

Solving Large Scale Assignment Problems in High-Level Synthesis by Approximate Quadratic Programming *

Florin Balasa
University of Illinois at Chicago
Dept. EECS, Chicago, IL 60607
fbalasa@eecs.uic.edu

Werner Geurts
Target Compiler Technologies
Leuven, B-3001, Belgium
geurts@retarget.com

Francky Catthoor Hugo De Man
Interuniv. Microelectronics Center
Katholieke Universiteit Leuven
{catthoor,deman}@imec.be

Abstract

Several assignment problems encountered in high-level synthesis, e.g., assigning operations from a signal flow graph to data-path units, can be modeled as binary quadratic optimization problems. However, these formal approaches are usually avoided, as the size of the resulting problems for real-life applications is discouragingly high relative to the capabilities of the existing optimization methods.

In this paper, an approximate binary quadratic technique will be presented. This technique has been successfully applied for solving assignment problems during data-path allocation and background memory allocation. The approach has several appealing characteristics: it can handle large examples with a better accuracy than heuristic techniques, it does not require any constraint concerning the convexity of the cost function, and it requires a low implementation effort as it can employ any available LP solver.

1 Introduction

Models based on integer linear programming (ILP) formulations have been used in high-level synthesis to solve scheduling problems, or simultaneous scheduling and hardware allocation, signal-to-register binding [6]. However, assignment problems encountered in distributed data processing or high-level synthesis can be better expressed as binary *quadratic* programming (QP) problems, as the quadratic objective functions allow a more refined modeling:

$$\begin{aligned} \text{QP :} \quad & \text{minimize} \quad \frac{1}{2} \mathbf{x}^T \mathbf{C} \mathbf{x} + \mathbf{c}^T \mathbf{x} \\ & \text{subject to} \quad \mathbf{A} \mathbf{x} \leq \mathbf{b} \quad , \quad x_j \in \{0, 1\} \quad \forall j \end{aligned}$$

where \mathbf{C} is a symmetric $n \times n$ matrix, \mathbf{A} is a $m \times n$ matrix, and \mathbf{c} and \mathbf{b} are n - and m - vectors, respectively. The notation \mathbf{x}^T means transposition of a column vector \mathbf{x} .

* *Proceedings of the 11th ACM Great Lakes Symp. on VLSI*, pp. 19-24, 2001.

However, these formal models are usually avoided in high-level synthesis because of the size of the resulting problems for real-life applications. The optimization methods existent in the literature (see Section 2 for a brief overview) only provide results for small examples (up to a few tens unknowns and a few tens constraints), relative to the actual high-level synthesis requirements. According to our own experience, while testing some of the existing techniques, the time computation grows very quickly with the size of the problem. The outcome is that quadratic models are usually considered impractical for high-level synthesis assignment problems, and are therefore disregarded. Furthermore, heuristic sequential approaches – in which one object is selected according to some rule, and afterwards assigned according to a positioning rule – are usually preferred to the (more costly) global approaches – yielding an assignment solution for all the objects simultaneously.

There are two main reasons leading to failure when modeling the assignment as a binary quadratic programming problem. First, the models employed are usually flattened, attempting to assign individual objects, like binary operators or scalar signals. Second, *exact* optimization techniques are employed for solving the practical problems of QP-type, which is usually too time-consuming or even impossible. As models are, naturally, approximations of real phenomena, an *approximate* solution of the practical QP problems should provide reasonably good assignment solutions.

In this paper, an approximate binary quadratic optimization technique is described. The method has been applied to assign clusters of operations from a signal flow graph to custom data paths, and to carry out the signal-to-memory assignment for multi-dimensional (array) signals.

Section 2 presents an overview of the major 0-1 quadratic optimization techniques. Section 3 describes our approximate approach based on alternate continuous relaxations and partial assignments. Section 4 introduces the quadratic models for two high-level synthesis problems: the assignment of operation clusters to execution units, and the as-

segment of multi-dimensional signals to background memories. A result overview is presented in Section 5, followed by conclusions of this work.

2 Overview of binary quadratic programming techniques

The quadratic zero-one programming gained attention in the domain of operations research because it has several important applications as distributed computer network scheduling and graph partitioning [3]. From the complexity point of view, the problem is NP-hard since it is related to the minimum cut and maximum clique problems [7].

A category of existing techniques is based on implicit enumeration. This class of approaches was initiated by Balas, who applied a branch-and-bound strategy – achieving in fact an investigation for all solution configurations [1]. The backtrack idea for an exhaustive search of the solution space has been further refined by Geoffrion, who proved its applicability for handling nonlinear objective functions [8]. Taha has proposed a Balasian-based algorithm for solving the zero-one programs with polynomial objective functions [13]. The basic idea is that a polynomial problem can be converted into an equivalent binary linear system with nonlinear secondary constraints – representing the polynomial terms. The solution space of the subsequent problem is investigated further with a similar implicit enumeration technique. Our experience while testing Geoffrion's and Taha's algorithms has shown that enumerative approaches are unlikely to solve problems with more than 30 - 40 variables, as the computation time is really blowing up.

Another category of existing techniques is based on the transformation of the problem QP into a 0-1 linear problem by introducing additional variables. This strategy is motivated by the fact that linear methods are easier to program and refine, and a more extensive theory does exist for the linear case. Furthermore, based on the fact that x^k can be replaced by x in the objective function and constraints (where x is an arbitrary variable of the problem, and $k \in N$), these methods can be easily adapted to solve the binary *polynomial* programming problems. These approaches differ from each other by the transformation rules, the goal being to achieve more economical representations of the polynomial programs. For instance, Glover and Woolsey proposed to replace the polynomial cross-product terms by *continuous* variables rather than *integer* variables [10], since the difficulty of (mixed-) integer programming problems depends more strongly on the number of integer variables rather than on the number of continuous ones. According to Glover's rules, the problem QP is transformed as follows:

- (1) replace each x_j^2 in the objective function by x_j ;
- (2) replace each cross-product $x_i x_j$ by a new nonnega-

tive continuous variable x_{ij} satisfying the new constraints:

$$x_i + x_j - x_{ij} \leq 1 \quad , \quad x_i \geq x_{ij} \quad , \quad x_j \geq x_{ij}$$

The conversion methods have recently gained more attention for practical reasons, as more and more efficient mixed ILP packages are becoming available. For instance, Glover's transformations and the mixed ILP solver LAMPS [14] have been employed by Geurts in order to assign operations from a signal flow graph to functional units [9].

The conversion methods depend on the sparsity of matrix \mathbf{C} : the number of cross-products drastically affects the number of variables and the number of constraints of the resulting equivalent linear problem. We have experimented on examples resulting in mixed ILP problems with over 1000 variables and 1000 constraints, solved by LAMPS [14] in CPU times of tens of seconds. However, assignment problems from realistic examples can yield much larger problems.

Other approaches require as a prerequisite the convexity¹ of the objective function [11].

The major drawback of these *exact* optimization techniques is the incapability to handle large binary quadratic programming problems. Therefore, an *approximate* but effective approach – able to deal with much larger problem sizes – is presented in the sequel. Moreover, unlike many of the existing methods, our novel technique does not require any constraint concerning the convexity of the cost function.

3 Approximate binary quadratic optimization

After modeling the given assignment problem as a binary quadratic optimization, a size evaluation of the equivalent ILP is carried out. The small/medium size problems are handled directly, employing the linearization technique of Glover and Wolsey [10]. When the problem is large (in terms of variables, cross-products, and constraints), a greedy process – based on continuous relaxations – is initiated. The general scheme of the proposed approach is the following:

- derive an equivalent objective function (if necessary)
- in order to ensure that matrix \mathbf{C} is positive definite ;
- while (*problem QP is too large*) {
 - solve relaxed QP : minimize $\frac{1}{2} \mathbf{x}^T \mathbf{C} \mathbf{x} + \mathbf{c}^T \mathbf{x}$
 - subject to $\mathbf{A} \mathbf{x} \leq \mathbf{b}$, $\mathbf{0} \leq \mathbf{x} \leq \mathbf{1}$
 - assign to all components of \mathbf{x}^{opt} satisfying $x_i^{opt} < \epsilon$
 - or $x_i^{opt} > 1 - \epsilon$ the values 0 and 1, respectively ;
 - derive a new QP with reduced dimensions, eliminating
 - from the old problem the variables with assigned values;
- }
- transform QP into a mixed ILP, employing Glover's rules
- solve the resulting 0-1 mixed ILP ;

The main aspects of this scheme will be discussed below.

¹That is, matrix \mathbf{C} must be positive semidefinite: $\mathbf{x}^T \mathbf{C} \mathbf{x} \geq 0$ for all vectors $\mathbf{x} \in \mathbb{R}^n$. If $\mathbf{x}^T \mathbf{C} \mathbf{x} \leq 0$ for all vectors $\mathbf{x} \in \mathbb{R}^n$, then \mathbf{C} is negative semidefinite, and the cost function is concave.

3.1 The positive definite transformation

When modeling an assignment problem as a binary QP, it is difficult to directly ensure the convexity of the cost function – an important property exploited by many optimization techniques, and in particular, by the continuous relaxation presented in subsection 3.2. Therefore, the initial cost function of the binary QP must be transformed (if necessary) into an equivalent one, which is strictly convex (matrix \mathbf{C} is positive definite, that is, $\mathbf{x}^T \mathbf{C} \mathbf{x} > 0$ for all nonzero $\mathbf{x} \in \mathbb{R}^n$).

It must be observed that $x_i^2 = x_i$ at every zero-one solution of QP. Therefore, each term in the objective function of the form $1/2 c_{ii} x_i^2$ can be replaced by the linear term $1/2 c_i x_i$. This simple property can be used to ensure that the new matrix \mathbf{C}^{new} is positive definite: it is sufficient to increase the diagonal coefficients of \mathbf{C} with some values Δc_{ii} selected large enough to ensure the diagonal dominance of the new matrix \mathbf{C} :

$$c_{ii}^{new} = c_{ii} + 2\Delta c_{ii} \quad \text{and} \quad c_i^{new} = c_i - \Delta c_{ii}$$

As pointed out by Carter [4], as the values Δc_{ii} are increased, the solution to the relaxed problem gets closer to the point $(\frac{1}{2}, \dots, \frac{1}{2})$, which is an undesirable effect. Consequently, the values Δc_{ii} must not be too large or too small. The transformation scheme proposed by Carter is employed in our algorithm. According to this scheme, the Δc_{ii} are chosen such that the spread between the largest and smallest eigenvalues is minimized. This has positive implications on both stability and on the performance of the programming algorithm since the resulting function contours tend to be circular in shape (the nearest integer to a continuous minimum is more likely to be optimal). For the sake of consistency, Carter's transformation based on a modified Choleski factorization $\mathbf{C} = \mathbf{L}\mathbf{L}^T$ [4] is reproduced below:

for $j = 1$ to n {
 compute $p_i = c_{ij} - \sum_{r=1}^{j-1} l_{jr} l_{ir}$, $i = j+1, \dots, n$
 let $l_{jj} = \left(\sum_{i=j+1}^n p_i^2 \right)^{\frac{1}{4}}$; $l_{ij} = p_i / l_{jj}$, $i = j+1, \dots, n$
 the increase of the diag. coefficient $\Delta c_{jj} = \sum_{i=1}^j l_{ji}^2 - c_{jj}$
 update $c_{jj}^{new} = c_{jj} + \Delta c_{jj}$, $c_j^{new} = c_j - \Delta c_{jj} / 2$
}

At this stage, \mathbf{C}^{new} is positive semidefinite with at least one zero eigenvalue. Finally, each diagonal element of \mathbf{C}^{new} is further increased with a positive constant α . In order to ensure a well conditioned problem, the recommended value of α (which will be the minimum eigenvalue of matrix \mathbf{C}^{new}) is (according to [4]): $\alpha = \sqrt{\sum_{i,j} c_{ij}^2} / (k-1)$, with $k = 500$.

3.2 The solution of the relaxed quadratic program

The continuous relaxed QP is solved with a cutting plane approach. Let $\hat{\mathbf{x}}$ be the solution of the unconstrained optimization: minimize $\frac{1}{2} \mathbf{x}^T \mathbf{C} \mathbf{x} + \mathbf{c}^T \mathbf{x}$. Then $\hat{\mathbf{x}} = -\mathbf{C}^{-1} \mathbf{c}$,

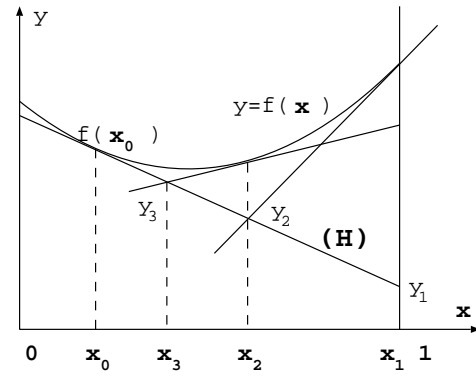


Figure 1: Geometrical interpretation for solving relaxed QP

and the corresponding value of the objective function is $\hat{y} = -\frac{1}{2} \mathbf{c}^T \mathbf{C}^{-1} \mathbf{c}$ (\mathbf{C} is positive definite, thus nonsingular).

If $\mathbf{A} \hat{\mathbf{x}} \leq \mathbf{b}$ and $\mathbf{0} \leq \hat{\mathbf{x}} \leq \mathbf{1}$, then $\hat{\mathbf{x}}$ is also the solution of the continuous relaxed quadratic programming problem. Otherwise, let $\mathbf{x}_0 = (x_1^0, \dots, x_n^0)$ be an arbitrary point in $[0, 1]^n$. As the tangent hyperplane in $(\mathbf{x}_0, f(\mathbf{x}_0))$ at the hypersurface $y = f(x_1, \dots, x_n)$, where $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is continuously differentiable, has the equation

$$y - f(\mathbf{x}_0) = \sum_i \frac{\partial f}{\partial x_i}(\mathbf{x}_0) (x_i - x_i^0)$$

the tangent hyperplane at $y = f(\mathbf{x}) \stackrel{def}{=} \frac{1}{2} \mathbf{x}^T \mathbf{C} \mathbf{x} + \mathbf{c}^T \mathbf{x} + \hat{y}$ in $(\mathbf{x}_0, f(\mathbf{x}_0))$ is

$$y - \frac{1}{2} \mathbf{x}_0^T \mathbf{C} \mathbf{x}_0 - \mathbf{c}^T \mathbf{x}_0 - \hat{y} = (\mathbf{x}_0^T \mathbf{C} + \mathbf{c}^T) (\mathbf{x} - \mathbf{x}_0)$$

The hypersurface $y = f(\mathbf{x}) \equiv \frac{1}{2} \mathbf{x}^T \mathbf{C} \mathbf{x} + \mathbf{c}^T \mathbf{x} + \hat{y}$ is contained (see Fig. 1) in the half-space (H)

$$y \geq \frac{1}{2} \mathbf{x}_0^T \mathbf{C} \mathbf{x}_0 + \mathbf{c}^T \mathbf{x}_0 + \hat{y} + (\mathbf{x}_0^T \mathbf{C} + \mathbf{c}^T) (\mathbf{x} - \mathbf{x}_0) \quad (1)$$

This inequality is equivalent to $\mathbf{x}^T \mathbf{C} \mathbf{x} - 2\mathbf{x}^T \mathbf{C} \mathbf{x}_0 + \mathbf{x}_0^T \mathbf{C} \mathbf{x}_0 \geq 0$. According to the Choleski factorization $\mathbf{C} = \mathbf{L}\mathbf{L}^T$, where \mathbf{L} is a lower-triangular matrix, it results that $(\mathbf{x}^T \mathbf{L} - \mathbf{x}_0^T \mathbf{L}) (\mathbf{x}^T \mathbf{L} - \mathbf{x}_0^T \mathbf{L})^T \geq 0$. This result was expected because of the convexity of $f(\mathbf{x})$. In conclusion, introducing the new constraint equivalent to the equation of the half-space (1)

$$(\mathbf{x}_0^T \mathbf{C} + \mathbf{c}^T) \mathbf{x} - (y - \hat{y}) \leq \frac{1}{2} \mathbf{x}_0^T \mathbf{C} \mathbf{x}_0 \quad (2)$$

should not affect the solution of the programming problem:

$$\begin{aligned} &\text{minimize} \quad y - \hat{y} \quad \text{where} \quad y - \hat{y} \equiv \frac{1}{2} \mathbf{x}^T \mathbf{C} \mathbf{x} + \mathbf{c}^T \mathbf{x} \\ &\text{subject to} \quad \mathbf{A} \mathbf{x} \leq \mathbf{b}, \quad \mathbf{0} \leq \mathbf{x} \leq \mathbf{1} \end{aligned}$$

The goal of the new constraint (2) is to reduce the search space, as shown in the sequel. Notice that the constraint $y - \hat{y} \geq 0$ does not affect the programming problem above, as \hat{y} is the value of the objective function corresponding to an unconstrained minimization.

The general scheme for solving the relaxed QP, assuming $\hat{\mathbf{x}}$ is not a feasible solution, is the following:

let $\mathbf{A}_0 = \begin{bmatrix} \mathbf{A} \\ \mathbf{I} \end{bmatrix}$, $\mathbf{a}_0 = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$, $\mathbf{b}_0 = \begin{bmatrix} \mathbf{b} \\ \mathbf{1} \end{bmatrix}$;

let \mathbf{x}_0 be an arbitrary point in $[0, 1]^n$; let $m = 0$;

do { $m = m + 1$; $\mathbf{A}_m = \begin{bmatrix} \mathbf{A}_{m-1} \\ \mathbf{x}_{m-1}^T \mathbf{C} + \mathbf{c}^T \end{bmatrix}$,

$\mathbf{a}_m = \begin{bmatrix} \mathbf{a}_{m-1} \\ -1 \end{bmatrix}$, $\mathbf{b}_m = \begin{bmatrix} \mathbf{b}_{m-1} \\ \frac{1}{2} \mathbf{x}_{m-1}^T \mathbf{C} \mathbf{x}_{m-1} \end{bmatrix}$;

solve the LP problem: minimize $y - \hat{y}$ subject to

$$\begin{bmatrix} \mathbf{A}_m & \mathbf{a}_m \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ y - \hat{y} \end{bmatrix} \leq \mathbf{b}_m ; \quad \mathbf{x} \geq \mathbf{0} , \quad y - \hat{y} \geq 0 ;$$

let (\mathbf{x}_m, y_m) be the optimal solution of the LP problem;

} while $(\frac{1}{2} \mathbf{x}_m^T \mathbf{C} \mathbf{x}_m + \mathbf{c}^T \mathbf{x}_m > y_m - \hat{y} + \epsilon)$;

The algorithm stops when y_m is close enough from $f(\mathbf{x})$. Each new iteration introduces a new constraint, restraining the search space (see Fig. 1). Indeed, the new constraint appended at iteration $(m + 1)$

$$(\mathbf{x}_m^T \mathbf{C} + \mathbf{c}^T) \mathbf{x} - (y - \hat{y}) \leq \frac{1}{2} \mathbf{x}_m^T \mathbf{C} \mathbf{x}_m \quad (3)$$

is not fulfilled by the solution (\mathbf{x}_m, y_m) obtained at the previous iteration, as it contradicts the looping condition.

The iterative process can be proven to be convergent. The practical convergence speed is controlled by the parameter ϵ . A small value of ϵ entails a slow convergence rate, while a too high value can lead to a poor approximate solution. When modeling a new assignment problem, a preliminary tuning process is necessary – a drawback acceptable in most practical cases though.

4 High-level synthesis applications

This section describes briefly the quadratic models for two assignment problems encountered in the synthesis of real-time signal processing applications, addressed in the design environment CATHEDRAL [5]. The first refers to the assignment of operation clusters to application specific units (ASU's) [9]. The second refers to the assignment of groups of signals to background memories [2]. These applications have provided the benchmark tests for our quadratic optimization approach (see Section 5).

4.1 The ASU assignment model

In the CATHEDRAL design system, the atomic operations of the signal flow graph are grouped into clusters. The goal of the assignment problem is to assign each of the N_{cl} clusters to exactly one of the N_{ASU} application specific units [9]. The objective function to be minimized represents a measure of the total overhead area involved by the assignment of clusters to ASU's. This function is constructed based on the pairwise compatibility of clusters, i.e., the area overhead encountered when two clusters are assigned to the same ASU. The overhead is zero if the two clusters are identical, and it becomes larger when the clusters are more dissimilar.

A natural formal model of this assignment problem is a QP having the objective function $\sum_{s=1}^{N_{ASU}} \sum_{i,j} C_M(i, j) \cdot x_{i,s} \cdot x_{j,s}$, where $C_M(i, j)$ represents the compatibility measure of two clusters i, j [9], and $x_{i,s}$ is a boolean variable equal to 1 iff cluster i is assigned to ASU s . This function is an upper-bound for the area overhead involved in multiplexing N_{cl} clusters on N_{ASU} application specific units [9].

Two types of constraints are needed. First, each cluster should be uniquely assigned to a single ASU: $\sum_{s=1}^{N_{ASU}} x_{i,s} = 1$, $i = 1, \dots, N_{cl}$. Second, the number of clusters that are assigned to the same ASU should not exceed a given cycle budget N_{cyc} : $\sum_{i=1}^{N_{cl}} x_{i,s} \leq N_{cyc}$, $s = 1, \dots, N_{ASU}$.

The assignment of operation clusters to ASU's has thus been modeled as a QP-type problem. The matrix in the objective function is not necessarily positive definite. In [9], the assignment of clusters to ASU's has been solved employing the Glover's linearization approach [10] – that converts a quadratic problem to an integer linear program (see Section 2). With this approach, it is possible to optimally solve small and medium size problems. For large size problems, an approximate approach – as the one presented in Section 3 – is definitely needed.

4.2 The signal-to-memory assignment model

In previous binary assignment formulations (see [6]), each boolean variable x_{ij} characterized the assignment of a scalar signal to a memory. In [2], a memory allocation approach – based on partitioning the multi-dimensional signals in real-time signal processing applications – has been described. In our model, the binary variable x_{ij} defines the assignment of a group of signals i (resulting after partitioning) to a memory j . The outcome of the preceding background memory allocation step is a set of m memories, with a determined port structure and size [2].

The assignment cost – which must be minimized – is evaluated as an area overhead due to three factors:

1) The assignment of a group of signals i of b_k bits to a memory j having a superior word-length $b_j > b_k$. As the area per bit of memory j is $A_j / b_j N_j$ (where N_j, A_j are the number of locations and area of memory j), assigning n_i signals of b_k bits to memory j results in a loss of $n_i(b_j - b_k)$ bit locations. The total area overhead is estimated to be:

$$C_1 = \sum_{k=2}^m \sum_{i \in COL_k} \sum_{j=1}^{k-1} n_i \frac{b_j - b_k}{b_j} \frac{A_j}{N_j} x_{ij}$$

where COL_k is the collection of groups of signals of b_k bits.

2) The assignment of groups of signals belonging to the same operand in the behavioral specification to different memories. Intuitively, minimizing this term tends to reduce the address logic for each memory as it is more likely that these signals can then share the same address equations and thus

address logic. This cost term is proportional to:

$$C_2 = \sum_{k=1}^m \sum_{(i_1, i_2) \in \text{op}d_k} \sum_{j=1}^k (x_{i_1 j} - x_{i_2 j})^2$$

where $\text{op}d_k$ are operand domains of signals of b_k bits in the behavioral description.

3) The memory allocation technique described in [2] provide the optimal values of the memory sizes N_k relative to an area model for on-chip memories. The third cost term, C_3 , penalizes the assignment solutions resulting in memory dimensions different from the “optimal” ones, implying therefore an area overhead. This term is also quadratic, as both *positive* and *negative* differences has to be penalized:

$$C_3 = \sum_{k=1}^m \left(\sum_{i \in \text{maxCOL}_{k,m}} n_i x_{ik} - N_k \right)^2 \cdot \frac{A_k^2}{N_k^2}$$

where $\text{maxCOL}_{k,m}$ is the maximal collection of groups of signals between b_k and b_m bits (thus able to share memory k) for which the life-times are overlapping.

A first set of constraints states that each group of signals is to be assigned to one memory having at least the necessary word-length. A second set of linear constraints ensures that the assignment solution does not result in more simultaneous *read* and *write* accesses than what is compatible with the memory port structure yielded by the allocation step [2].

This assignment problem was tackled in [2] employing a two-phase heuristic approach. First, the groups of signals were sequentially selected and assigned to the memories, aiming to decrease the cost function as much as possible. Afterwards, the initial assignment was iteratively improved by means of a branch-and-bound technique, which proved to be computationally expensive even for medium-size problems.

5 Overview of the experimental results

The implementation of the presented approach was done in C++, employing the mixed ILP package LAMPS [14]. The assignment experiments have been carried out on an HP 9000/735 workstation. The results presented below correspond to practical QP problems derived from the quadratic models described in Section 4.

The names of the high-level synthesis applications are given in the first column of the tables. The parameters represent the number of groups of objects (operation clusters or groups of signals) to be assigned, and the number of partitions (ASU’s or memories). Columns 2, 3, and 4 display the dimensions of the QP problems: the number of variables, the number of cross-products – in order to indicate the sparsity of the matrix, and the number of constraints – the number of rows of matrix \mathbf{A} . Column 5 displays the dimensions (variables, constraints) of the equivalent ILP problem, in

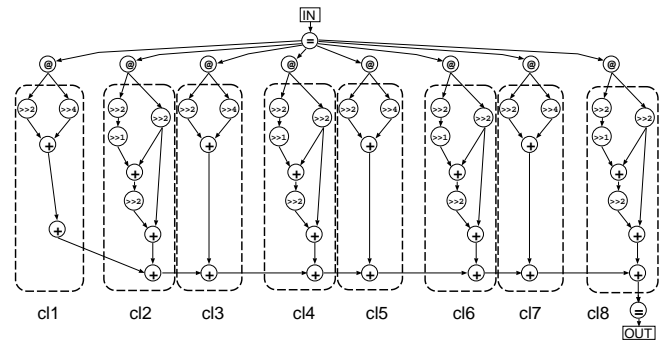


Figure 2: Clustered flow graph of the 8-tap FIR example

case the Glover - Woolsey transformation rules [10] would be applied. The proper experimental results are presented in the next two columns: the cost values of the assignment solution yielded by our approximate approach, along with the corresponding CPU times expressed in seconds.

5.1 Experimental results for the ASU assignment model

The 8-tap FIR filter (Figure 2) is a well known high level synthesis benchmark [12]. We shall target an implementation where the constant multiplications have been expanded into additions and shift operations. We will assume that the signal flow graph has been partitioned into eight clusters (cl1,...,cl8), corresponding to the eight sections of the FIR. In order to illustrate our assignment method, we will furthermore assume that there are only two different constant values, leading to two types of clusters. We target an implementation with two ASU’s ($N_{ASU} = 2$) and with a throughput of one evaluation every four clock cycles ($N_{cyc} = 4$). The compatibility between the two cluster types is 0.125 mm^2 . In the optimal solution, all clusters of type 1 are assigned to one ASU and all clusters of type 2 to the other ASU.

A number of experiments were carried out on larger examples. All these examples are FIR structures of various size. The results are summarized in Table 1. The examples labeled “rfir” and “sfir” have different multiplication coefficients. The first number in the example name is the number of clusters (i.e. the number of objects to be assigned), the second number is the desired number of partitions.

The binary QP problems were also solved using the exact linearization approach [10]. The optimal values (Opt*) of the quadratic cost functions are listed in column 8, along with the run times (CPU*) corresponding to Glover’s method in column 9. Our approximate optimization yielded good results (the errors are inferior to 20%), and the computational effort was significantly lower.

5.2 Experimental results for signal-to-memory assignment

The real-time signal processing applications from which the tested QP problems have been derived are the following: (1)

Example	N_{var}	N_{prod}	N_{constr}	Equiv. ILP size	Cost	CPU(s)	Opt*	CPU*(s)
rfr_16_2	40	240	18	280 × 538	11.69	0.1	10.72	42
sfr_16_2	40	240	18	280 × 538	0	0.9	0	9.4
sfr_16_3	52	360	19	412 × 791	3.0	1.2	0	16
sfr_24_3	80	828	27	908 × 1763	4.33	1.6	0	83
sfr_32_2	74	992	34	1066 × 2092	15.9	5.0	0	1852
sfr_32_3	104	1488	35	1592 × 3115	0	2.9	0	406

Table 1: Experimental results for the ASU assignment model on 16-, 24-, and 32- tap filters

Example	N_{var}	N_{prod}	N_{constr}	Equiv. ILP size	Cost	CPU(s)	Cost**	CPU**(s)
SVD_90_2	180	780	540	960 × 2280	1292	15	2696	92
SVD_90_3	270	1170	720	1440 × 3330	3318	34	3806	137
SVD_90_4	360	1560	900	1920 × 4380	1552	76	1952	183
SVD_180_3	540	4125	1440	4665 × 10230	2643	110	3517	272
SVD_180_4	720	5500	1800	6220 × 13520	4201	251	4542	363
SVD_180_5	900	6875	2160	7775 × 16810	6886	386	7287	455
MOT_296_3	888	11313	2368	12201 × 25882	209963	246	215776	448
MOT_296_4	1184	15084	2960	16268 × 34312	320561	394	324108	596
MOT_428_2	856	16250	2568	17106 × 35924	9803	239	10415	432
MOT_428_3	1284	24375	3424	25659 × 53458	124479	552	125956	647

Table 2: Experimental results for the signal-to-memory assignment model

a singular value decomposition (SVD) updating algorithm – algebraic approach used, for instance, in image coding, system identification, recursive least squares estimation, and beamforming; (2) the kernel of a motion detection (MOT) algorithm from video coding applications.

It can be noticed that the resulting QP problems are much larger than in the previous subsection. An exact optimal solution could be obtained only for the first example in Table 2, by converting the problem into a mixed ILP. The optimal result was 932, but the running time to obtain it was about 50 minutes CPU, in contrast to 15 seconds for our approximate method. The other assignment problems could not be solved by Glover's linearization technique [10] because of the resulting huge size of the mixed ILP's.

In order to carry out a thorough assessment of the approximate approach, the assignment problems were also solved applying the heuristic technique mentioned in subsection 4.2: the values of the objective function (Cost**) and the corresponding run times (CPU**) are displayed in columns 8 and 9. Although the heuristic approach could find an initial assignment solution for each problem in a few seconds, the iterative improvement proved to be computationally expensive. The approximate optimization yielded lower cost solutions and required less CPU time.

6 Conclusions

Several assignment problems in high-level synthesis can be modeled as binary quadratic programming problems. These formal models are usually considered as presenting only a theoretical interest since the exact techniques existent in the literature are extremely time-expensive or even untractable when dealing with realistic problems. In this paper an ap-

proximate binary quadratic optimization technique has been presented. This method – based on alternate continuous relaxations and partial assignments – has been used to solve practical assignment problems in high-level synthesis.

References

- [1] E. Balas, "An additive algorithm for solving linear programs with 0-1 variables," *Oper. Res.*, Vol. 13, pp. 517-546, 1965.
- [2] F. Balasa, F. Catthoor, H. De Man, "Dataflow-driven memory allocation for multi-dimensional processing systems", *Proc. IEEE Int. Conf. Comp. Aided Design*, pp. 31-34, 1994.
- [3] E.R. Barnes, "An algorithm for partitioning the nodes of a graph," *SIAM J. Alg. Discrete Methods*, Vol. 3, No. 4, 1982.
- [4] M. Carter, "The indefinite zero-one quadratic problem," *Discrete Applied Math.*, Vol. 7, pp. 23-44, 1984.
- [5] H. De Man *et al.*, "Architecture-driven synthesis techniques for mapping digital signal processing algorithms into silicon," *Proc. of the IEEE*, Vol. 78, No. 2, pp. 319-335, 1990.
- [6] G. De Michelli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.
- [7] M.R. Garey, D.S. Johnson, L. Stockmeyer, "Some simplified NP-complete graph problems," *Theor. Computer Science*, Vol. 1, pp. 237-267, 1976.
- [8] A.M. Geoffrion, "Integer programming by implicit enumeration and Balas' method," *SIAM Review*, Vol. 9, No. 2, 178-190, April 1967.
- [9] W. Geurts, F. Catthoor, H. De Man, "Quadratic zero-one programming based synthesis of application specific data paths," *IEEE Trans. CAD*, Vol. 14, No. 1, pp. 1-11, 1995.
- [10] F. Glover, E. Woolsey, "Converting the 0-1 polynomial programming problem to a 0-1 linear program," *Operations Research*, Vol. 22, No. 1, pp. 180-182, Jan. 1974.
- [11] R.D. McBride, J.S. Yorlmark, "An implicit enumeration algorithm for quadratic integer programming," *Management Science*, Vol. 26, No. 3, pp. 282-296, March 1980.
- [12] N. Park, A.C. Parker, "Sehwa: a software package for synthesis of pipelines from behavioral specifications," *IEEE Trans. on CAD*, Vol. CAD-7, No. 3, pp. 356-370, March 1988.
- [13] H.A. Taha, "A Balasian-based algorithm for zero-one polynomial programming," *Management Science*, Vol. 18, No. 6, pp. 328-343, Feb. 1972.
- [14] ***, *LAMPS User Guide – Linear and Mathematical Programming System*, Advanced Math. Software Ltd., 1993.