Optimal ILP-based Approach for Throughput Optimization Using Simultaneous Algorithm/Architecture Matching and Retiming

Y.G. DeCastelo-Vide-e-Souza*
Dept. EE-Systems
Univ. Southern California

M. Potkonjak
C&C Res. Labs
NEC USA

and

Alice C. Parker
Dept. EE-Systems
Univ. Southern California

Abstract – System level design and behavior transformations have been rapidly establishing themselves as design steps with the most influential impact on final performance metrics, throughput and latency, of a design. In this paper we develop a formal ILP-based approach for throughput and latency optimization when algorithm-architecture matching, retiming, and pipelining are considered simultaneously. The effectiveness of the approach is demonstrated on several real-life examples.

1 Introduction

Throughput and latency are consistently among the most important keys for differentiation among electronics products. The design techniques (r)evolution is that the emphasis in the design process is inevitably shifting toward higher levels of abstraction. Interestingly, recent case studies and experience indicate that the higher-level design decisions impact on optimization is more significant than lower-level decisions.

It is important to note that the selection of the best architecture for a given application is a non-trivial problem. An ample confirmation of strong dependence between architecture and algorithms is presented for numerous computers (architectures) and many algorithms in [1].

Our goal in the research presented in this paper is to develop an approach which, in an integrated and formally sound way, combines the effectiveness of transformations and system-level design tasks for throughput optimization. In particular, we address the problem of throughput optimization using matching between algorithm, architecture, and retiming.

The types of system specification we are considering describes systems where there is repeated computation on a sequence (stream) of data sets. Initially, each data set is usually processed during a single iteration, or major cycle (within that major cycle there can be many minor, or clock cycles). We can transform the computation, so that some of the processing occurs during an earlier or later data cycle. In order to differentiate the tasks which are delayed one or more cycles, delays are inserted in the flow graph to indicate that the task following the delay is postponed. We call such delays *hierarchical delays*. The problem we solve is to position delays so that the throughput of the system is maximized. We simultaneously select the algorithm which implements each task or block as well as the processor(architecture) to which each task or block will be allocated, so that the combination of simultaneous algorithm/processor selection and *hierarchical delay* positioning is used to find an optimal throughput.

The problem of throughput optimization using algorithm selection was recently addressed in [15]. However, our scope of the problem is significantly broadened: we consider both algorithm selection and architecture selection, and more importantly, we simultaneously consider two important transformations with algorithm/architecture matching: retiming and pipelining. Also, our ILP-based algorithm is capable of achieving provably optimal solutions on all practically sized examples in relatively short run times.

2 Problem Formulation

2.1 Assumptions, Basic Definitions, and Notation

We use a hierarchical synchronous data flow model of computation, which is suitable for describing a periodic computation on a semi-infinite stream of data. The selected model of computation assumes that the target design is described by a computation which is composed of building (functional) blocks and hierarchical delays. Building blocks themselves can possess sequential behavior and are recursively defined, so that at the lowest level of hierarchy only standard control structures (procedures, conditionals, and loops), algebraic operations, and internal delays are present. Hierarchical delays (states) are used to denote the boundary between successive iterations and can be repositioned using retiming only to the edges of the highest level of hierarchy.

This computational model is a slight modification of the standard synchronous data flow model. It is commonly used to specify many designs in video, image, DSP, communication, control, and information theory-related applications at the behavioral level. We allow an arbitrarily high sharing of hardware resources, but only to the extent that it does not influence critical paths in the control data flow graph (CDFG). In the rest of the paper we will interchangeably use both CDFG

*also known as J. C. DeSouza-Batista.

32nd ACM/IEEE Design Automation Conference ®

and signal flow graph (SFG) to refer to computations represented using the hierarchical synchronous data-flow model of computation.

The sampling period of the computation is equal to the maximum rate at which it can accept and process the data samples of the input data streams. In the common control-data flow graph (CDFG) representation, the sampling period is the longest distance between any two delays, or between any primary input and any delay, or between any delay and any primary output, or between any primary input and any primary output in the CDFG. Throughput (sampling rate) is the inverse of sampling period.

Latency is the time interval between the arrival of the set of inputs corresponding to one iteration of computation and the delivery of all corresponding outputs as defined by the specification. In the common CDFG domain, latency is the longest distance between any pair of delays or between any primary input and any of the corresponding outputs.

2.2 Illustrative Example and Graph Theoretic Problem Formulation

In this subsection we present an illustrative, real life example of simultaneous algorithm/architecture matching and retiming for throughput optimization. We also pose the optimization problem at the graph level of abstraction.

Figure 1 shows the CDFG of the transform domain adaptive LMS filter [5]. Sigma denotes vector sum, DCT is the direct discrete cosine transform. D denotes a hierarchical delay. KD indicates k hierarchical delays. If each input edge of a block has k hierarchical delays, the functionality of the design is not altered if those delays are deleted and replaced by k hierarchical delays on each output of the block or vice versa. This delay manipulation technique is retiming [12]. The problem we pose here is where to insert the delays so that throughput is maximized while selecting an implementation for each block. This example is solved later in the paper.

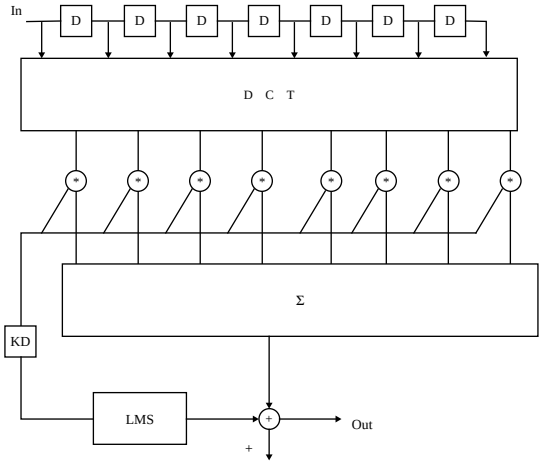

Figure 1: Throughput Optimization using Simultaneous Algorithm/Architecture Matching and Retiming

Each block in Figure 1 can be implemented using a variety of algorithms (CDFGs) and architectures; e.g. commonly used DCT algorithms include Lee's, Wang's, planar rotation, DIF (decimation in frequency), DIT (decimation in time) and FFT-based algorithms [5]. The architecture choices include various DSP, video, microcontroller, and microprocessor chips from many vendors.

Equations (2.1) and (2.2) show the distances (*propagation time delays*) between each pair of nodes for two among several other DCT algorithm/architecture matchings. The matrices are formed in a such way that element $a_{i,j}$ represents the distance between input i and output j.

$$DCT_i = \begin{pmatrix} 4 & 5 & 6 & 6 & 8 & 8 & 8 & 8 \\ 4 & 5 & 6 & 6 & 8 & 8 & 8 & 8 \\ 4 & 5 & 6 & 6 & 8 & 8 & 8 & 8 \\ 4 & 5 & 6 & 6 & 8 & 8 & 8 & 8 \\ 4 & 5 & 6 & 6 & 8 & 8 & 8 & 8 \\ 4 & 5 & 6 & 6 & 8 & 8 & 8 & 8 \\ 4 & 5 & 6 & 6 & 8 & 8 & 8 & 8 \\ 4 & 5 & 6 & 6 & 8 & 8 & 8 & 8 \end{pmatrix} \quad (2.1)$$

$$DCT_{ii} = \begin{pmatrix} 4 & 5 & 5 & 4 & 5 & 8 & 5 & 8 \\ 4 & 5 & 5 & 4 & 6 & 8 & 5 & 5 \\ 4 & 5 & 6 & 4 & 4 & 8 & 5 & 5 \\ 4 & 5 & 7 & 4 & 4 & 8 & 5 & 5 \\ 4 & 5 & 5 & 4 & 6 & 8 & 5 & 5 \\ 4 & 5 & 5 & 4 & 6 & 8 & 5 & 5 \\ 4 & 5 & 6 & 4 & 7 & 8 & 5 & 5 \\ 4 & 5 & 7 & 4 & 6 & 8 & 5 & 5 \end{pmatrix} \quad (2.2)$$

The goal is to select an algorithm/architecture pair for each building block of the hierarchical CDFG so that after the application of retiming, throughput is maximized. It is important to note that there is a strong dependence between retiming and algorithm selection, and that successive resolution of the two optimization steps may result in an inferior solution. Figure 3 illustrates this point.

The design in Figure 3a has three functional blocks. Only block B has more than one choice for its algorithm. If the throughput is optimized first, the best choice is obviously algorithm A1 which results in a sampling rate of 30 cycles. However, if retiming is considered simultaneously, the best choice is A2. This is because, after the application of retiming, the critical path can be reduced to 15 cycles, as shown in Figure 3b. Note that initially (before retiming) this algorithm selection makes the critical path 35 cycles long.

2.3 Complexity

It is well known that retiming is a polynomial complexity problem [12]. It has been demonstrated that throughput optimization using algorithm selection belongs to the class of NP-complete problems [15]. Since the problem presented in [15] is a special case of the problem discussed in this paper, we can conclude that the new problem is also of high complexity, at least NP-complete. Interestingly, the proof in [15] is built on the assumption that all potential algorithms have states (delays).

Numerous real life applications do not have delays in their specifications. We derived a new proof, using a transformation from the equal-subset problem which shows that throughput (latency) optimization using algorithm selection is an NP-complete problem, even when only algorithms with no delays are considered [5]. The new proof is also more general in the

sense that it covers variants of the problem when pipelining is used in conjunction with retiming and the target is latency optimization instead of throughput maximization.

3 Related Work

3.1 Algorithm and Architecture Matching

Although the interaction between the computation and architecture has been sporadically addressed for a long time, only recently it has attracted complete attention in the CAD optimization-intensive framework of hardware/software codesign [11] [2], [8].

Potkonjak and Rabaey were first to consider a particular instance of the problem considered in this paper in the optimization-intensive hardware/software codesign framework [15]. Although they restricted their attention only to the algorithm selection problem, assuming a fixed hardware platform, and did not consider any transformations, their heuristic algorithm demonstrated an order of magnitude improvement in throughput and area.

3.2 Retiming

Retiming has been used under a variety of names and algorithms in digital signal processing, control theory and applications for several decades, mainly to derive various canonical forms for linear computations and to optimize throughput and cost of fully hardwired implementations of linear filters. Note that retiming is just a member of a large set of more general algorithm transformations such as ones studied by Park [13]

In the early 1980s, retiming of delays corresponding to boundaries of control loops had been introduced as a software compilation technique under the framework of software pipelining.

Arguably, the greatest impact on the popularization of retiming was produced by a series of papers by Leiserson and Saxe [12], who were the first to treat retiming as a combinatorial optimization problem and to present polynomial algorithms for minimizing the delay of the critical path and the number of delays in the design.

The first application of retiming in CAD and high level synthesis was proposed by Goosens et al. [7] where its application was restricted to minimization of the number of registers of fully hardwired designs. Retiming in high level synthesis was recognized by the developers of Hyper, who demonstrated its effectiveness for the minimization of all components of data path (execution units, registers, and interconnect) [14], power and fault-tolerance.

To the best of our knowledge this work represents the first optimization-intensive and explicit use of retiming in system-level synthesis.

3.3 Integer-Linear Programming

We conclude our overview of related work by briefly surveying several integer linear programming (ILP) high level synthesis techniques, which are, to a limited extent, similar at the algorithmic level to our techniques.

Since Hafer and Parker [10] first proposed mixed ILP as the optimization method for datapath synthesis, numerous authors demonstrated the effectiveness of this method as a scheduling, assignment, and resource allocation tool. Notable work includes research reported in [6]. More recently ILP was also used as an optimization vehicle in system-level synthesis [16] [4].

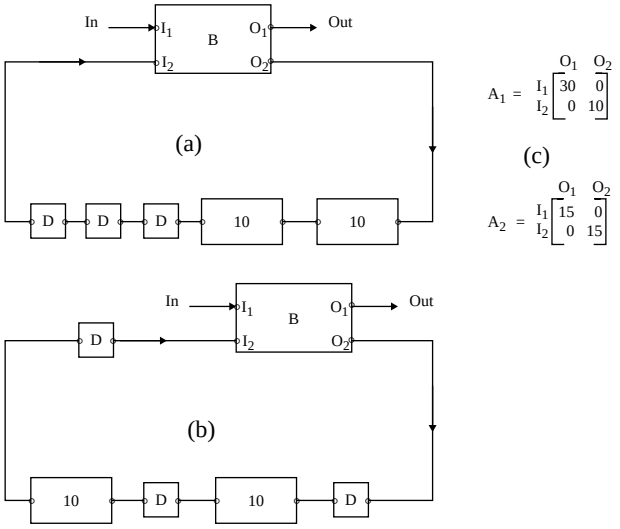

Figure 2: Importance of Simultaneous Consideration of Algorithm/Architecture Selection and Retiming

4 Mathematical Model

In this section we develop an efficient ILP formulation for simultaneous algorithm/architecture matching and retiming. We call each of the computation nodes in the system-level synchronous dataflow graph (e.g. DCT block in Figure 1) *computational block*. We start first with an important special case when all computational blocks are of type n inputs - 1 output. This special case is used as a basis for the development of a fully general ILP model where each of n inputs and m outputs has independent timing. The special case model (n inputs - 1 output) is also interesting in itself, because often in design practice the basic block sends its output data in one packet. This organization of the computation usually results in a simpler, less expensive implementation. We conclude this section by discussing the size of the ILP model.

4.1 Case I: *n inputs - 1 output* computation blocks

Given a synchronous data-flow computation composed of computation blocks (functional elements) and delay elements (states), where each computation block u has n_u inputs and one output ($(n_u, 1)$ *block-type*). We can have two timing disciplines:

i) Computation blocks have only *time equidistant* inputs.

ii) Computation blocks have *time non-equidistant* inputs.

DEFINITION 1: Given a computation block u, u will have *time equidistant* inputs, if for every pair of inputs i and j of the block u , $i \neq j$, we have:

$$\triangle(input(i,u)\ to\ output(u)) = \triangle(input(j,u)\ to\ output(u)) = d_u \quad (4.0)$$

Where $\triangle(\bullet)$ = computation (propagation) delay

DEFINITION 2: A computation block is of (1,1) type if and only if it has only one input and one output.

A synchronous data-flow with computation blocks of type $(n,1)$ with time equidistant inputs can be modeled by a directed graph $G =< V, E, d, w >$. V is the set of vertices, E is the set of edges, d is a function giving the computation delay (measured in control steps) in each vertex (computation block), and w is the function which denotes the number of delay (state) elements in each edge before retiming [12]. The following theorem was derived by Leiserson and Saxe [12] and it is the basis for our mathematical model.

THEOREM 1. Let $G =< V, E, d, w >$ be a synchronous data-flow [12], let c be an arbitrary positive real number, and r be a function from V to the integers, then r is a legal retiming of G such that $\Phi(G_r) \leq c$ (G_r is the data-flow after retiming) if and only if
a)$r(u) - r(v) \leq w(e)$ for every edge e $= u \rightarrow v$ of G, and
b)$r(u) - r(v) \leq W(u,v) - 1$ for all vertices $u, v \in V$ such that $D(u,v) \geq c$

Where
$\Phi(G_r)$ *is the iteration period measured in control steps after retiming*
$W(u,v) = min\{W_{path}(u,v,p)$*: p is a path from u to v*$\}$
$D(u,v) = max\{D_{path}(u,v,p)$*: p is a path from u to v such that* $W_{path}(u,v,p) = W(u,v)\}$
$W_{path}(u,v,p)$ *is number of delay elements in the path p*
$D_{path}(u,v,p)$ *is the sum of of computation delays of the vertices (blocks) in the path p*

The number of delays in an edge e after retiming $w_r(e)$ is given by the expression below [12], where $w(e)$ is the number of delays before retiming.

$$w_r(e) = w(e) + r(v) - r(u) \tag{4.1}$$

Assuming, without loss of generality, that the designer assumes that no edge should have more than w_M delays, where

$$w_M \geq max_e(w(e)),\ e \in E \tag{4.2}$$

$w_r(e)$ can be expressed as

$$w_r(e) = \sum_{i=1}^{w_M} x_{e,i} \leq w_M \quad , x_{e,i} \in \{0,1\} \tag{4.3}$$

$$x_{e,i} \leq x_{e,i+1} \quad , i = 1, ..., w_M - 1 \tag{4.4}$$

$x_{e,i} = 0$ indicates that less than $w_M - i + 1$ delays will be placed on edge e. $x_{e,i} = 1$ indicates that at least $w_M - i + 1$ delays will be placed on edge e.
Note that constraints (4.2) and (4.3) guarantee that condition (a) of Theorem1 will always be satisfied.

The delay d_u of a vertex u is given by:

$$d_u = \sum_i \sum_j d_{u,i,j}\sigma_{u,i,j}\ ,\ \sigma_{u,i,j} \in \{0,1\} \tag{4.5}$$

where

$$\sum_i \sum_j \sigma_{u,i,j} = 1 \tag{4.6}$$

$\sigma_{u,i,j} = 1$ indicates that block u has been implemented an architecture of cost $C_{u,i,j}$.
In a similar way, the cost $Cost_u$ of a vertex (block) u is given by:

$$Cost_u = \sum_i \sum_j C_{u,i,j}\sigma_{u,i,j} \tag{4.7}$$

where $C_{u,i,j}$ and $d_{u,i,j}$ are the *cost* and *delay* of implementation (i,j) for vertex (block) u. Where implementation (i,j) for computation block u denotes *the use of algorithm* $\aleph_i^u$ *mapped into a processor (architecture) of type* j *to implement block* u.

Let p be a path from vertex u to vertex v ($u \xrightarrow{p} v$). We define $D_{path}(u,v,p)$ and $W_{path}(u,v,p)$ as:

$$D_{path}(u,v,p) = \sum_{a \in p} d_x = \sum_{a \in p} \sum_i \sum_j d_{a,i,j}\sigma_{a,i,j} \tag{4.8}$$

$$W_{path}(u,v,p) = \sum_{e \in p} w(e) \tag{4.9}$$

$W_{path}(u,v,p)$ can be precomputed. On the contrary, $D_{path}(u,v,p)$ depends on the choice of which implementations (algorithm/architecture match) will be used for the vertices (blocks) in the path.

For every path p connecting vertex u to vertex v we have

$$r(u) - r(v) = \sum_{e \in p} w(e) - \sum_{e \in p} \sum_{i=1}^{w_M} x_{ei} \tag{4.10}$$

$$r(u) - r(v) = W_{path}(u,v,p) - \sum_{e \in p} \sum_{i=1}^{w_M} x_{ei} \tag{4.11}$$

This expression can be generalized to a circular walk p_c in G. There is a circular walk starting in vertex u if and only if there is a corresponding cycle in the undirected graph G_o derived from G by not considering the directions of the edges of G and assuming the same weight values ($w(e)$) as in G.

$$W_{circ}(u,p_c) = \sum_{e \in p_c} sign(e,p_c)w(e) - \sum_{e \in p_c} sign(e,p_c) \sum_{i=1}^{w_M} x_{ei} \tag{4.12}$$

$$\forall\ circular\ walk\ p_c\ in\ G,\ W_{circ}(u,p_c) = 0 \tag{4.13}$$

where
$sign(e,p_c) = +1$ if e has the same direction of p_c
$sign(e,p_c) = -1$ if e has direction opposite to p_c

The maximum size set of linear independent equation involving circular walks in G can be found by finding the minimum spanning tree of G_o, where the distance between two adjacent vertices u, v is given by $w(e)$, e=(u,v). Each edge $e_* = (u,v)$

which is not in the spanning tree defines a unique cycle made by the edge itself and the path between u and v in the spanning tree. As each cycle in G_o defines a corresponding circular walk in G, we find a set of linear independent equations. This set is maximum because any other circular walk will derive an equation that is a linear combination on the equations in the set. The size of this set is $O(|E|-|V|+1)$, because $|E| \geq |V|-1$ (G_o is a connected graph). A tree shaped graph does not have circular walk equations.

4.1.1 Linearization of condition (b)

The condition (b) of Theorem 1 refers to particular paths $p(u \longrightarrow v) \in G$ such that $D_{path}(u,v,p)$ is maximum among all paths from u to v with minimum $W_{path}(u,v,p)$. As $w(e)$, $e \in E$, is known, we can easily find the set of paths $P(u,v) = \{p|W_{path}(u,v,p)\ is\ minimum\} = \{p|W_{path}(u,v,p) = W(u,v)\}$ by executing the all shortest path algorithm on G. However we cannot decide which path $p \in P(u,v)$ will have maximum $D_{path}(u,v,p)$ before we have chosen the particular implementation (algorithm/architecture mapping) of each vertex (block) $v_o \in p$. Therefore we need to check condition (b) for each path $p \in P(u,v)$. This is accomplished by means of the following linearization, assuming the *iteration period upper bound* c to be a positive integer.

$\forall p \in P(u,v)$

$$D_{path}(u,v,p) > c \Longleftrightarrow D_{path}(u,v,p) \geq c+1 \tag{4.14}$$

$$D_{path}(u,v,p) \geq c+1-(1-\alpha_p)\Omega_{\alpha_p} \tag{4.15}$$

$$-D_{path}(u,v,p) \geq -c-\alpha_p\Omega_{\alpha_p} \tag{4.16}$$

where
$\alpha_p \in \{0,1\}$.
Ω_{α_p} is a sufficient large constant, $\Omega_{\alpha_p} \geq max(c+1-D_{path}(u,v,p), D_{path}(u,v,p)-c)$ for all possible choices of block implementations (algorithm/architecture mappings).

$\forall p \in P(u,v)$

$$r(u)-r(v) \leq W(u,v)-1+(1-\beta_{u,v})\Omega_{\beta_{u,v}} \tag{4.17}$$

$$-r(u)+r(v) \leq -W(u,v)+\beta_{u,v}\Omega_{\beta_{u,v}} \tag{4.18}$$

where $\beta_{u,v} \in \{0,1\}$.
$\Omega_{\beta_{u,v}}$ is a sufficient large constant.
By equation (4.11), and as $p \in P(u,v) \Longleftrightarrow W(u,v) = W_{path}(u,v,p)$, we find the more tight constraints below.

$$0 \leq \sum_{e \in p} \sum_{i=1}^{w_M} x_{e,i} - \beta_{u,v} \tag{4.19}$$

$$\sum_{e \in p} \sum_{i=1}^{w_M} x_{e,i} \leq \beta_{u,v} * w_M * |\{e, e \in p\}| \tag{4.20}$$

where $|S|$ is the number of elements of the set S.

Inequality 4.20 leads to the following additional constraint:

$$\forall e \in p,\ x_{e,i} \leq \beta_{u,v},\ 1 \leq i \leq w_M \tag{4.21}$$

Therefore condition (b) can be expressed by the following constraint:

$$\forall p \in P(u,v),\ \alpha_p \leq \beta_{u,v} \tag{4.22}$$

Constraints (4.15) and (4.16) can be expressed in a more tight way as indicated below.

$$D_{path}(u,v,p) \geq c+1-(1-\alpha_p)\Omega^0_{\alpha_p} \tag{4.23}$$

$$-D_{path}(u,v,p) \geq -c-\alpha_p\Omega^1_{\alpha_p} \tag{4.24}$$

where

$$\Omega^0_{\alpha_p} = max(0, c+1-D^{min}_{path}(u,v,p)) \tag{4.25}$$

$$\Omega^1_{\alpha_p} = max(0, -c+D^{max}_{path}(u,v,p)) \tag{4.26}$$

$D^{max}_{path}(u,v,p)$ is the *maximum* possible value of $D_{path}(u,v,p)$
$D^{min}_{path}(u,v,p)$ is the *minimum* possible value of $D_{path}(u,v,p)$

For the *time equidistant* case, we find the following two results.

$$D^{max}_{path}(u,v,p) = \sum_{v_o \in p} d^{max}_{v_o} \tag{4.27}$$

$$D^{min}_{path}(u,v,p) = \sum_{v_o \in p} d^{min}_{v_o} \tag{4.28}$$

4.1.2 Summary of the model

We briefly describe the 0-1 ILP formulation for the case with *(n,1)-type* computation blocks, having *time equidistant* inputs, in the following way.

Given a positive integer c, upper bound on the *throughput* measured in number of control steps,

$Minimize\ \sum_{u \in V} Cost_u$

subject to the constraints given by 4.2-9, 4.12-13, 4.19-28, given the Boolean variables $\sigma_{u,i,j}, x_{e,i}, \alpha_p, \beta_{u,v}$ such that $u,v \in V,\ e \in E,\ p \in P(u,v)$.

4.1.3 Blocks with *time non-equidistant* inputs and more than one output

The present mathematical model can be easily extended to handle synchronous data-flow computations with computation blocks having *time non-equidistant* inputs and more than one output (Case II) by applying a set of *substitutions* discussed in the technical report of this paper [5].

4.2 Size of the model

The size of the model is highly dependent on the system-level flow graph, because many constraints in the mathematical model are not needed for particular ranges of values for c as well as for some combinations of implementation delays of the computational block. We developed many methods to prune as many constraints as possible at compile time (model generation) by pre-evaluating the values of many variables whenever it is possible to do so. These optimizations are case specific, and for the sake of brevity, they are included only in the technical report of this paper. However, we can estimate a worst-case value for the size of the model (4.38) (4.39), assuming a system with only $(n,1)$ time equidistant blocks:

$$O(\#variables) = O(\#constraints)$$
$$= O(\#blocks + \#(valid\ paths)\ ) \quad (4.38)$$

where the number of valid paths is a function of the number of non-zero delay blocks, i.e., blocks in which the maximum *implementation delay* is larger than zero.

$$O(\#(valid\ paths)) = O(\#(non\ zero\ delay\ blocks)^2) \quad (4.39)$$

5 Software Platform and Experimental Results

The ILP-model of a given system-level task flow graph (such as in Figure 1) is automatically generated from the TFG by a 1500 line C program. The model is solved using a branch-and-bound ILP-solver BOZO [9]. The model generator program makes an extensive effort to prune unnecessary constraints by pre-processing (pre-evaluating) many of the constraints in the model, which helps to ensure low CPU times during the optimization phase made by the ILP solver.

The following three audio and video processing examples have been optimized using the proposed methodology and algorithms.
(1) HOM - System for Homomorphic filtering of images [5];
(2) IMAGE - System for Enhancement of compressed images [5];
(3) LMS - Transform domain adaptive LMS filter (Q = 1, 2 and 3) [5];

Parameter Q in the LMS examples denotes different versions of updating algorithms with three different rates of convergence. Table 2 shows the initial throughput and the throughput after optimization. The average and median improvement are by factors of 1.97 and 1.81 respectively, clearly indicating the importance of hierarchical pipelining in system-level design. Run times of the ILP-based optimization algorithm on a SUN4 computer are also given in Table 1. We assume all architectures have the same implementation cost on these benchmarks for the sake of simplicity, so only algorithm selection is performed.

Examples	Sampling Period before optimization	Sampling Period after optimization	Improvement factor (speed-up)	Run-Time (sec) on SUN4
HOM	29	13	2.23	395
IMAGE	21	8	2.63	502
LMS (Q = 1)	21	13	1.62	960
LMS (Q = 2)	29	16	1.81	1270
LMS (Q = 3)	37	24	1.54	1005

Table 1: Throughput maximization

6 Conclusion

We introduced a new system-level synthesis technique for throughput optimization which simultaneously considers algorithm/architecture matching and retiming. The technique is generalized so that pipelining is considered. The effectiveness of the ILP-based optimization algorithm is demonstrated on several real life examples. The application domain of the algorithm is enhanced by exploring the relationship between the new problem and several other problems in behavioral and logic synthesis.

Acknowledgment

The authors wish to thank Mr. Diogenes Da Silva Jr. for the helpful comments during the preparation of this paper.

References

[1] E.C. Anderson, J. Dongarra, "Performance of LAPACK: A Portable Library of Numerical Linear Algebra Routines",*Proc. of the IEEE*, Vol. 81, No. 8, pp. 1094-1102, 1993.

[2] P. Chou, E.A. Walkup, G. Borrielo, "Scheduling for Reactive Real-Time Systems",*IEEE MICRO*, Vol. 14, No. 4, pp. 37-47, 1994.

[3] M.R. Corazao, M. Khalaf, L. Guerra, M. Potkonjak, J. Rabaey, "Instruction Set Mapping for Performance Optimization", pp. 518-521, *ICCAD93*, November 1993.

[4] J. C. DeSouza-Batista and A. C. Parker, "Optimal Synthesis of Application Specific Heterogeneous Pipelined Multiprocessors", *Proceedings of the 1994 Application Specific Array Processor Conference (ASAP)*, San Francisco, August 1994.

[5] Y. G. DeCastelo-Vide-e-Souza, M. Potkonjak and A. C. Parker, "Optimal ILP-based Approach for Throughput Optimization Using Simultaneous Algorithm/Architecture Matching and Retiming", Technical Report *in preparation*, Department of EE-Systems, Univ. Southern California, 1995.

[6] C. H. Gebotys and M. I. Elmasry, "Optimal Syntheis of High-Performance Architectures", *IEEE Journal of Solid-State Circuits*, vol. 27, no. 3, pp. 389-397, 1992.

[7] G. Goossens, J. Wandewalle and H. De Man, "Loop optimization in register-transfer scheduling for DSP- systems", *26th Design Automation Conference*, pp. 826-831, Las Vegas, NV, 1989.

[8] R.K. Gupta, C.N. Coelho, G. De Micheli, "Program Implementation Schemes for Hardware-Software System", *IEEE Computer*, Vol. 27, No. 1, pp. 48-55, 1991.

[9] Hafer, L.J. and Hutchings, E., "Bringing up Bozo", *Tech Rep. CMPT TR 90-2*, School of Computing Science, Simon Fraser University, Burnaby, B.C., 1990.

[10] L. Hafer, A. Parker, "A Formal Method for the Specification, Analysis, and Design of Register-Transfer Level Digital Logic", *IEEE Trans. on CAD*, Vol. 2, No. 1, pp. 4-18, 1983.

[11] A. Kalavade, E.A. Lee, "A Hardware-Software Codesign Methodology for DSP Applications",*IEEE Design & Test of Computers*, Vol. 10, No. 3, pp. 16-28, 1993.

[12] C. L. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry", *Algorithmica*, vol. 6, pp. 5-35, 1991.

[13] N. Park and A. C. Parker,"Theory of Clocking for Maximum Execution Overlap of High-Speed Digital Systems", *IEEE Transactions on Computers*, vol. 37, no.6, pp. 678-690, June 1988.

[14] M. Potkonjak and J. Rabaey, "Optimizing the Resource Utilization Using Transformations",*ICCAD-91*, pp. 88-91, Santa Clara, 1991.

[15] M. Potkonjak, J.M. Rabaey, "Algorithm Selection: A quantitative computation-intensive optimization approach",*ICCAD94*, paper 2B.1, 1994.

[16] S. Prakash, and A.C. Parker, "SOS: Synthesis of Application Specific Heterogeneous Multiprocessor Systems", *Journal of Parallel and Distributed Computing*, no. 16, pp. 338-351. December 1992.