\[
\|P(X_2) - P(X_1)\|_{\alpha} = \max_{X_2, X_1} \min_{\xi_{\text{inf}}} P(X_2, \xi)^\alpha_{\text{inf}} - P(X_1, \xi)^\alpha_{\text{inf}}
\]

\[
\leq \sum_{X_2, X_1} \sum_{\xi_{\text{inf}}} (2e_g)_{\text{inf}}^\alpha - \sum_{g} (2b_g)_{\text{inf}}^\alpha - \sum_{g} (2a_{\xi})_{\text{inf}}^\alpha
\]

\[
+ \sum_{g} (2b_g)_{\text{inf}}^\alpha - \sum_{g} (2a_{\xi})_{\text{inf}}^\alpha
\]

Thus the four coefficients \( c_1, c_2, c_3 \) and \( c_4 \). So there are a total of ten rules. Let \( \alpha = 0.95 \). We compute the \( \alpha \)-pessimistic values of the products of the combinations using rough simulation. Table V shows the resulting ten rules.

By comparing the ‘OneMove’ differences between two schedules, the difference between their objective function values can be obtained using these rules as shown in Example 4.

Example 4: There are seven instructions to be scheduled in five time slots and two feasible schedules \( X_1 \) and \( X_2 \) are shown in Table VI. Suppose the instructions \( s_1, s_2, s_3 \) and \( s_4 \) belong to cluster \( c_3 \) and the instructions \( s_4, s_5 \) and \( s_6 \) belong to cluster \( c_4 \). Suppose the instructions all take one time slot. Then the symbolic power consumption profiles of \( X_1 \) and \( X_2 \) are as shown in Table VI.

\[
X_2 \text{ exhibits a difference of two 'OneMove' from } X_1 \text{ – one is from time slot 2 to 3 and the other is from time slot 4 to 5. Thus according to Corollary 7 and Theorem 6, the difference between the objective function values of } X_2 \text{ and } X_1 \text{ is given by}
\]

\[
P(X_2, \xi)^\alpha_{\text{inf}} - P(X_1, \xi)^\alpha_{\text{inf}} = (2c_4c_1)^\alpha_{\text{inf}} - (2c_1c_4)^\alpha_{\text{inf}} + (2c_4c_1)^\alpha_{\text{inf}} - (2c_1c_4)^\alpha_{\text{inf}}
\]

The four \( \alpha \)-pessimistic values can be obtained in Table V. Thus

\[
P(X_2, \xi)^\alpha_{\text{inf}} - P(X_1, \xi)^\alpha_{\text{inf}} = 2 \times 945.87 - 1063.46 - 841.52 = -13.24
\]

VI. A GENETIC ALGORITHM USING RULE-BASED SCHEDULE RANKING

The chance-constrained rough program RP1 is solved using a GA. An instruction schedule is represented by a chromosome which is an array of integer variables each representing one instruction in the schedule. The execution time slot allocated to an instruction is indicated by the value assigned to its corresponding variable. An initial population of feasible candidate
schedules is created randomly. The size of the population will be denoted by \( \text{pop\_size} \).

The fitness of schedules in each generation is evaluated by the objective function (22). The difference between the objective function values of any two candidate schedules are obtained using the rule-based approach described in Section V. In each generation, candidate schedules are sorted in a non-decreasing order of their objective function values. The next generation of schedules are generated through the application of the genetic operations of selection, crossover and mutation until the optimal solution is found.

The rank-based roulette-wheel selection scheme [38] is adopted for selection. Suppose the candidate schedules of a generation have been sorted in non-decreasing order of fitness. The \( i \)-th schedule is assigned a probability of selection by the objective function (22). The difference between the objective function values of any two candidate schedules are obtained using the rule-based approach described in Section V.

In order to prevent premature convergence, each candidate schedule has a probability to be mutated as governed by the mutation rate. A mutation operator which randomly changes the allocated time slots of an instruction is applied. Similar to the crossover operation, all infeasible schedules are discarded.

When a pre-determined number of generations is reached, the algorithm terminates. The maximum number of generations depends on the size of the problem, i.e. the number of instructions and the number of available time slots. Finally, rough simulation has to be performed once to obtain the objective function value of the resulting best schedule because the rule-based approach only computes the differences between the objective function values during the search.

In order to prevent premature convergence, each candidate schedule has a probability to be mutated as governed by the mutation rate. A mutation operator which randomly changes the allocated time slots of an instruction is applied. Similar to the crossover operation, all infeasible schedules are discarded.

The rough-programming approach arrives in Table II, the rough programming method approach arrives at the following schedule:

\[
X_{\text{RIP}} = \{ x_1^1, x_4^2, x_3^3, x_5^4, x_6^5, x_7^6, x_8^7, x_9^8, x_{10}^9, x_{11}^{10}, x_{12}^{11}, x_{13}^{12}, x_{14}^{13} \}
\]

The population size is set to be 30 and the number of generations is also 30. A low crossover rate of 0.2 and a high mutation rate of 0.8 are used in order to prevent premature convergence. This is because the crossover operator often cannot create enough feasible offspring due to violations of the constraints. Unfortunately there is no formula available for determining the parameter values at this time.

The \( \alpha \)-pessimistic objective function values for the three schedules \( X, X_{\text{MIP}} \) and \( X_{\text{RP}} \) are computed by rough simulation using a confidence level of 0.95. They are given by

\[
P(X, \xi)^{0.95}_{\text{inf}} = 6.6808 \times 10^4
\]
\[ P(X_{MIP}, \xi)^{0.95}_{\text{conf}} = 5.5633 \times 10^4 \] \hspace{1cm} (32)
\[ P(X_{RP}, \xi)^{0.95}_{\text{conf}} = 5.4781 \times 10^4 \] \hspace{1cm} (33)

These results show that the schedule obtained by rough programming guarantees a smaller power variation compared than the performance optimal schedule and the optimal schedule obtained by integer programming.

Note that if the instructions are not clustered and power measurements are exact, then the power model will exhibit no imprecision. In this case, both the MIP and RP results will be the same. However, the rough programming approach will always produce superior results if uncertainties are inherent in the power model.

### B. More Extensive Results

More substantial results are obtained using problem instances from the Trimaran [31] and MedianBench [30] benchmarks. The C6711, a VLIW digital signal processor, is used as the target processor. The rough power model of TMS320C6711 and the rules for schedule ranking are those shown in Examples 2 and 3 respectively. All the computational experiments are performed on a 2.8GHz Intel Pentium 4 personal computer with 512MB RAM running under Microsoft Windows 2000.

The scheduling results for a confidence level of 95% are presented in Table VII. For each problem instance, the problem dimension (Dim.) is an ordered pair indicating the number of available time slots and the number of instructions to be scheduled. Columns ‘X’, ‘\(X_{MIP}\)’ and ‘\(X_{RP}\)’ show the values of (22) for schedules obtained through the performance-oriented compiler, the mixed-integer program method and the rough program method respectively.

The results show that the rough program approach always produce schedules with less power deviation compared with the MIP approach. The improvement ranges from 18.7% to 64.8% for the Trimaran benchmarks and from 4.5% to 53% for the MediaBench ones. The computation time required by the proposed problem-specific GA (Column ‘T’) show that it increases linearly with the number of instructions to be scheduled. The absolute times required even for the largest program schedules show that our technique is efficient enough for implementation in production compilers.

### VIII. Conclusions

A rough programming approach to the power-balanced VLIW instruction scheduling optimization problem has been proposed. Our work is the first attempt to take into account power model uncertainty so that the schedules obtained have guaranteed optimal power variations to a user specified degree of confidence. The uncertainty in the power model is handled by rough set techniques. The instruction scheduling problem based on the rough power model is formulated as a chance-constrained rough program. Finally, a problem-specific GA is developed to find solutions to the rough program. A computationally efficient rule-based approach is proposed to rank the candidate schedules without having to use rough simulation which is computationally expensive. Extensive results using the MediaBench and Trimaran benchmarks show that the quality of the schedules produced are superior to those obtained using mixed-integer programming. Furthermore, the computational requirement of the rule-based GA is low enough for it to be incorporated into practical compilers.

The GA proposed in Section VI can locate good solutions but cannot guarantee optimality. Thus one extension of current work is to develop algorithms that can find the optimal schedules of the rough program.

### Appendix

#### Notations

- \(n\) is the total number of instructions to reschedule.
- \(i\) is the total number of time slots available.
- \(x_i^j = 1\) if instruction \(j\) is allocated to time slot \(i\). Otherwise, \(x_i^j = 0\).
- \(X\) is the set of variables \(x_i^j\). Thus \(X\) specifies a schedule.
- \(C\) is the total number of clusters which the target instruction set is divided into.
- \(c_k\) is the power consumption value of the \(k\)-th cluster.
- \(\xi\) is the set of rough power model coefficients \(c_k\).
- \(P(X, \xi)\) is the power variation of the schedule \(X\) using the set of rough power model coefficients \(\xi\).
- \(M\) is the average power over all \(t\) time slots.
- \(P_t\) is the average power in the time slot \(i\).
- \(r_k^i\) is the number of the cluster \(k\) instructions being executed in time slot \(i\).
- \(c_k^j = 1\) if instruction \(j\) belongs to cluster \(k\) and zero otherwise.
- \(L\) is the number of execution pipeline steps for instruction \(j\).
- A function \(\Lambda(x)\) is defined by
  \[
  \Lambda(x) = \begin{cases} 
  1, & x \geq 1 \\
  0, & \text{otherwise}
  \end{cases}
  \]

#### Table VII

<table>
<thead>
<tr>
<th>Dim.</th>
<th>Trimaran (X_{MIP}) ((\times 10^4))</th>
<th>Trimaran (X_{RP}) ((\times 10^4))</th>
<th>% Improv.</th>
<th>T (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(20,15)</td>
<td>11.06</td>
<td>9.81</td>
<td>6.15</td>
<td>37.5</td>
</tr>
<tr>
<td>(28,30)</td>
<td>94.45</td>
<td>94.45</td>
<td>94.45</td>
<td>94.45</td>
</tr>
<tr>
<td>(55,34)</td>
<td>10.11</td>
<td>10.11</td>
<td>10.11</td>
<td>10.11</td>
</tr>
<tr>
<td>(30,27)</td>
<td>48.46</td>
<td>48.46</td>
<td>48.46</td>
<td>48.46</td>
</tr>
<tr>
<td>(55,58)</td>
<td>197.70</td>
<td>197.70</td>
<td>197.70</td>
<td>197.70</td>
</tr>
</tbody>
</table>

Note that if the instructions are not clustered and power measurements are exact, then the power model will exhibit no imprecision. In this case, both the MIP and RP results will be the same. However, the rough programming approach will always produce superior results if uncertainties are inherent in the power model.

These results show that the schedule obtained by rough programming guarantees a smaller power variation compared than the performance optimal schedule and the optimal schedule obtained by integer programming.

Note that if the instructions are not clustered and power measurements are exact, then the power model will exhibit no imprecision. In this case, both the MIP and RP results will be the same. However, the rough programming approach will always produce superior results if uncertainties are inherent in the power model.

The scheduling results for a confidence level of 95% are presented in Table VII. For each problem instance, the problem dimension (Dim.) is an ordered pair indicating the number of available time slots and the number of instructions to be scheduled. Columns ‘X’, ‘\(X_{MIP}\)’ and ‘\(X_{RP}\)’ show the values of (22) for schedules obtained through the performance-oriented compiler, the mixed-integer program method and the rough program method respectively.

The results show that the rough program approach always produce schedules with less power deviation compared with the MIP approach. The improvement ranges from 18.7% to 64.8% for the Trimaran benchmarks and from 4.5% to 53% for the MediaBench ones. The computation time required by the proposed problem-specific GA (Column ‘T’) show that it increases linearly with the number of instructions to be scheduled. The absolute times required even for the largest program schedules show that our technique is efficient enough for implementation in production compilers.

### VIII. Conclusions

A rough programming approach to the power-balanced VLIW instruction scheduling optimization problem has been proposed. Our work is the first attempt to take into account power model uncertainty so that the schedules obtained have guaranteed optimal power variations to a user specified degree of confidence. The uncertainty in the power model is handled by rough set techniques. The instruction scheduling problem based on the rough power model is formulated as a chance-constrained rough program. Finally, a problem-specific GA is developed to find solutions to the rough program. A computationally efficient rule-based approach is proposed to rank the candidate schedules without having to use rough simulation which is computationally expensive. Extensive results using the MediaBench and Trimaran benchmarks show that the quality of the schedules produced are superior to those obtained using mixed-integer programming. Furthermore, the computational requirement of the rule-based GA is low enough for it to be incorporated into practical compilers.

The GA proposed in Section VI can locate good solutions but cannot guarantee optimality. Thus one extension of current work is to develop algorithms that can find the optimal schedules of the rough program.

### Appendix

#### Notations

- \(n\) is the total number of instructions to reschedule.
- \(i\) is the total number of time slots available.
- \(x_i^j = 1\) if instruction \(j\) is allocated to time slot \(i\). Otherwise, \(x_i^j = 0\).
- \(X\) is the set of variables \(x_i^j\). Thus \(X\) specifies a schedule.
- \(C\) is the total number of clusters which the target instruction set is divided into.
- \(c_k\) is the power consumption value of the \(k\)-th cluster.
- \(\xi\) is the set of rough power model coefficients \(c_k\).
- \(P(X, \xi)\) is the power variation of the schedule \(X\) using the set of rough power model coefficients \(\xi\).
- \(M\) is the average power over all \(t\) time slots.
- \(P_t\) is the average power in the time slot \(i\).
- \(r_k^i\) is the number of the cluster \(k\) instructions being executed in time slot \(i\).
- \(c_k^j = 1\) if instruction \(j\) belongs to cluster \(k\) and zero otherwise.
- \(L\) is the number of execution pipeline steps for instruction \(j\).
- A function \(\Lambda(x)\) is defined by
  \[
  \Lambda(x) = \begin{cases} 
  1, & x \geq 1 \\
  0, & \text{otherwise}
  \end{cases}
  \]

#### Table VII

<table>
<thead>
<tr>
<th>Dim.</th>
<th>MediaBench (X_{MIP}) ((\times 10^4))</th>
<th>MediaBench (X_{RP}) ((\times 10^4))</th>
<th>% Improv.</th>
<th>T (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(55,34)</td>
<td>10.11</td>
<td>10.11</td>
<td>10.11</td>
<td>10.11</td>
</tr>
<tr>
<td>(30,27)</td>
<td>48.46</td>
<td>48.46</td>
<td>48.46</td>
<td>48.46</td>
</tr>
<tr>
<td>(55,58)</td>
<td>197.70</td>
<td>197.70</td>
<td>197.70</td>
<td>197.70</td>
</tr>
</tbody>
</table>
where $x$ is an integer.

- $D^j$ is the number of delay slots of instruction $j$.

### References


