Optimum and Heuristic Synthesis of Multiple Word-Length Architectures

George A. Constantinides, *Member, IEEE*, Peter Y. K. Cheung, *Senior Member, IEEE*, and Wayne Luk, *Member, IEEE*

***Abstract*—This paper explores the problem of architectural synthesis (scheduling, allocation, and binding) for multiple word-length systems. It is demonstrated that the resource allocation and binding problem, and the interaction between scheduling, allocation, and binding, are complicated by the existence of multiple word-length operators. Both optimum and heuristic approaches to the combined problem are formulated. The optimum solution involves modeling as an integer linear program, while the heuristic solution considers intertwined scheduling, binding, and resource word-length selection. Techniques are introduced to perform scheduling with incomplete word-length information, to combine binding and word-length selection, and to refine word-length information based on critical path analysis. Results are presented for several benchmark and artificial examples, demonstrating significant resource savings of up to 46% are possible by considering these problems within the proposed unified framework.**

***Index Terms*—Binding, bitwidth, digital signal processing, field-programmable gate array, high-level synthesis, scheduling, word-length.**

I. Introduction

THE accuracy observable at the outputs of a digital signal processing (DSP) system is a function of the word-lengths used to represent all intermediate variables in the algorithm. However, accuracy is less sensitive to some variables than to others, as is implementation area. It has been demonstrated that by considering error and area information in a structured way, it is possible to achieve highly efficient DSP implementations utilizing different word-lengths for different internal variables [1]–[4].

This paper considers the problem of architectural synthesis for multiple word-length systems. The work described in this paper may be considered as a "post-processing" step to word-length and scaling determination procedures, as illustrated in Fig. 1. Breaking the problem in this way means that the procedures described in this paper, unlike some word-length determination techniques, do not depend on the type of DSP system to be synthesised (restricted to linear time invariant systems in [5], differentiable systems in [6] and [7], and nonrecursive dataflow in [8] and [9]). In addition, the complexity of the synthesis algorithms is reduced by breaking the problem into manageable pieces. In this paper, we concern ourselves only with the architectural synthesis block in Fig. 1.

Manuscript received October 22, 2002; revised March 9, 2003, July 28, 2004.

G. A. Constantinides and P. Y. K. Cheung are with the Department of Electrical and Electronic Engineering, Imperial College, London SW7 2BT, U.K. (e-mail: g.constantinides@ic.ac.uk; p.cheung@ic.ac.uk).

W. Luk is with the Department of Computing, Imperial College, London SW7 2BZ, U.K. (e-mail: wl@doc.ic.ac.uk).

Digital Object Identifier 10.1109/TVLSI.2004.840398

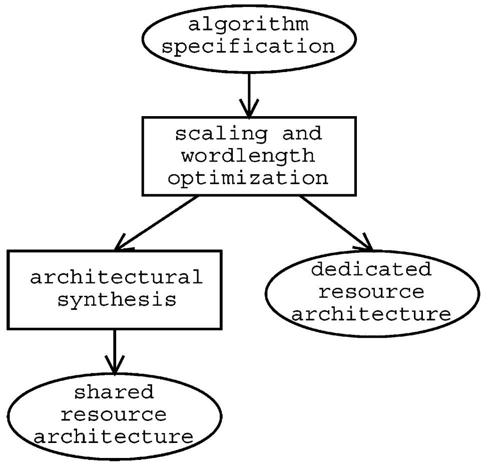

algorithm specification
scaling and wordlength optimization
architectural synthesis
dedicated resource architecture
shared resource architecture

Fig. 1. Overall design flow for general resource binding architectures.

The problems of resource allocation (deciding how many resources of a particular type should be used), resource binding (deciding which operation is to be executed on which resource), and scheduling (deciding at which clock cycle, or "time step," each operation should execute) have all been well studied [10]. However, such work has invariably considered all operations of a particular type, such as "multiply" or "add," to have identical implementations or at least an identical library of possible implementations [11]–[14].

There has been little previous work [2], [15], [16] on architectural synthesis for multiple word-length systems. The multiple word-length paradigm has a significant impact on the traditional problems of high-level synthesis, arising from two factors. First, each computational unit of a specific type, for example "multiply," cannot be assumed to have equal cost in a multiple word-length implementation, since area scales with operator word-length. This issue has been considered by [2] and [15]–[17]. Second, the choice of word-length for an operation can impact on the latency of that operation. For instance, larger bit-parallel multipliers may have longer latency than smaller bit-parallel multipliers. The consideration of multiple word-lengths, therefore, complicates the resource binding problem, and also increases the interaction between binding and scheduling of operations. This issue has not been previously considered in the context of multiple word-length synthesis.

The main original contributions of this paper are therefore:

- the formulation of the multiple word–length architectural synthesis problem;
- the transformation of this problem into an integer linear program;

- the construction of a polynomial time heuristic involving scheduling with incomplete word-length information, combined resource binding and word-length selection, and techniques to iteratively refine word-length information;
- the evaluation of these approaches, showing that area savings of up to 46% are possible even for modest problem sizes, when compared to previous approaches.

Section III provides a more concrete formulation of the problem to be addressed, after which Section IV introduces an integer linear programming (ILP) approach to the problem. The ILP approach is not practical for large problems, due to the computational complexity associated with solving the ILP model. This drawback is the motivation for a polynomial-time heuristic algorithm presented in Section V. Synthesis results are reported and discussed in Section VI and the paper is summarized in Section VII.

II. Background

Choi *et al.* published a short early paper on architectural synthesis for algorithms consisting of operations with different word-lengths [18]. The authors propose a resource binding technique for algorithms that have already been scheduled. The proposed technique partitions the operations into a number of sets. A resource binding is then performed on each of these sets separately, so that operations in different sets cannot share the same resource. The authors compare their approach to a standard "word-length blind" resource binding, and demonstrate that area reductions between 2.8% and 6.8% are obtained over a benchmark set consisting of a fourth-order infinite impulse response filter, a fourth-order least-mean-square adaptive filter, and a 12th-order least-mean-square adaptive filter.

Constantinides *et al.* have described an approach to the architectural synthesis problem [16]. As with [18], the techniques proposed in [16] perform resource binding as a post-processing step, when the schedule is already known. Three distinct techniques are proposed: 1) a branch and bound optimal solution; 2) a simulated annealing heuristic; and 3) a novel application-specific heuristic, each based on the coloring of conflict graphs.[1]

Kum and Sung have developed some approaches to the architectural synthesis problem for algorithms consisting of operations with possibly different word-lengths [2], [15]. These approaches include word-length conscious list scheduling [2], [15], and recently, also word-length conscious integer linear programming scheduling [2]. There is, however, a key simplifying assumption implicitly present in the work of Kum and Sung: no dependence is considered between the word-length of an operation and the latency of that operation. Indeed, all operation latencies are assumed to be one control-step, independent of the word-length of the resource to which they are bound. The authors present results for a fourth-order IIR filter (implemented as two second-order sections) and a fifth-order elliptic filter [2]. These results indicate that the described heuristic results in an area of between 0% and 46% (average 6%) greater than the optimum obtained through an integer linear program.

[1]A conflict graph is a graph where each node corresponds to an operation, and edges exist between two nodes iff their execution intervals overlap [19].

Molina, Mendias, and Hermida have also explored resource binding for prescheduled multiple word-length implementations [17] targeting ASICs, however, this work currently only supports multiple word-length addition; multipliers are implemented by breaking them into adders accumulating partial products. Resource sharing is encouraged by allowing the carry chains of adders to be broken and combined with other adders on a cycle-by-cycle basis.

To summarize, the previous work in this field includes techniques for resource binding only [16]–[18] and combined scheduling and binding [2], with both heuristic [2], [16]–[18] and optimal [2], [16] approaches. None of the techniques presented thus far have considered combined scheduling and resource binding in a context where operational latency can vary with resource word-length.

III. Motivation and Problem Formulation

An algorithm is considered to be made up of additions and multiplications as the core arithmetic components, although, it is straightforward to extend the proposed method to other operation types. Each arithmetic operation v is associated with a word-length. For an adder, a word-length is a positive integer $b^A(v)$, representing the bit width of the core (integer) adder required in order to implement the multiple word-length addition (see [20] for a detailed discussion of multiple word-length addition). For a multiplier, a word-length is a pair $b^M(v)$, representing the two input bit widths of the core (integer) multiplier required. The elements of the pair are arranged such that the first element is always greater than or equal to the second element. Thus, for the remainder of this paper, it is sufficient to refer to a "10-b addition" or a "23 × 12-bit multiplication." Since this paper only considers these two operation types, for ease of notation, the remainder of this paper distinguishes between additions and multiplications by whether their word-length is an integer or a pair of integers.

These concepts are formalized in the definition of a dataflow graph given in Definition 1. The restriction on the ordering of the word-length tuple for multiplier nodes allows the algorithms that follow to implicitly take advantage of the commutativity of multiplication.

Definition 1: A *dataflow graph* $P(V, D)$ is a directed acyclic graph (DAG), representing the dataflow during a single iteration of an algorithm. The set V is in one-to-one correspondence with the set of operations. The directed edge set $D \subset V \times V$ is in correspondence with the flow of data from one operation to another. A TYPE function exists for elements of V (1). Each node $v \in V$ with $\mathrm{TYPE}(v) = \mathrm{MULT}$ has a word-length tuple $b^M(v) = (p_v, q_v) \in \mathbb{N}^2$ with $p_v \geq q_v$ representing the word-lengths of the two multiplier inputs, and each node $v \in V$ with $\mathrm{TYPE}(v) = \mathrm{ADD}$ has a single word-length $b^A(v) \in \mathbb{N}$ representing the word-length of the core adder

$$\mathrm{TYPE} : V \rightarrow \{\mathrm{ADD}, \mathrm{MULT}\}. \tag{1}$$

Example 1: A simple dataflow graph is illustrated in Fig. 2. The node set consists of five multiplications and four additions. Note that dependencies on external inputs are not shown, hence some two-input operations have fewer than two in-edges.

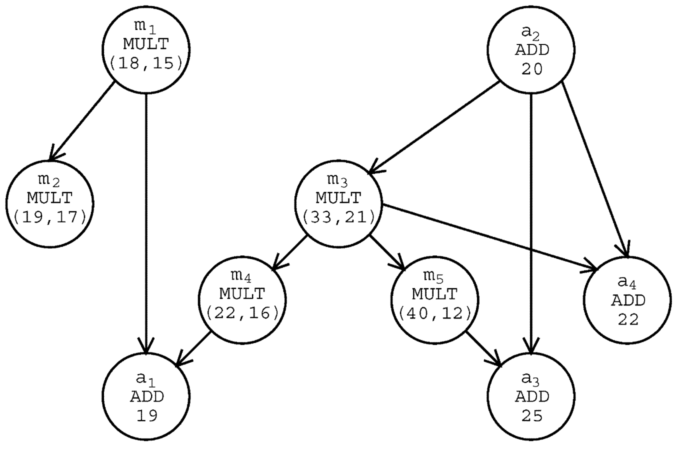


Fig. 2. Simple dataflow graph: nodes are labeled with their word-length(s) and type.

The *multiple word-length architectural systhesis* problem may now be defined in Problem 1. Note that the implementation in a structural hardware description language from the resulting information is not considered in this paper, as such techniques are well known [21].

Problem 1 (Multiple Word-Length Architectural Synthesis): Given a dataflow graph $P(V, D)$, and a specified maximum latency λ, determine:

- a set of resources and their associated word-lengths Y
- a mapping from operation to resource $\mathcal{R} : V \rightarrow Y$
- a mapping from operation to time-step $\mathcal{S} : V \rightarrow \mathbb{N} \cup \{0\}$

such that the area consumed by the set of resources is minimal, all data-dependencies are preserved, no resource executes more than one operation in each clock cycle, and the entire dataflow graph completes within λ cycles.

It is assumed that for any given target architecture, a core generator exists, capable of generating adder and multiplier cores of arbitrary precision. For each target architecture, it is necessary to construct an empirically derived function which determines the required number of clock cycles for each multicycle resource type. This construction has been performed for the Sonic reconfigurable computing architecture [22] for word-lengths up to 64-b, the results of which are given in (2). This relationship indicates how the number of fixed-period clock cycles on the Sonic computing platform varies with word-length (or word-length pair) r in practice. It is a simple matter to rederive such a formulation for any hardware implementation, and the proposed techniques do not require this relationship to have a fixed functional form

$$L(r) = \begin{cases} \lceil (p+q)/8 \rceil, & r = (p,q) \in \mathbb{N}^2 \\ 2, & r \in \mathbb{N}. \end{cases} \tag{2}$$

The cost function used for each resource is given in terms of its word-length in (3), where $\alpha \in \mathbb{R}$ is a technology-dependent constant representing the relative cost of adder and multiplier implementations. For the target technology used to collect the results of Section VI, $\alpha = 1$

$$\text{cost}(r) = \begin{cases} p \cdot q, & r = (p,q) \in \mathbb{N}^2 \\ \alpha \cdot r, & r \in \mathbb{N}. \end{cases} \tag{3}$$

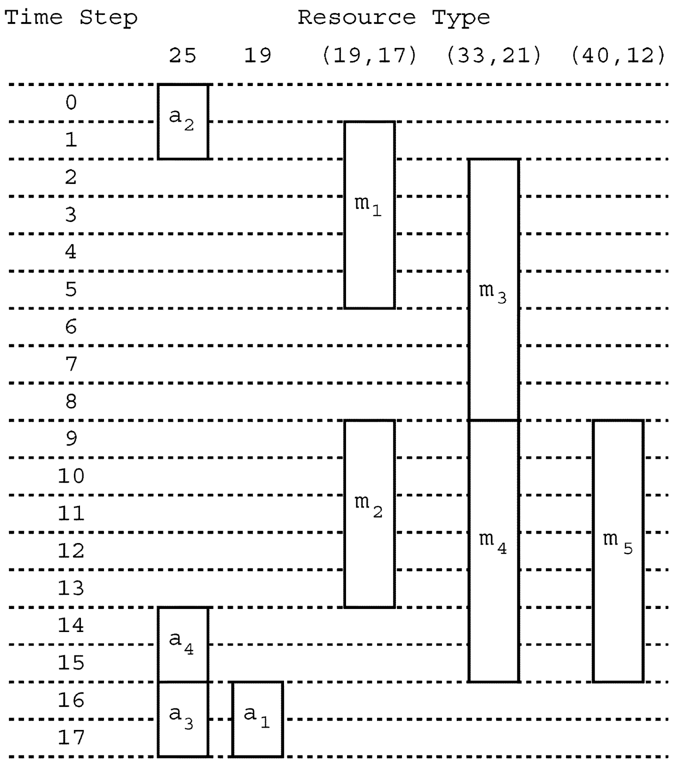


Fig. 3. An optimum scheduling, resource binding, and word-length selection for the dataflow graph illustrated in Fig. 2.

Example 2: As a motivational example, consider again the dataflow graph shown in Fig. 2. An area-optimal schedule, binding, and word-length selection for this dataflow graph is illustrated in Fig. 3 for the case $\alpha = 1$ and no operation pipelining. This resource allocation consists of two adders: one of 25-b and one of 19-b, and three multipliers: one is a 19×17-b multiplier, one a 33×21-b multiplier, and one a 40×12-b multiplier. The graphical matrix illustrates which resource is being used by which operation at which time step.

Note that in Fig. 3, resources can perform operations up to the word-length of the resource, even if implementation in a larger resource leads to a longer latency than a "tight-fitting" resource would require. For example, from (2) operation m_4 is implemented in a resource of latency 7 cycles, although, its word-length only requires a 22×16-bit multiplier which would take 5 cycles to complete. This "stretching" of operations that are not on the critical path can conceivably lead to significantly reduced area, by exposing possibilities for resource sharing.

In this paper, the notation $f(X' \subseteq X)$ for a function $f : X \rightarrow Y$ is used to denote the range $f(X' \subseteq X) = \{y \in Y : \exists x \in X' : f(x) = y\}$. The notation $\pi_i(x_1, x_2, \ldots, x_n) = x_i$ is used for the projection operator, and set subtraction is denoted by $\setminus$, i.e., $X \setminus Y = \{x \in X : x \notin Y\}$. A table summarizing the more specific notation used in the remainder of this paper is provided in Table I for reference.

IV. An ILP Solution

Integer linear programming (ILP) [23] has been used in high-level synthesis for some time [10], [24]–[27]. This section presents an extension to these ILP formulations in order to

TABLE I
NOTATION SUMMARY

	Problem specification
$P(V, D)$	the problem data flow graph (definition 1).
V	the set of vertices in a data flow graph, in one-to-one correspondence with the atomic operations.
V_m	the subset of vertices representing multiplication operations.
V_a	the subset of vertices representing addition operations.
D	the set of directed edges, in one-to-one correspondence with the data dependencies between operations.
$b^M(v)$	the word-length pair associated with a multiplication $v \in V$.
$b^A(v)$	the word-length associated with an addition $v \in V$.
λ	the user-defined upper bound on overall latency.
	Technology parameters
$L(r)$	the latency of a resource type $r \in \mathbb{N}$ (adder) or $r \in \mathbb{N}^2$ (multiplier).
$\text{cost}(r)$	the (area) cost of a resource type $r \in \mathbb{N}$ (adder) or $r \in \mathbb{N}^2$ (multiplier).
α	the relative cost of a n-bit adder compared to a $\sqrt{n} \times \sqrt{n}$-bit multiplier.
	Problem solution
Y	the set of resources used in an implementation.
$\mathcal{R}$	the function mapping operations to resources $\mathcal{R} : V \to Y$.
$\mathcal{S}$	the function mapping operations to scheduled time-steps $\mathcal{S} : V \to \mathbb{N} \cup \{0\}$.
	Notation for ILP model
$R^A(v)$	the set of adder types which could implement addition $v \in V_a$.
$R^M(v)$	the set of multiplier types which could implement multiplication $v \in V_m$.
$R(v)$	the set of resource types which could implement operation $v \in V$.
R	the set of possible resource types.
$I(r)$	upper bound on the number of instances required of resource type r.
$\ell(v)$	the latency of an operation $v \in V$.
$\ell_{\text{min}}(v)$	the minimum latency of an operation $v \in V$.
$\ell_{\text{max}}(v)$	the maximum latency of an operation $v \in V$.
$\text{ASAP}(v)$	the earliest execution step of operation $v \in V$, with all operations executing in minimum latency.
$\text{ALAP}(v)$	the latest execution step of operation $v \in V$, with all operations executing in minimum latency.
$T(v, r)$	the set of possible time steps at which operation v could start, if executing in a node of type r.
$T(v)$	the set of possible time steps at which operation v could start.
$b_{i,r}$	Boolean ILP variable, indicating whether instance i of resource type r is used by an operation.
$x_{v,t,i,r}$	Boolean ILP variable, indicating whether node v starts execution at time step t on instance i of resource type r.
	Notation for heuristic approach
β	the number of adders to allow in a solution per multiplier allowed.
$G(V \cup R, C \cup H)$	a word-length compatibility graph.
C	a set of directed edges, $(v_1, v_2) \in C \Leftrightarrow v_1$ completes execution before v_2 begins execution.
H	a set of undirected edges, $\{v, r\} \in H \Leftrightarrow$ operation $v \in V$ could be performed by resource type $r \in R$.
$G^+(V, C)$	the 'compatibility' subgraph of $G(V \cup R, C \cup H)$ with node set V and edge set C.
$\mathbf{c}$	bounds on the number (count) of resources of each type, used for list-scheduling.
S	the small cardinality 'scheduling subset' $S \subseteq R$.
$O(r)$	the set of operations performable by resource type r, *i.e.* $\{v \in V : \exists\{v, r\} \in H\}$.
$S(v)$	the subset of S which could implement operation v, *i.e.* $\{s \in S : \exists\{v, s\} \in H\}$.
V_c	the subset $V_c \subseteq V$ consisting of operations on the critical path.
V_b	the subset $V_b \subseteq V$ consisting of operations on the bound critical path.

solve Problem 1 [28]. Formulation as an ILP is useful from an analytical perspective, because it formalizes the problem and its constraints. In addition, for small problem instances, ILP solvers such as `lp_solve` [29] may be used to obtain globally optimum solutions to the synthesis problem. These optimum solutions are valuable references for comparison with heuristic approaches.

A. Resources, Instances, and Control Steps

Before presenting the ILP formulation of Problem 1, it is necessary to define certain quantities and notations, to be used in the following sections.

The starting point for the ILP approach is the dataflow graph $P(V, D)$ and target overall latency constraint λ.

Let $V_m \subseteq V$ be the subset of operations of type MULT and $V_a \subseteq V$ be the subset of operations of type ADD.

Any resource of the correct type, and large enough in word-length, can perform an operation. For example, a resource type (p, q) can perform any $p' \times q'$-bit multiplication, so long as $p' \leq p$ and $q' \leq q$. However, the search-space for area-efficient implementations may be trimmed significantly by observing that area-optimal resource bindings will only ever use the resource word-length that is just large enough to cover all operations assigned to that resource. For an adder to which operations $V' \subseteq V$ have been assigned, this corresponds to a word-length of $\max_{a \in V'} b^A(a)$. For a multiplier to which operations $V' \subseteq V$ have been assigned, this corresponds to a word-length of $(\max_{m \in V'} \pi_1(b^M(m)), \max_{m \in V'} \pi_2(b^M(m)))$, where $\pi_1(\cdot)$ and $\pi_2(\cdot)$ are the projection operators.

There are, therefore, only certain resource types which can arise from the optimal sharing of resources between operations. Let $R(v)$ denote the set of resource types which could implement the operation $v \in V$. Then, for addition and multiplication operations, $R(v)$ is given by (4) and (5), respectively. R denotes the set of all such resource types (6)

$$R(v) = \{p \in b^A(V_a) : p \geq b^A(v)\}, v \in V_a \quad (4)$$
$$\begin{aligned} R(v) = \{(p,q) \mid & \exists (p,b) \in b^M(V_m) \\ & \exists (c,q) \in b^M(V_m) : \\ & p \geq c \wedge q \geq b \wedge p \geq d \wedge q \geq e \\ & \text{where } (d,e) = b^M(v)\}, v \in V_m \end{aligned} \quad (5)$$
$$R = \bigcup_{v \in V} R(v). \quad (6)$$

Note that the formulation of $R(v)$ for multipliers allows resource-sharing between multiplications where the larger word-length multiplicand derives from one multiplication operation, whereas the smaller word-length multiplicand derives from another multiplication operation. For example, a 20×10-b multiplication can be shared with a 15×15-b multiplication through the use of a 20×15-b multiplier.

An upper bound $I(r)$ may be placed on the number of instances of each resource type that could arise. For an adder resource, there can be as many instances of an r-bit adder as there are r-bit addition operations (7). For a multiplier resource, each $p \times q$-bit resource can only arise due to resource sharing of a $p \times b$-bit and a $c \times q$-bit multiplication with $p \geq c$ and $q \geq b$. The number of these pairings is bounded by (8)

$$I(r) = |\{a \in V_a : b^A(a) = r\}|, \quad \text{for } r \in R(V_a) \quad (7)$$
$$\begin{aligned} I(p,q) = \min\{ & |\{m \in V_m : q \geq b \text{ where } (p,b) = b^M(m)\}| \\ & |\{m \in V_m : p \geq c \text{ where } (c,q) = b^M(m)\}|\} \\ & , \text{for } (p,q) \in R(V_m). \end{aligned} \quad (8)$$

From (2) it is possible to define the maximum latency $\ell_{\max}(v)$ and minimum latency $\ell_{\min}(v)$ of each operation $v \in V$ according to (9) and (10)

$$\ell_{\min}(v) = \min_{r \in R(v)} L(r) \quad (9)$$
$$\ell_{\max}(v) = \max_{r \in R(v)} L(r). \quad (10)$$

In order to bound the possible execution control steps of each operation, it is necessary to utilize as-soon-as-possible (ASAP) and as-late-as-possible (ALAP) scheduling [30]. Consider performing ASAP and ALAP scheduling of the operations, using a latency $\ell = \ell_{\min}$ for all operations. Let $\mathrm{ASAP}(v)$ denote the resulting ASAP control step for each operation $v \in V$. Similarly, let $\mathrm{ALAP}(v, \lambda)$ denote the ALAP control step for each operation $v \in V$ given a user-specified latency bound of λ and under the same operation latencies.

Each operation $v \in V$, executing on resource type $r \in R(v)$, can only start its execution during one of the time steps in the set $T(v,r)$ (11)

$$\begin{aligned} T(v,r) = \{ & t \in \mathbb{N} \cup \{0\} : t \geq \mathrm{ASAP}(v) \\ & \wedge t \leq \mathrm{ALAP}\ (v,\lambda) - L(r) + \ell_{\min}(v)\}. \end{aligned} \quad (11)$$

It will be useful to enumerate all possible start times $T(v)$ for each operation $v \in V$, according to (12), and indeed the complete set of time-steps T (13)

$$T(v) = \{t \mid \exists r \in R(v) : t \in T(v,r)\} \quad (12)$$
$$T = \bigcup_{v \in V} T(v). \quad (13)$$

B. ILP Formulation

Extending the notation used by Landwehr, *et al.* [27], we formulate the ILP as follows. Let $b_{i,r}$ define a Boolean variable with $b_{i,r} = 1$ iff instance number i of resource type r has at least one operation bound to it. This allows the objective function to be formulated in linear form (14)

$$\text{Minimize} \sum_{r \in R} \mathrm{cost}(r) \sum_{i=1}^{I(r)} b_{i,r} \quad (14)$$

In order to introduce the constraints, let $x_{v,t,i,r}$ be defined as in (15)

$$x_{v,t,i,r} = \begin{cases} 1, & \text{if operation } v \text{ is scheduled at time-step } t \\ & \quad \text{on the } i\text{th instance of resource type } r \\ 0, & \text{otherwise.} \end{cases} \quad (15)$$

The minimization is performed subject to three types of constraint. The first are the binding constraints, to ensure that each operation is executed on exactly one instance (16). The second are the resource constraints, to ensure that no resource instance is executing more than one operation at a time (17). The final set are the precedence constraints, to ensure that all operations obey the dependencies in the dataflow graph (18)

$$\forall v \in V, \sum_{r \in R(v)} \sum_{i=1}^{I(r)} \sum_{t \in T(v,r)} x_{v,t,i,r} = 1 \quad (16)$$
$$\begin{aligned} & \forall t \in T, \quad \forall r \in R, \forall i \in \{1, \ldots, I(r)\} \\ & \sum_{v \in V : r \in R(v)} \left(\sum_{t_1 \in \{t, \ldots, t+L(r)-1\} \cap T(v,r)} x_{v,t_1,i,r} \right) \leq b_{i,r} \end{aligned} \quad (17)$$
$$\begin{aligned} & \forall (v_1, v_2) \in D, \quad \forall t \in T(v_2) \cap \{\mathrm{ASAP}(v_1) \\ & \quad + \ell_{\min}(v_1) - 1, \ldots, \mathrm{ALAP}(v_1) + \ell_{\max}(v_1) - 1\} \\ & \sum_{r \in R(v_2)} \sum_{i=1}^{I(r)} \sum_{t_2 \in T(v_2,r) : t_2 \leq t} x_{v_2,t_2,i,r} \\ & \quad + \sum_{r \in R(v_1)} \sum_{i=1}^{I(r)} \sum_{t_1 \in T(v_1,r) : t_1 > t - L(r)} x_{v_1,t_1,i,r} \leq 1. \end{aligned} \quad (18)$$

Example 3: Recall the simple dataflow graph of Fig. 2. The ILP formulation for this dataflow graph contains 164 variables and 166 constraints for $\lambda = 18$, the lowest achievable latency. Fig. 3 illustrates an optimal solution corresponding to this latency constraint, which has the following optimization variables taking the value 1: $x_{a_2,0,1,25}$, $x_{a_4,14,1,25}$, $x_{a_3,16,1,25}$, $x_{a_1,16,1,19}$, $x_{m_1,1,1,(19,17)}$, $x_{m_2,9,1,(19,17)}$, $x_{m_3,2,1,(33,21)}$, $x_{m_4,9,1,(33,21)}$, $x_{m_5,9,1,(40,12)}$, $b_{1,25}$, $b_{1,19}$, $b_{1,(19,17)}$, $b_{1,(33,21)}$, $b_{1,(40,12)}$. All other variables are equal to zero.

After solution, the values of the ILP optimization variables $x_{v,t,i,r}$ and $b_{i,r}$ encode a solution to Problem 1 given in (19)–(21)

$$Y = \{(i,r) : b_{i,r} = 1\} \tag{19}$$

$$\mathcal{R}(v) = \sum_{r \in R(v)} \sum_{i=1}^{I(r)} \sum_{t \in T(v,r)} (i,r) \cdot x_{v,t,i,r}, \quad \text{for} \quad v \in V \tag{20}$$

$$\mathcal{S}(v) = \sum_{r \in R(v)} \sum_{i=1}^{I(r)} \sum_{t \in T(v,r)} t \cdot x_{v,t,i,r}, \quad \text{for} \quad v \in V. \tag{21}$$

Optimum solutions can only be found for relatively small examples using ILP, due to the large number of variables and constraints. Moreover, the number of variables and constraints increases linearly with the relaxation of λ. These drawbacks have motivated the search for efficient heuristic solutions to this problem, as presented in the following section.

V. Proposed Heuristic

This section presents an heuristic approach to Problem 1 [31]. The proposed algorithm iteratively refines word-length information while using resource-constrained scheduling and a combined resource binding and word-length selection procedure, in order to steer the solution toward feasibility with respect to the user-specified latency constraint.

The following description starts with an overview of the heuristic in Section V-A and a description of the word-length compatibility graph in Section V-B. Each of the algorithm steps is then described: calculation of resource bounds (Section V-C) and latency bounds (Section V-D), scheduling using incomplete word-length information (Section V-E), combined binding and word-length selection (Section V-F) and word-length refinement (Section V-G).

A. Overview

A high-level overview of the proposed heuristic is shown in Algorithm 1, and illustrated diagramatically in Fig. 4. The algorithm arrives at a solution through an iterative refinement of word-length information in order to reach the user-specified latency target λ. An initial solution is constructed by allowing each operation to be scheduled using the longest latency of all resources which could perform that operation. Scheduling in this manner guarantees that any resource binding will not violate the schedule, and it is expected that a great deal of resource sharing can be achieved. However, using the upper bound latency of each operation may result in a violation of the overall latency target λ. At each iteration of Algorithm 1,

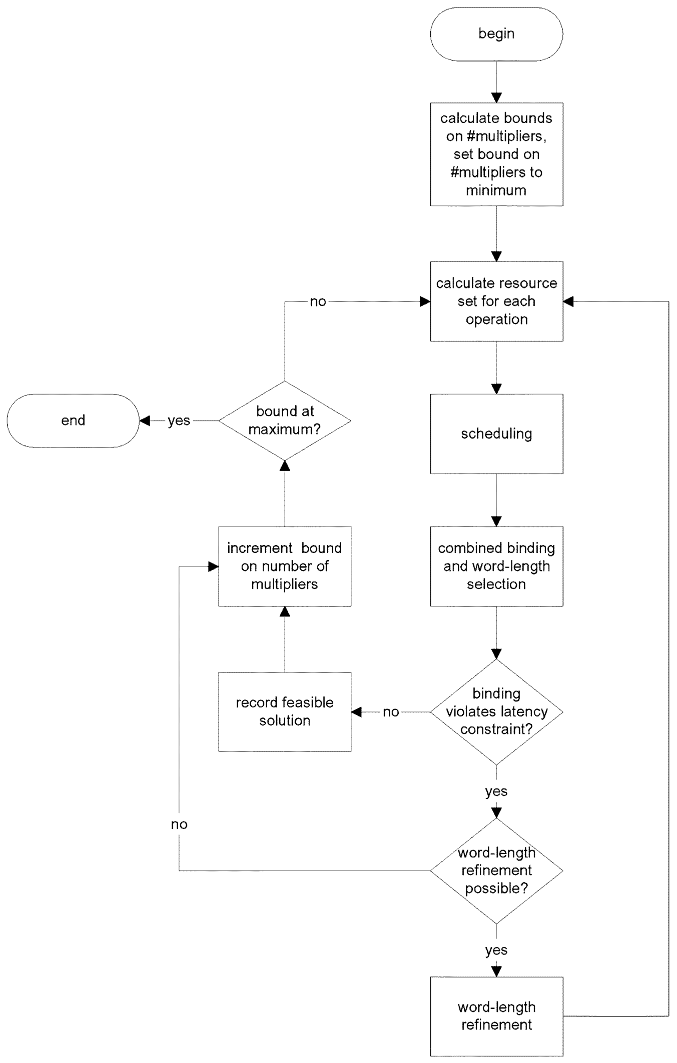

begin
calculate bounds on #multipliers, set bound on #multipliers to minimum
calculate resource set for each operation
scheduling
combined binding and word-length selection
binding violates latency constraint?
record feasible solution
increment bound on number of multipliers
bound at maximum?
end
word-length refinement possible?
word-length refinement
yes
no

Fig. 4. Flowchart of the proposed heuristic.

these upper bounds are refined by selecting an operation and reducing its upper-bound latency, and hence, the range of different word-length resources which could implement that operation.

In most implementation cases, the area consumed by a multiplier is significantly larger than that consumed by an adder. It is for this reason that Algorithm 1 calculates bounds on the number of *multipliers* required and constructs the solution accordingly, searching for solutions with between $m_{\min}$ and $m_{\max}$ multipliers. The corresponding bound on the number of adders is determined through a simple scaling with a factor β. By performing the optimization in this manner, the bounds on the number of each resource type need not be optimized individually, leading to an improvement in algorithm execution time at the possible penalty of a few extra adders in the resulting architecture. For our current implementation, we use the empirically derived $\beta = 4$. Of course, if the set of available resource types were expanded beyond adders and multipliers in a way that destroys this imbalance in implementation area, this approach could no longer be used. In the most general case, it would be necessary to extend Algorithm 1, introducing a new loop similar to steps 1–2 for each resource type.

```
Algorithm 1 (ArchSynth)
Input: A dataflow graph $P(V, D)$ and a latency
    constraint $\lambda$
Output: Scheduling, binding, and word-length
    for each operation
Objective: Minimize the area of the resulting
    implementation
begin
  1. calculate $m_{\min}$ and $m_{\max}$, bounds on #multipliers
  2. for $n \in \{m_{\min}, \dots, m_{\max}\}$ do
  2.1    do
  2.1.1    calculate the resource set covering each operation
  2.1.2    search for upper-bounds on latency of each
             operation
  2.1.3    search for a feasible schedule using latency upper
             bounds and no more than $n$ mults and
             $\beta \cdot n$ adds
  2.1.4    perform combined binding and word-length
             selection
  2.1.5    if binding violates the latency constraint $\lambda$ do
             try to refine operation word-length information
           else do record this feasible solution
           end if
         while refinement (step 2.1.5) is possible
       end for
end
```

The calculation of resource bounds (step 1) and each of the steps 2.1.1–2.1.5 will be discussed in detail in the following sections. In addition, failure conditions can arise in finding upper-bounds (step 2.1.2), deadlocks in scheduling with incomplete word-length information (step 2.1.3), and refining upper bounds (step 2.1.5). Each of these cases will also be considered in the following sections.

B. Word-length Compatibility Graph

A fundamental model that underlies the majority of the proposed heuristic is the *word-length compatibility graph*.

Definition 2: A *word-length compatibility graph* $G(V \cup R, C \cup H)$ is a representation of information about the *type* of each operation, the *word-length* of each operation, and schedule-derived information on time-compatibility between operation pairs. The vertex set can be partitioned into two subsets V and R, where V denotes the set of operations, and R denotes the set of resource types (6). The set of edges can also be partitioned into two subsets C and H. H is a set of undirected edges $\{v, r\}$, where $v \in V$ and $r \in R$, representing the information that operation v could be performed by resource type r. C is a set of directed edges (v_1, v_2), where $v_1, v_2 \in V$, representing the information that operation v_1 is scheduled to complete execution before operation v_2 is scheduled to start execution.

The scheduling algorithm to be described in Section V-E utilizes the operation—resource-type compatibility encoded in the edge set H and implicitly creates the edge set C in the process. The combined binding and word-length selection algorithm to be described in Section V-F utilizes both the time-compatibility edge set C and the operation resource-type compatibility set H.

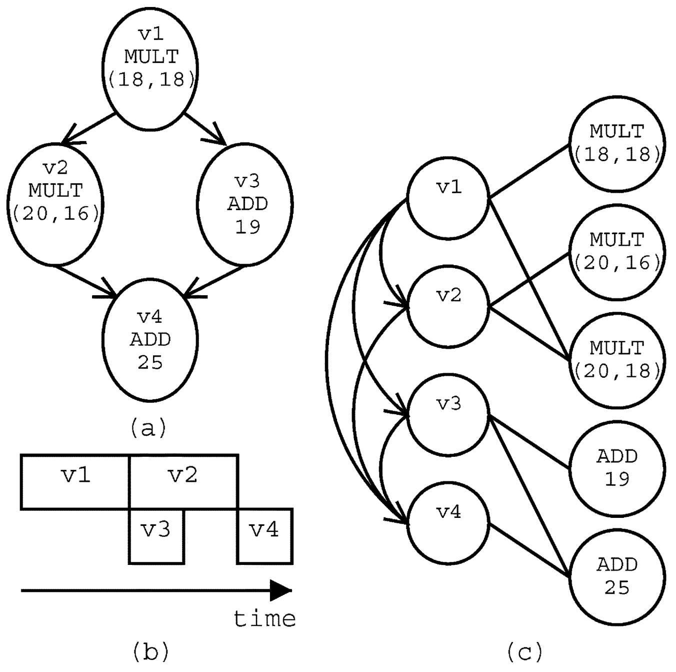


Fig. 5. Word-length compatibility graph. (a) Sequencing graph. (b) Schedule. (c) Word-length compatibility graph.

It is important to note that the edge set C has been chosen to ensure that subgraph $G^+(V, C)$ exhibits a transitive orientation [32], since if $v_1 \in V$ finishes before v_2 starts, and v_2 finishes before v_3 starts, it follows that v_1 finishes before v_3 starts. This orientation will be used in Section V-F to aid fast resource binding. Note also that the set C of directed edges in the graph need not be constructed explicitly in a software implementation, but can be inferred from the scheduled times of the operations.

Example 4: A simple word-length compatibility graph is shown in Fig. 5(c), corresponding to the dataflow graph and schedule shown in Fig. 5(a) and (b), respectively.

The initial word-length compatibility graph is constructed in the following manner: A resource set R is constructed following (4)–(6). Edge set H is initialized to the set $\{\{v \in V, r \in R\} : r \in R(v)\}$. C is initialized to the empty set. As Algorithm 1 executes, word-length refinement will result in the deletion of edges from the set H.

C. Resource Bounds

The first stage of the heuristic is to find the smallest and largest sensible upper bounds to place on the number of multipliers required. These values are obtained from a study of how the iteration latency achieved by list-scheduling decreases with the number of multipliers allowed.

A standard resource-constrained list scheduling algorithm [10], [33] can be used to heuristically obtain these bounds. Standard ALAP urgency-based list scheduling with a bound $\mathbf{c} \in \mathbb{N}^2$ on the number of resources of each type is used. $\mathbf{c}$ is a 2-vector of integer elements, the first corresponding to the bound on the number of multipliers, and the second corresponding to the bound on the number of adders.

The bounds $m_{\min}$ and $m_{\max}$ used in Algorithm 1 can now be defined, assuming that the given latency bound is

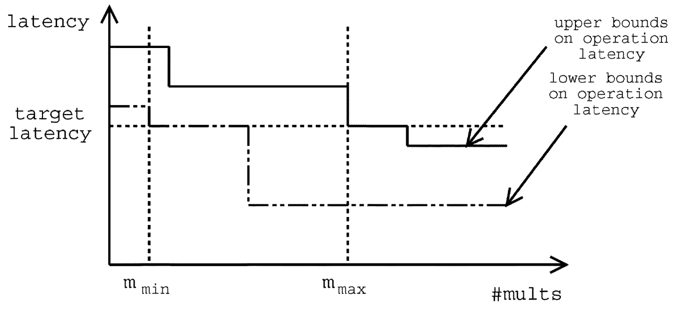


Fig. 6. Calculating bounds on the number of multipliers $m_{\min}$ and $m_{\max}$.

realizable. Bound $m_{\min}$ is the smallest value such that the list-scheduled latency is within the constraint for all schedules with $\mathbf{c} \geq (m_{\min}, \beta m_{\min})$ and $\ell(v) = \ell_{\min}(v)$ for all $v \in V$. Similarly bound $m_{\max}$ is the smallest value such that the list-scheduled latency is within the constraint for all schedules with $\mathbf{c} \geq (m_{\max}, \beta m_{\max})$ and $\ell(v) = \ell_{\max}(v)$ for all $v \in V$. For tight latency constraints there may be no such $m_{\max}$ value, in which case $m_{\max}$ is set to $|V_m|$. In each of these cases, a binary search is used to determine the bounds.

The rationale behind these bounds is the following. If an algorithm cannot be scheduled to meet the imposed latency constraint under a resource constraint $\mathbf{c}$, even when all operations have their minimum possible latency, then the algorithm cannot meet this latency constraint for any $\mathbf{c}' \leq \mathbf{c}$. This provides a lower bound on the number of each type of resource required. Similarly, if the imposed timing constraint can be met under a resource constraint $\mathbf{c}$, even when all operations have their maximum latency, then the algorithm can meet this latency constraint for all $\mathbf{c}' \geq \mathbf{c}$. This provides an upper bound on the number of each type of resource required. Fig. 6 illustrates the way in which the achieved latency varies with the bound on the number of multipliers supplied to the list-scheduler.

Example 5: An example derivation of resource bounds is illustrated in Fig. 7. Fig. 7(a) illustrates a simple dataflow graph, with corresponding initial word-length compatibility graph shown in Fig. 7(b). The latency curves resulting from list scheduling for different resource bounds are plotted in Fig. 7(c). The points corresponding to the minimum possible number and maximum necessary number of multiplier resources have been highlighted.

D. Latency Bounds

Before entering the main refinement loop in Algorithm 1, it is possible to significantly reduce the number of iterations by pruning the operation latency search space. If it is not possible to list-schedule a dataflow graph $P(V, D)$ when all operations $V \setminus \{v\}$ have their minimum possible latency while operation v has latency $\ell(v)$, then it is assumed that a feasible schedule will equally not be possible if operation v has any latency $\ell'(v) > \ell(v)$. This allows the edge set H of an initially constructed word-length compatibility graph to be refined. The approach is illustrated in Algorithm 2. After first checking that a feasible solution exists in steps 1–2 (by trying to schedule when all operations have minimum latency), the algorithm proceeds by deleting edges from the set H. Each operation node v is tested

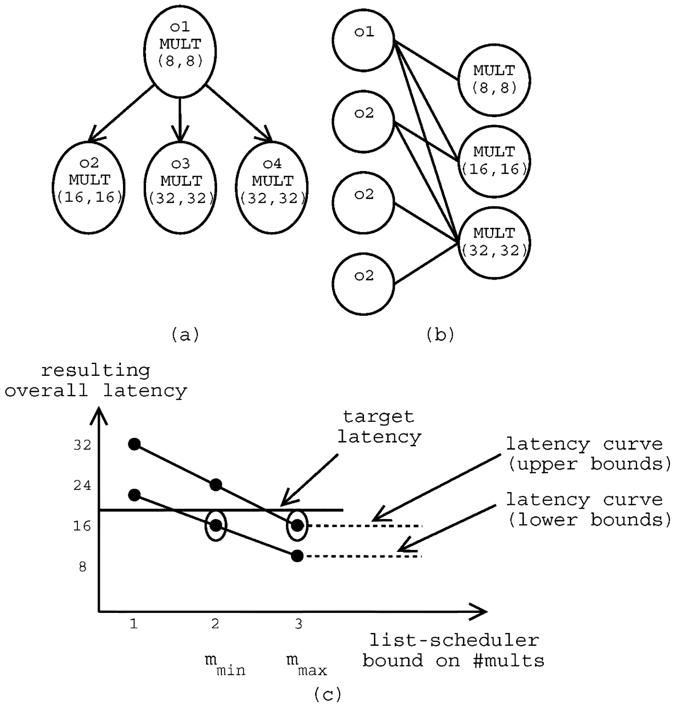


Fig. 7. Example multiplier bounds.

in turn (step 3) to find the maximum latency the operation could have (out of those corresponding to resource types which could implement that operation) while not violating the overall latency constraint. Once this value is found (step 3.1), any edges connecting node v to a resource type with a greater latency are removed from the word-length compatibility graph (step 3.2).

```
Algorithm 2 LatencyBounds
Input: A dataflow graph $P(V, D)$, initial
    word-length compatibility graph $G(V \cup R, C \cup H)$,
    resource constraint vector $\mathbf{c}$ and latency constraint $\lambda$
Output: A refined word-length compatibility graph
Objective: Minimize the set of resources to which
    each operation could be bound
begin
  1.  $\ell(v) \leftarrow \min_{\{v,r\} \in H} L(r)$ for all $v \in V$
  2.  if ListSchedule$(P, \ell, \mathbf{c})$ returns a schedule violating
        latency constraint $\lambda$ do
        return failure case
      end if
  3.  foreach $v \in V$ do
  3.1   Search for maximum $\ell(v) \in \{L(r) : \exists \{v,r\} \in H\}$
        such that ListSchedule $(P, \ell, \mathbf{c})$ returns a
        schedule satisfying latency constraint $\lambda$
  3.2   $H \leftarrow H \setminus \{\{v,r\} \in H : L(r) > \ell(v)\}$
  4.    return $\ell(v)$ to its original value $\ell(v) \leftarrow \min_{\{v,r\} \in H} L(r)$
      end foreach
end
```

Using Algorithm 2, an upper bound on the latency of each operation can be established. Once this has been done, the set of edges H of the word-length compatibility graph, representing the possible design decisions, has been pruned.

E. Scheduling With Incomplete Word-length Information

At the time of scheduling in Algorithm 1, the word-length of each operation may not be fixed. Indeed, each operation could be implemented using any resource type to which the operation is linked by an edge in the word-length compatibility graph. The scheduling problem, therefore, has incompletely defined constraints [34], and a technique must be developed to incorporate these constraints into directing the search for a solution. The following paragraphs illustrate the need for such a technique, before introducing the proposed solution.

Traditional resource-constrained scheduling techniques such as force-directed list scheduling [35], require resource constraints to be expressed in terms of a bound on the number of resources of each type. During standard list scheduling, these constraints are tested at each time step before deciding whether to schedule a new operation. The constraints may be formally expressed as follows. Let $e_{v,t}$ be defined as in (22). Thus, $e_{v,t} = 1$ iff operation v is *executing* during time-step t. Given a set of control steps T (13), a set of operations V, and the maximum number of resources c_k of type k, the traditional resource constraints may be expressed as (23)

$$e_{v,t} = \begin{cases} 1, & \text{if } (\mathcal{S}(v) \le t) \wedge (\mathcal{S}(v) + \ell(v) > t) \\ 0, & \text{otherwise} \end{cases} \quad (22)$$

$$\forall k \in \{\text{ADD}, \text{MULT}\}, \max_{t \in T} \sum_{v \in V : \text{TYPE}(v) = k} e_{v,t} \le a_k. \quad (23)$$

In the case of multiple word-length systems, these constraints tend to be too relaxed to guarantee that no more than a_k resources of type k will be used by the given schedule.

Example 6: Consider the schedules and corresponding word-length compatibility graphs shown in Fig. 8. Such graphs could arise during the execution of Algorithm 1. Fig. 8(a) has fully defined word-length information for each operation. It is clear that even though v_1, v_2, and v_3 are all multiplications and do not overlap in execution, three distinct multiplier resources will still be required for their implementation. However, the standard scheduling constraint (23) would be satisfiable for $a_{\text{MULT}} = 1$.

Fig. 8(b) has an incomplete specification (there is at least one operation that could be implemented in more than one possible resource type). However, a 32×32-bit multiplier could conceivably implement every operation. Thus, it is possible to implement the entire system using a single multiplier resource.

Fig. 8(c) illustrates a general case, corresponding to the deletion of a single edge from the word-length compatibility graph of Fig. 8(b). Using traditional methods, it is unclear in this case how to incorporate such constraints into the search for an appropriate schedule.

These examples demonstrate that a more sophisticated approach to scheduling is required to take word-length information into account. In general, it is necessary to consider the *incomplete* word-length specification provided by an edge set H.

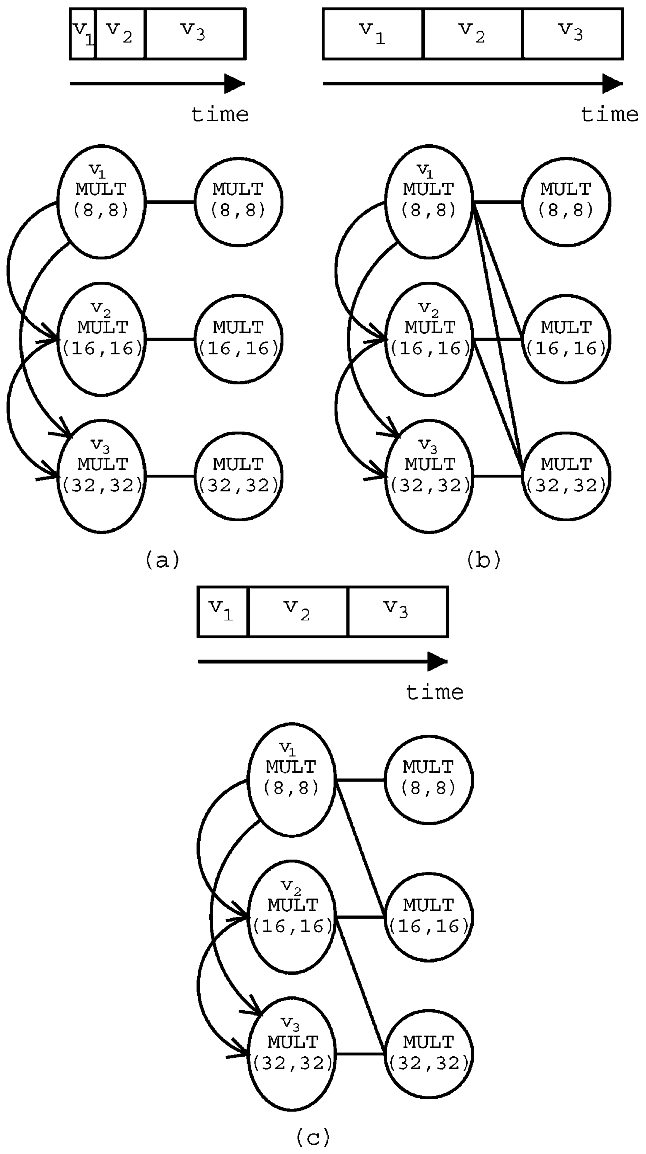
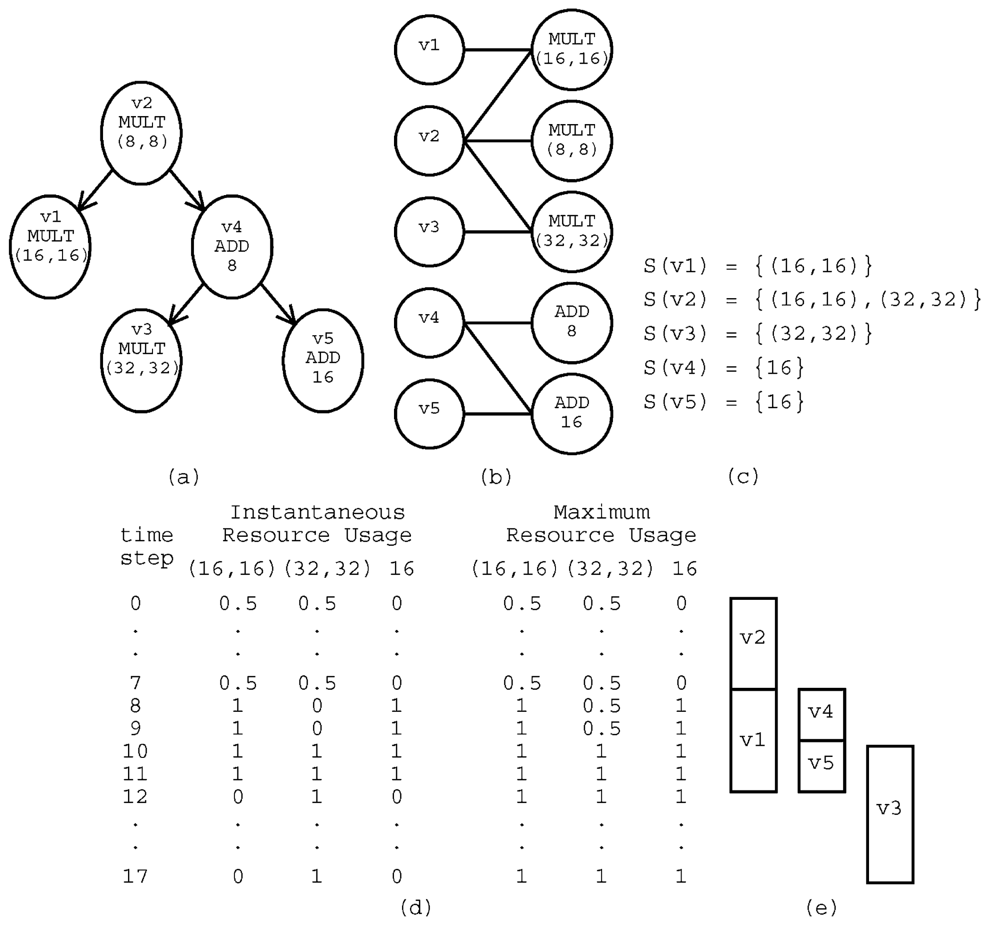


Fig. 8. Some schedules and word-length compatibility graphs. (a) Fully specified word-length information. (b) Incomplete specification (full sharing possible). (c) Incomplete specification (general case).

The scheduling algorithm proposed is a modification of a standard list scheduling [10]. The modification lies in the resource constraint calculation. Before any scheduling takes place, a small cardinality subset $S \subseteq R$ is found such that $\forall v \in V, \exists s \in S : \{v, s\} \in H$. Conceivably, a resource binding could consist only of resource of types represented in S. Define $O(r)$ to be the set of operations performable by resource type $r \in R$, i.e., $O(r) = \{v \in V : \exists \{v, r\} \in H\}$. Similarly, let $S(v)$ denote the subset of resource types in S which could implement operation $v \in V$, i.e., $S(v) = \{s \in S : \exists \{v, s\} \in H\}$. Then the proposed constraint function to be used in the algorithm can be expressed as in (24)

$$\forall k \in \{\text{ADD}, \text{MULT}\}$$
$$\sum_{s \in S : \text{TYPE}(s) = k} \max_{t \in T} \left\{ \sum_{v \in O(r)} e_{v,t} |S(v)|^{-1} \right\} \le a_k. \quad (24)$$



S(v1) = {(16,16)}
S(v2) = {(16,16),(32,32)}
S(v3) = {(32,32)}
S(v4) = {16}
S(v5) = {16}

time step	Instantaneous Resource Usage			Maximum Resource Usage		
	(16,16)	(32,32)	16	(16,16)	(32,32)	16
0	0.5	0.5	0	0.5	0.5	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
7	0.5	0.5	0	0.5	0.5	0
8	1	0	1	1	0.5	1
9	1	0	1	1	0.5	1
10	1	1	1	1	1	1
11	1	1	1	1	1	1
12	0	1	0	1	1	1
.	.	.	.	.	.	.
.	.	.	.	.	.	.
17	0	1	0	1	1	1

Fig. 9. Example of scheduling a dataflow graph with incomplete word-length specifications. (a) Sequencing graph. (b) Word-length compatibility graph. (c) Sets $S(v)$. (d) Resource usage matrix. (e) Schedule.

This is an heuristic measure with the following justification. First, (24) is at least as strict as (23), which is a special case of the former under the condition $|\mathrm{TYPE}(V)| = |S|$, the smallest sized S possible. This represents the case where each multiplication could be performed by a single large multiplier, and each addition could be performed by a single large adder. As the possibilities for the implementation of each operation are reduced during execution of Algorithm 1, the balance on the left-hand side of (24) shifts from the $\max_{t\in T}$ to the $\sum_{s\in S}$ to reflect the stricter constraints. The small cardinality S is used in order to relax the constraint as much as possible, since any two operations in $O(s)$ could *possibly* be eventually bound to the same resource. Operations belonging to more than one $O(s)$, i.e., those $v \in V$ with $|S(v)| > 1$, are accounted for by "sharing" equally their usage between each of the elements $S(v)$.

Algorithm 3 illustrates this scheduling based on incomplete information. Two auxilliary data structures are used in the algorithm to keep track of the scheduling constraint (24), $\mathrm{usage}(s)$ and $\mathrm{maxusage}(s)$. Respectively, these keep track of the instantaneous and peak usage of resource type $s \in S$. The algorithm starts by setting the latency $\ell(v)$ of each operation to its maximum (step 1). After doing so, a standard ALAP-based urgency measure [10] is calculated for each node (step 2), and the time step index is initialized (step 3). The set S described above (step 4), and its related function $S(v)$ (step 5), to be used in the scheduling constraint (24) are then calculated. Step 6 ensures that the peak usage $\mathrm{maxusage}(s)$ for each element of that set is initialized. The algorithm then enters its main scheduling loop, with one iteration per time step (step 7).

At the start of each iteration, the instantaneous usage of resources is initialized (step 7.1), the ready-list is calculated (step 7.2), and the prime candidate for scheduling is selected (step 7.3). The algorithm then enters a secondary loop (step 7.4), which tries to schedule this and any other operation of the same type. The current left-hand side of (24) is first calculated (step 7.4.1), and then updated (step 7.4.2) for any $s \in S$ for which scheduling in the current control step would use more than the current peak usage for that s. If the updated (24) is still satisfied, then the scheduling of the operation is accepted (step 7.4.3), and the peak usage is updated (step 7.4.4). Deadlocks, to be discussed below, may occur in the scheduling process. These are detected by step 7.5.

Example 7: An example execution of Algorithm 3 is shown in Fig. 9. The dataflow graph and word-length compatibility graph are shown in Fig. 9(a) and (b), respectively. Fig. 9(c) enumerates the $S(v)$ sets for this example, and Fig. 9(d) shows how the usage and maxusage variables evolve as the algorithm executes for $a_{\mathrm{ADD}} = 1, a_{\mathrm{MULT}} = 2$. The resulting schedule is shown in Fig. 9(e), and could be resource-bound as a single 16-b adder together with both a 16×16-b multiplier and a 32×32-b multiplier. Details on how such a resource binding can be found for general graphs are discussed in the following section.

```
Algorithm 3 (IncompSched)
**Input:** A dataflow graph $P(V, D)$, word-length
      compatibility graph $G(V \cup R, C \cup H)$
      and maximum number **c** of each resource type
**Output:** A schedule $\mathcal{S} : V \rightarrow \mathbb{N} \cup \{0\}$ for each $v \in V$
**Objective:** Minimize the schedule length
**begin**
  1. $\ell(v) = \max_{\{v,r\} \in H} L(r)$
  2. Determine the 'urgency' of each operation $v \in V$
        through ALAP scheduling
  3. $t \leftarrow 0$
  4. Find $S \subseteq R$ of smallest size such that
        $\forall v \in V, \exists s \in S : \{v, s\} \in H$
  5. Let $S(v) = \{s \in S : \exists \{v, s\} \in H\}$
  6. $\mathrm{maxusage}(s) \leftarrow 0$ for all $s \in S$
  7. **do**
  7.1    $\mathrm{usage}(s) \leftarrow 0$ for all $s \in S$
  7.2    $E \leftarrow$ the list-schedule 'ready list' [10],
           sorted by urgency
  7.3    $e \leftarrow$ most urgent element of $E$
  7.4    **do**
  7.4.1      $\mathrm{total} \leftarrow \sum_{s \in S(\{v \in V : \mathrm{TYPE}(v) = \mathrm{TYPE}(e)\})} \mathrm{maxusage}(s)$
  7.4.2      **foreach** $s \in S(e)$:
               $\mathrm{usage}(s) + |S(e)|^{-1} > \mathrm{maxusage}(s)$ **do**
               $\mathrm{total} \leftarrow \mathrm{total} - \mathrm{maxusage}(s) + \mathrm{usage}(s) + |S(e)|^{-1}$
             **end foreach**
  7.4.3      **if** $\mathrm{total} \leq a_{\mathrm{TYPE}(e)}$ **do**
               $\mathcal{S}(e) \leftarrow t$
               $\mathrm{usage}(s) \leftarrow \mathrm{usage}(s) + |S(e)|^{-1}$
  7.4.4        **foreach** $s \in S(e)$ :
                 $\mathrm{usage}(s) > \mathrm{maxusage}(s)$ **do**
                 $\mathrm{maxusage}(s) \leftarrow \mathrm{usage}(s)$
               **end foreach**
             **end if**
  7.4.5      $e \leftarrow$ next most urgent element of $E$,
               if one exists
           **while** such an $e$ exists $\wedge \exists k \in \{\mathrm{ADD}, \mathrm{MULT}\}$ :
               $\sum_{s \in S : \mathrm{TYPE}(s) = k} \mathrm{maxusage}(s) < a_i$
           $t \leftarrow t + 1$
  7.5    **if** deadlock detected **do**
           **return** failure case
         **end if**
       **while** there remains at least one unscheduled operation
**end**
```

There are a number of significant differences between standard list scheduling and Algorithm 3. Information on resource usage is accumulated over control steps in Algorithm 3, rather than each step being constraint, function, and independent of each other step. There are two related drawbacks from this: First, it is possible for the proposed list-scheduler to deadlock, by scheduling operations belonging to $O(s_1)$ for the some $s_1 \in S$ early in the schedule and then having no remaining resources to schedule operations belonging to $O(s_2)$ for some $s_2 \in S, s_2 \neq s_1$ later in the schedule. Such deadlocks can be easily detected: if all operations have finished by the current time-step and yet no operation has been scheduled by the end of that time-step, deadlock has occurred. Second, although the scheduler may not deadlock, greedy allocation of parallel $O(s_1)$ operations early-on in the schedule may cause schedules of longer than optimal latency. Thus, Algorithm 3 has a greedy bias toward earlier time steps.

The subset $S \subseteq R$ used by Algorithm 3 can be found easily through Algorithm 4. Starting from an empty set S, this algorithm simply iteratively adds those resource types from the set H which could implement the most (thus far, uncovered) operations.

```
Algorithm 4
**Input:** Word-length compatibility graph $G(V \cup R, C \cup H)$
**Output:** Set $S \subseteq R$ required by Algorithm 3
**Objective:** Minimize the cardinality of set $S$
**begin**
  $S \leftarrow \emptyset$
  $V' \leftarrow V$
  $H' \leftarrow H$
  **while** $|V'| > 0$ **do**
      Find $r \in R$ such that $|\{\{v, r\} \in H'\}|$ is maximum
      $S \leftarrow S \cup \{r\}$
      $V' \leftarrow V' \setminus \{v \in V : \exists \{v, r\} \in H\}$
      $H' \leftarrow H' \setminus \{\{v, r\} \in H\}$
  **end while**
**end**
```

F. Combined Binding and Word-length Selection

Once a dataflow graph has been scheduled, resource binding and word-length selection can be performed. No resource binding can violate the scheduling latency constraint, since latency upper bounds have been used when performing the scheduling (Algorithm 1). The *combined binding and word-length selection* problem (Problem 2) is, therefore, a subproblem of Problem 1.

Problem 2 (Combined Binding and Word-length Selection): Given a scheduled word-length compatibility graph $G(V \cup R, C \cup H)$, the *combined binding and word-length selection* problem is to select a set of resources and their associated word-lengths Y and a mapping from operation to resource $\mathcal{R} : V \rightarrow Y$ such that the area consumed by the set of resources is minimal and no resource executes more than one operation in each clock cycle.

Definition 3: Clique k is a *maximal clique* of graph G iff k is not a subgraph of any other clique of graph G.

Definition 4: Clique $k(V_k, E_k)$ is a *maximum clique* of graph G iff there is no clique $k'(v'_k, e'_k)$ of G with $|v'_k| > |v_k|$.

Definition 5: Clique $k(v', c')$ of the subgraph $G^+(V, C)$ of word-length compatibility graph $G(V \cup R, C \cup H)$ is a *feasible clique* iff $\exists r \in R : \forall v \in v', \{v, r\} \in H$.

The *combined binding and word-length selection* problem is approached by partitioning the subgraph $G^+(V, C)$ into a set K of feasible cliques. The feasibility constraint captures the requirement that there must be a single resource capable of performing all operations in the clique. The cost of this resource binding is then given by (25)

$$\sum_{k(v_k,e_k)\in K} \min_{r\in R:\forall v\in v_k,\exists\{v,r\}\in H} \mathrm{cost}(r). \tag{25}$$

This problem is a special case of the set-covering or weighted unate covering problem.

Problem 3 (SET COVERING, [36]): Consider a set of sets $U = \{u_1, u_2, \ldots, u_n\}$ and associated positive costs $c_{u_1}, c_{u_2}, \ldots, c_{u_n}$. Let $I = \bigcup_{u\in U} u$. A subset $U' \subseteq U$ is a *cover* iff $\bigcup_{u\in U'} u = I$. The *cost* of this cover is $\sum_{u\in U'} c_u$. The problem is to find a cover of minimum cost.

The combined binding and word-length selection problem can be cast as a set covering problem in the following manner. Let U denote the set of node sets of all feasible cliques in the graph $G^+(V,C)$ (26). The cost c_k associated with clique $k \in U$ is given by the corresponding term in the summation of (25)

$$U = \{V' \subseteq V : V' \text{induces a feasible clique in } G^+(V,C)\}. \tag{26}$$

The proposed approach is to extend a known heuristic for solving the unate covering problem [36] to the combined resource binding and word-length selection application. In order to present the proposed extensions to this simple heuristic, it is first reviewed below.

Intuitively, for a greedy algorithm it becomes more desirable to include a set u_j in the cover U' as the number of elements covered by u_j and not already covered by any previously chosen set increases. This is tempered by the cost of set u_j and, thus, the ratio of these two quantities forms an appropriate measure of desirability. This observation leads to the following heuristic proposed in [36]

```
Algorithm 5 (ChvatalHeur)
Input: An instance of the set covering problem (Problem 3)
Output: A cover $U' \subseteq U$
Objective: Minimize the cost of the cover $U'$
begin
  $U' \leftarrow \emptyset$
  while $\exists i : |u_i| \neq 0$ do
    find $i$ such that $|u_i|/c_i$ is maximum
    $U' \leftarrow U' \cup \{u_i\}$
    foreach $j \in \{1, \ldots, n\}$ do
      $u_j \leftarrow u_j \setminus u_i$
    end foreach
  end while
end
```

The first and simplest extension to [36], is to include some compensation for the greedy nature of the original algorithm. If a clique is chosen during one iteration of the algorithm, it is checked whether this clique could be extended to cover all operations covered by any of the cliques chosen at previous iterations. If such an extension is possible, the selected clique is grown accordingly and the previously chosen clique is deleted from the cover set.

The more important distinction is that the set U is never calculated, since its size can be very large (exponential in $|V|$). Instead, an implicit approach is used, which is polynomial in $|V|$.

Consider the set of clique node-sets $U_r \subseteq U$ that may be implemented using a resource $r \in R$, i.e., $U_r = \{u \in U : \forall v \in u, \{v,r\} \in H\}$. It is clear that it only makes sense to select those cliques induced by maximal subsets $u \in U_r : \nexists u' \in U_r : u \subset u'$ for implementation in resource type r. Nonmaximal cliques correspond to so-called "column domination" in unate covering [23]. However, a stronger statement can be made, that only maximum feasible cliques need to be considered as candidates, i.e., $u \in U_r : \nexists u' \in U_r : |u| < |u'|$. This is because Chvatal's heuristic [36] will always return a higher score for a maximum feasible clique than for a nonmaximum clique of the same resource type, and so a maximum clique will always be chosen in preference to a nonmaximum clique. Incorporating this knowledge leads to the proposed resource binding and word-length selection algorithm presented below as Algorithm 6.

The algorithm starts by initializing certain values (steps 1–4). In step 1, C is set to correctly reflect its definition (Table I). V_1 is initialized to the full set of operations (step 2), and will be iteratively reduced as operations are bound to resources (step 5.7). n_r is a counter of how many resources of type $r \in R$ have thus far been allocated by the binding, and is initialized to 0 (step 3). Y, the final set of resources, is initialized to be empty (step 4).

After initialization, the algorithm enters its main loop (step 5), where one resource type is selected, and operations bound to an instance of that resource type, on each iteration. In order to choose which resource type to select, Chvatal's heuristic is applied (steps 5.1–5.2). Compensation for the greedy nature of this heuristic is provided by step 5.3, which can backtrack on previous decisions, as described above. Finally, steps 5.4–5.7 perform the binding: step 5.4 adds a new resource to the existing set, step 5.5 binds each operation in the clique selected by steps 5.1–5.2 to this new resource. Finally, the number of resources of that type is incremented (step 5.6) and the set of unbound operations is reduced (step 5.7).

```
Algorithm 6 (ResBindWLSelect)
Input: A Word-length Compatibility Graph
    $G(V \cup R, C \cup H)$ and schedule $\mathcal{S} : V \rightarrow \mathbb{N} \cup \{0\}$
Output: A resource set $Y$ and a binding $\mathcal{R} : V \rightarrow Y$
Objective: Minimize the area of the resulting
    implementation
begin
  1. $C \leftarrow \{(v_1, v_2) : v_1, v_2 \in V \wedge \mathcal{S}(v_1) + \max_{\{v_1,r\}\in H} L(r) \leq \mathcal{S}(v_2)\}$
  2. $V_1 \leftarrow V$
  3. $n_r \leftarrow 0$ for all $r \in R$
  4. $Y \leftarrow \emptyset$
  5. while $|V_1| > 0$ do
  5.1   foreach $r \in R$ do
          $V' \leftarrow \{v \in N : \exists\{v,r\} \in H\}$
          Let $G'(V',E')$ be the subgraph of $G^+(V,C)$
            induced by vertex set $V'$
          Search $G'(V',E')$ for a maximum clique with
            node set $p_r \subseteq V'$
        end foreach
```

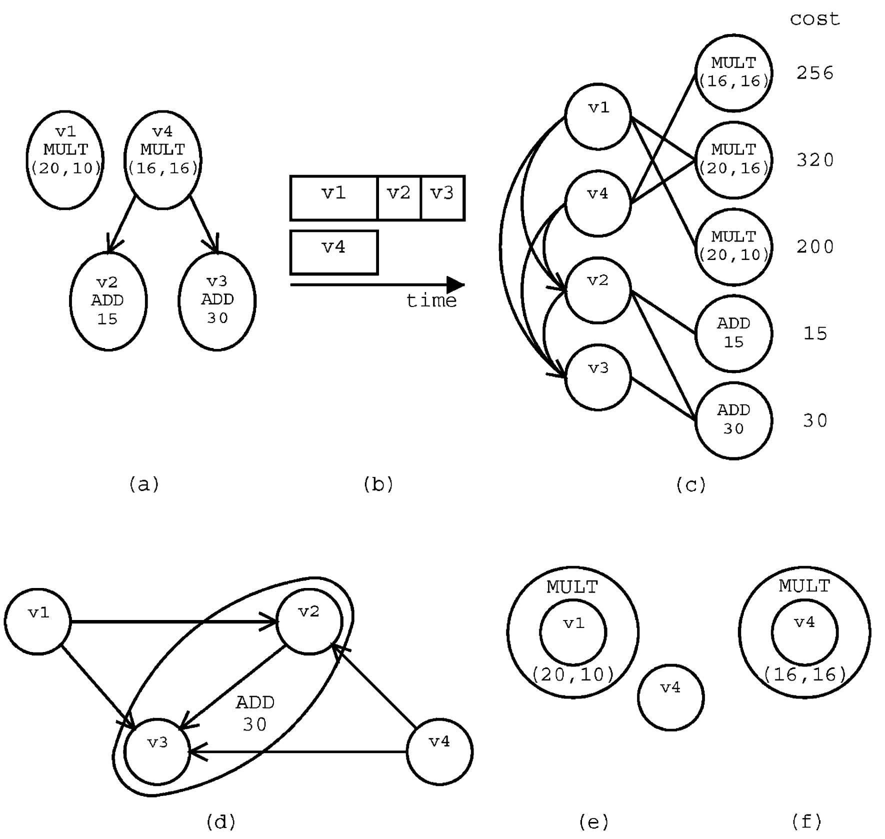


Fig. 10. Example execution of Algorithm 6. (a) Sequencing graph. (b) Schedule. (c) Word-length compatibility graph. (d) First iteration. (e) Second iteration. (f) Third iteration.

```
  5.2  Choose $r \in R$ such that $|p_r|(\text{cost}(r))^{-1}$ is maximum
  5.3  foreach $y \in Y$
         $V' \leftarrow \{v \in V : \mathcal{R}(v) = y\}$
         if $\forall v \in V' : \exists\{v, r\} \in H$ and $V' \cup p_r$ induces a
           clique in$G^+(V, C)$ do
           $Y \leftarrow Y \setminus \{y\}$
           $\mathcal{R}(v) \leftarrow (r, n_r)$ for all $v \in y$
         end if
       end foreach
  5.4  $Y \leftarrow Y \cup \{(r, n_r)\}$
  5.5  $\mathcal{R}(v) \leftarrow (r, n_r)$ for all $v \in p_r$
  5.6  $n_r \leftarrow n_r + 1$
  5.7  $V_1 \leftarrow V_1 \setminus p_r$
     end while
end
```

Because the graph $G^+(V, C)$ (and any subgraph induced by a vertex subset) is a transitively oriented graph, finding the maximum clique is a simple linear-time operation [32].

Example 8: Consider the dataflow graph illustrated in Fig. 10(a). An example execution of Algorithm 6 is illustrated in Fig. 10(d)–(f) for the schedule and word-length compatibility graph shown in Fig. 10(b) and (c). Three iterations are required. During the first iteration, a 30-b adder is selected to perform operations $v2$ and $v3$. The second iteration selects a 20×10-b multiplier to perform operation $v1$ and the final iteration selects a 16×16-b multiplier to perform operation $v4$. The two possibilities faced by the first iteration: a 30-b adder for operations $v2$ and $v3$ or a 15-b adder to perform operation $v2$ only, have equal heuristic scores. However, if the latter possibility were selected, the clique covering $v3$ (selected on the following iteration) would be grown to cover $v2$, resulting in the same binding.

G. Refining Word-length Information

On each iteration of Algorithm 1, if the latency constraint is violated, the word-length information in the word-length compatibility graph is refined in order to guide the algorithm toward a feasible solution. The *bound critical path*, defined below, is calculated in order to provide an insight into which operations may be blocking the creation of a feasible solution. Then a single operation on this bound critical path is selected, and its latency is refined, leading to the deletion of one or more edges from the word-length compatibility graph.

1) The Bound Critical Path: As a first step for refining latency upper bounds, the concept of the *bound critical path* is introduced by extension of the critical path.

Definition 6: Consider a dataflow graph $P(V, D)$. The *critical path* $V_c \subseteq V$ of a dataflow graph $P(V, D)$, given a latency $\ell(v)$ for each node $v \in V$ is defined to be the subset of nodes with equal ASAP and ALAP scheduling times with respect to the minimum possible latency constraint (27)

$$v \in V_c \Leftrightarrow \text{ASAP}(v) = \text{ALAP}(v, \max_{v' \in V}\{\text{ASAP}(v') + \ell(v')\}). \quad (27)$$

Given a dataflow graph $P(V, D)$, a word-length compatibility graph $G(V \cup R, C \cup H)$, a schedule $\mathcal{S} : V \rightarrow \mathbb{N} \cup \{0\}$, a resource set Y, and a resource binding $\mathcal{R} : V \rightarrow Y$, it is possible to construct a set of edges D^b representing operations abutting in time on the same resource (28). Nodes v_1 and v_2 are, thus, connected by an edge in D^b iff node v_1 finishes execution on a resource *the cycle before* node v_2 starts execution on the same resource

$$D^b = \{(v_1 \in V, v_2 \in V) : \mathcal{S}(v_1) + L(\pi_1(\mathcal{R}(v_1))) = \mathcal{S}(v_2) \wedge \mathcal{R}(v_1) = \mathcal{R}(v_2)\}. \quad (28)$$

Definition 7: The *bound critical path* V_b of a scheduled and resource-bound algorithm of dataflow graph $P(V, D)$ is defined to be the critical path of the augmented dataflow graph $P'(V, D \cup D^b)$.

It is possible in this way to capture information about which operations may be responsible for failure to meet the user-specified iteration latency. Once this critical subset of operations has been determined, methods can be applied to refine compatibility information present in the edge set H.

2) Refining Latency Upper Bounds: The reduction of the latency of an operation $v \in V_b$ in the bound critical path could possibly lead to the reduction of the overall latency. Indeed, in order for Algorithm 3 to meet the latency constraint λ, at least one of the operations in the subset V_b' (29) *must* have its latency reduced (V_b' is the subset of the bound critical path consisting of operations whose latency *could* be reduced and then complete within the latency constraint)

$$V_b' = \{v \in V_b : \mathcal{S}(v) + \min_{\{v,r\} \in H} L(r) \leq \lambda \wedge \ell_{\min}(v) \neq \ell_{\max}(v)\}. \quad (29)$$

Reducing the latency of operations that are not members of this set but are nevertheless members of V_b may be necessary, but will clearly not be sufficient to schedule the entire dataflow graph within the required latency bound.

On each iteration of Algorithm 1, one of the operations $v \in V_b'$ is chosen, and the edge set H is adjusted to reduce the upper bound on the latency of v. In the case that $|V_b'| > 1$, the following empirically derived heuristic tie-break rules are applied.

By reducing the upper bound on the latency of operation v, edges $\{v, r\} \in H : L(r) = \ell_{\max}(v)$ will be deleted from H. Considering word-length and type information alone, the potential set of operations $J(v)$ which could share a resource with operation $v \in V$ is given by $J(v) = \{v' \in V \mid \exists r \in R : \{v, r\}, \{v', r\} \in H\}$. A simple heuristic measure would be to select the operation $v \in V$ for which this set is least "affected" by the resultant loss of the edges in H. Thus, the node $v \in V$ minimizing measure (30) is selected. The set $J'(v)$, corresponding to $J(v)$ after removal of the edges in H, is defined as $J'(v) = \{v' \in V \mid \exists r \in R : (\{v, r\}, \{v', r\} \in H \wedge L(r) \neq \ell_{\max}(v))\}$

$$|J(v) \setminus J'(v)| \cdot |J(v)|^{-1}. \quad (30)$$

Once again, in case of tie break on the above measure, a further heuristic can be applied: those operations currently bound to resources utilizing less than the upper-bound latency of that operation are preferred candidates. Thus, an arbitrary node $v \in V_b'$ maximizing (30) and satisfying $L(\pi_1(\mathcal{R}(v))) < \ell_{\max}(v)$ is selected, if one exists. Otherwise simply an arbitrary node maximizing (30) is selected.

This procedure is illustrated in Algorithm 7. After constructing the abuttal edges (28) in step 1, the bound critical path is extracted (step 2), and the subset V_b' of the bound critical path (29) is found (step 3). If this set is empty, no refinement of word-length information can help the search for a feasible solution, and the failure case is returned (step 4). Otherwise, a search for an appropriate operation v to refine is conducted (steps 5–6), according to the heuristics discussed above. Once a node has been found, the edge set *H*, representing which resource types can perform which operations, it is refined by removing all edges connecting the chosen operation to resources of latency equal to the maximum of all resource types for that operation.

```
Algorithm 7 (WLRefinement)
**Input:** A dataflow graph $P(V, D)$, word-length
    compatibility graph $G(V \cup R, C \cup H)$,
    latency constraint $\lambda$, resource set $Y$ and
    resource binding $\mathcal{R} : V \to Y$
**Output:** A refined word-length compatibility graph
**Objective:** Minimize the number of resource types
    to which each operation could be bound
**begin**
  1. Construct the abuttal edges $D^b$ (28)
  2. Perform ASAP and ALAP scheduling on $P(V, D \cup D^b)$
  3. Extract the subset $V_b' \subseteq V$ of nodes on the bound
         critical path which could complete within the
         latency constraint (29)
  4. **if** $V_b' = \emptyset$ **do**
         **return** failure case
  5. Find $V_p \subseteq V_b'$ of nodes maximizing the measure (30)
  6. **if** $\exists v \in V_p : L(\pi_1(\mathcal{R}(v))) < \ell_{\max}(v)$ **do**
         Select one such node $v \in V_p$
     **else do**
         Select an arbitrary node $v \in V_p$
     **end if**
  7. $H \leftarrow H \setminus \{\{v, r\} \in H : L(r) = \ell_{\max}(v)\}$
**end**
```

Example 9: Fig. 11 illustrates an example refinement phase corresponding to the dataflow graph introduced in Fig. 2 and reproduced in Fig. 11(a) for convenience. Nodes that are on the computation critical path with respect to the dataflow graph $P(V, D)$ are highlighted in Fig. 11(a), $V_c = \{a_2, m_3, m_4, m_5, a_1, a_3\}$. This dataflow graph has been scheduled and resource-bound in Fig. 11(b). The portions of node execution time between $\ell_{\min}(v)$ and $\ell_{\max}(v)$ have been shaded in the figure.

The augmented dataflow graph $P'(V, D \cup D^b)$ obtained from time-abutment edges D^b is illustrated in Fig. 11(c) and consists of a single extra edge from operation m_2 to operation m_5. The resulting change in critical path is significant. The bound critical path is given by $V_b = \{m_1, m_2, m_5, a_3\}$.

For $\lambda = 18$, the lowest achievable latency constraint, the subset V_b' is given by $V_b' = \{m_1, m_2\}$. The heuristic measures described above may then be applied to decide which of these two nodes is to have its upper bound latency reduced.

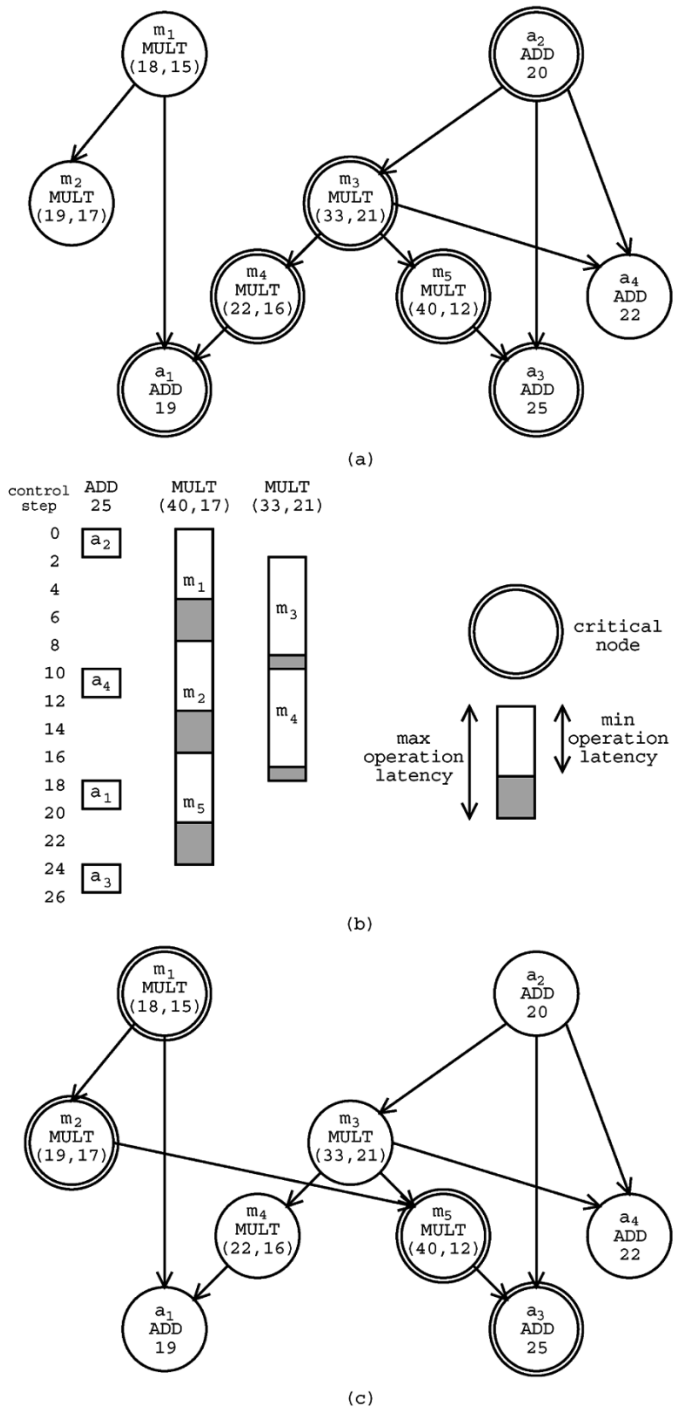


Fig. 11. Example use of the bound critical path for word-length refinement (Example 9). (a) Sequencing graph. (b) Initial schedule. (c) Augmented sequencing graph.

VI. Results and Discussion

Before considering in detail the performance of the methods introduced in this paper, it is instructive to follow through the dataflow graph of Fig. 2 which has been used as an example throughout the paper. Both optimal and heuristic schedules, resource allocations, bindings and word-length selections have been performed for this example in order to explore the area/latency tradeoff achievable. Fig. 12 plots these results. Not all ILP results are shown, due to excessive ILP solver execution time.

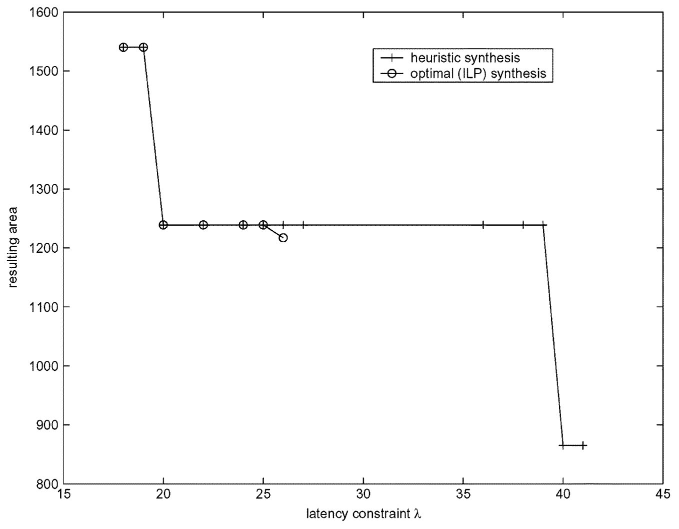


Fig. 12. Design-space exploration for the simple dataflow graph of Fig. 2.

The ILP and heuristic solutions are identical for all cases except $\lambda = 26$, where there is a slight difference caused by the presence of an extra adder in the heuristic solution.

Fig. 13 illustrates further the results for several benchmark circuits. The FIR filter is a 126-tap linear-phase low-pass Direct Form II transposed [37] structure, suggested by [38] as a representative DSP design. The DCT is an 8-point, one-dimensional (1-D) decimation in time structure from [30] which has also been suggested as a benchmark by [38]. As an illustration of the flexibility of multiple word-length implementations, two versions of this benchmark have been synthesized, one (DCT^1) with equal error tolerance on all outputs, and the other (DCT^2) with required signal-to-noise ratio (SNR) reducing by 3-dB/p DCT coefficient, so that low-frequency coefficients are less noisy than high-frequency ones. The IIR filter is of fouth-order, as used by [39]. The polyphase filter bank (PFB) is the design given in [40] for evaluation of the streams-*C* compiler. The RGB to YCrCb converter is of the form suggested by the ITU [41], and is also of particular interest for multiple word-length implementation, as it allows some quantization error in the Cr and Cb outputs whereas the Y output is guaranteed to be error-free.

For comparison, not only are the optimal (ILP) results and the heuristic results provided, but also the solutions corresponding to "word-length-blind" scheduling followed by an *optimal* resource binding [16]. All three sets of results are only provided for the IIR filter, the polyphase filter bank and the RGB to YCrYb converter, due to the excessive execution time of both the ILP solver and the optimal binding.

These results illustrate that for the benchmark circuits, the heuristic presented in this paper provides a significant improvement in area over a two-stage approach of scheduling and then binding (between -16% and 46%, average 15%), even when the binding is optimal. The heuristic results have between 0% and 62% (average 14%) worse area than the optimum combined solution, for cases where the optimum is known.

Table II provides details on the minimum, maximum, and final number of multipliers used by Algorithm 1 for the heuristic results of Fig. 13.

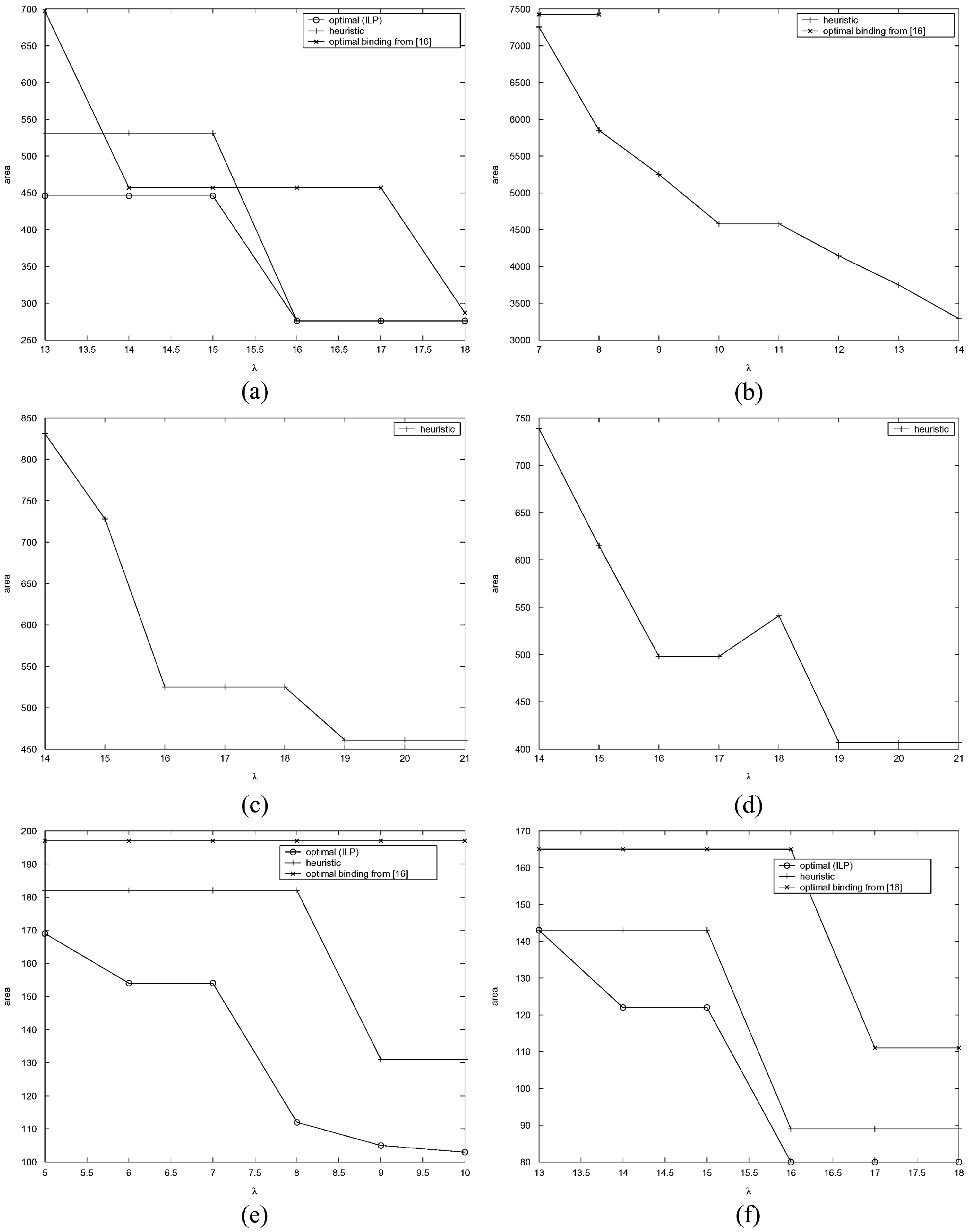


Fig. 13. Design-space exploration for some benchmark circuits. (a) IIR. (b) FIR. (c) DCT^1. (d) DCT^2. (e) PFB.(f) RGB-YCrCb.

In order to fully characterize the heuristic performance, further results have been obtained using artificially generated examples. For statistically significant data on solution quality, 200 random dataflow graphs have been generated for each (problem

TABLE II
MULTIPLIER STATISTICS FOR HEURISTIC ALLOCATION ON BENCHMARK CIRCUITS

Benchmark	λ	m_{min}	m_{max}	m_{final}
IIR	13–15	2	2	2
	16–18	1	1	1
FIR	7	63	63	63
	8	35	63	44
	9	32	63	40
	10–11	32	52	32
	12	21	32	29
	13	21	32	26
	14	18	32	21
DCT[1]	14	6	25	15
	15	5	25	12
	16–18	5	25	8
	19–21	4	6	4
DCT[2]	14	5	25	10
	15	5	25	8
	16–17	4	25	7
	18	4	10	6
	19–21	4	6	5
PFB	5–8	2	4	2
	9–10	1	1	1
RGB–YCrCb	13–15	2	2	2
	16–18	1	1	1

size $|V|$, latency constraint λ) pair, using a variant of the TGFF algorithm [42]. The first set of detailed quality results that has been collected measures the variation of the heuristic solution quality with both the problem size and the tightness of the supplied latency constraint. For each dataflow graph, the minimum possible latency constraint λ_{min} has been found using ASAP scheduling, from which latency constraints have been generated corresponding to a 0% to 30% relaxation of λ_{min}. Algorithm 1 has then been executed on the graph/latency constraint combination. The resulting area has been normalized with respect to the optimal solution resulting from [16] where operations may only share resources when the implemented resource has latency no longer than the minimum required to implement the given operations. These results are plotted in Fig. 14, as a percentage area penalty for using the approach of [16] over the heuristic presented in this paper. Each point represents the mean of two hundred representative designs.

The results illustrate that for designs with even a small "slack" in terms of latency constraints, significant area improvements of up to 30% can be made by performing the scheduling, binding, and word-length selection in the intertwined manner proposed. The area improvements come from increased resource sharing due to implementing small word-length operations in larger word-length resources with longer latency. Even for relatively small graphs, area improvements of tens of percent are possible.

Fig. 15 illustrates the increase in implementation area from using the heuristic presented in Section V over the optimum combined problem presented in Section IV. These results can only be provided for modest problem size and a minimum latency constraint $\lambda = \lambda_{\text{min}}$, as the ILP solution execution time scales rapidly with problem size and as the latency constraint is relaxed.

The variation of synthesis algorithm execution time (for designs with minimum latency) against problem size for 700 graphs using the ILP model (solved with `lp_solve` [29]

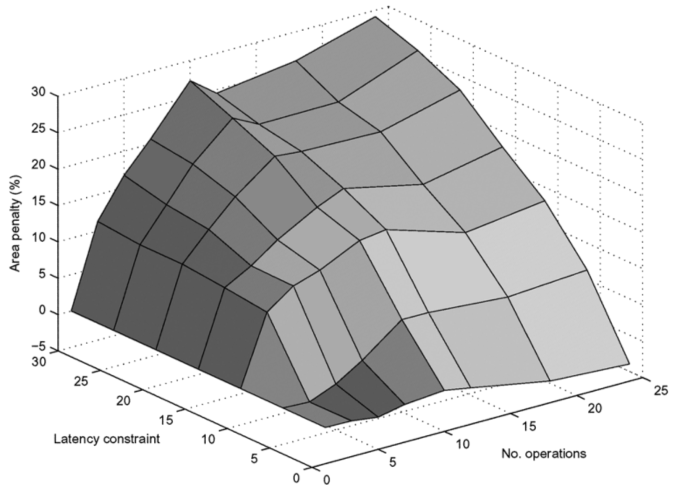


Fig. 14. Variation with number of operations and latency constraint of area penalty for [16] over the proposed heuristic.

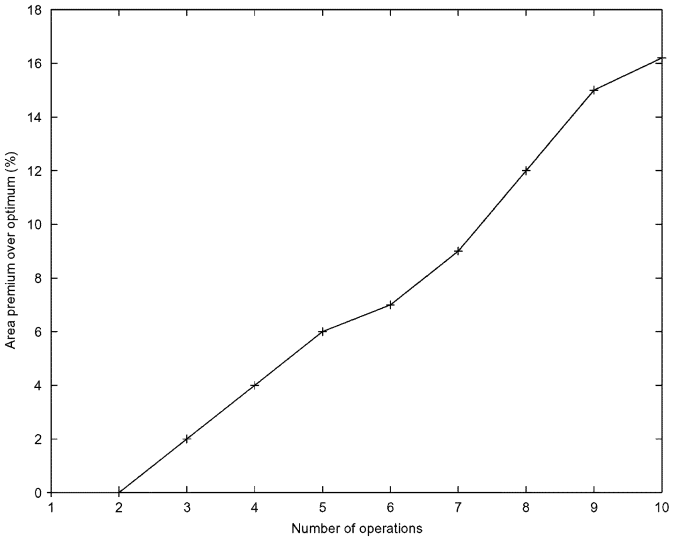


Fig. 15. Variation of area premium for Algorithm 1 over the optimum solution.

on an AMD Athlon 1.4 GHz) and the heuristic algorithm (implemented in C on the same machine) is shown in Fig. 16, illustrating the polynomial complexity of the heuristic against the exponential complexity of the ILP solution search. Over the range of 1 to 10 operations, the relative increase in area ranges from 0% to 17% whereas the ILP solution takes between one and three orders of magnitude greater time to execute. An important point not brought out by these results is the scaling of execution time with overall latency constraint. The number of variables in the ILP model scales with the latency constraint, making the execution time highly dependent on this parameter. This is illustrated in Table III for 700 9-operation dataflow graphs. By contrast, the execution time of Algorithm 1 does not scale with the latency constraint. Thus, the 1–3 orders of magnitude illustrated in Fig. 16 are under conditions most favorable to the ILP-based solution; a more relaxed constraint would lead to a much larger speedup. Note, however, that it may well be possible to take advantage of symmetries in the ILP formulation [43] to speed up the ILP execution time. Since the main use of ILP in this paper is as a comparative approach for

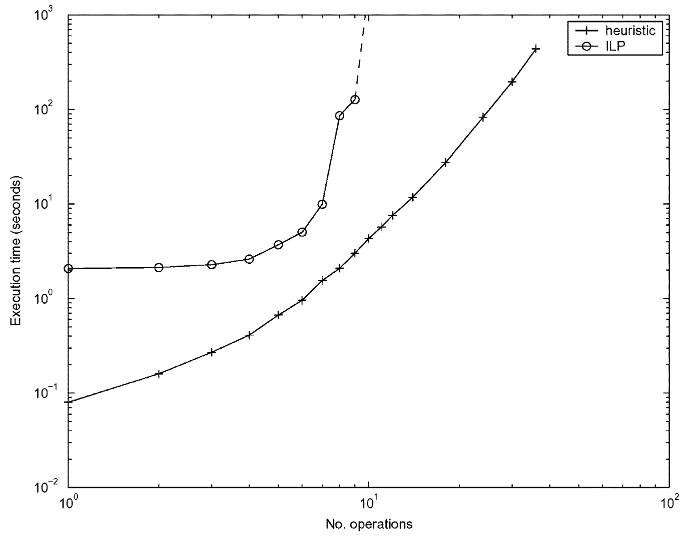


Fig. 16. Variation with number of operations of execution time for 700 graphs (heuristic and ILP solutions).

TABLE III
VARIATION OF EXECUTION TIME FOR 700 GRAPHS WITH $\lambda/\lambda_{\mathrm{min}}$ FOR HEURISTIC AND ILP SOLUTION

$\lambda/\lambda_{\mathrm{min}}$	heuristic (secs)	ILP (mins:secs)
1.00	3.02	2:07.09
1.05	3.51	4:05.21
1.10	3.73	15:55.56
1.15	3.52	>30:00.00

evaluating the quality of the results of the proposed heuristic, this has not been explored in detail.

The results presented clearly indicate the practical nature of the heuristic presented in this paper, whose execution results in a speedup of up to three orders of magnitude over ILP solution, even for modest graph sizes, at an area-penalty of between 0% and 16%.

VII. CONCLUSION

This paper has presented work addressing the problem of architectural synthesis for multiple word-length systems. It has been demonstrated that the traditional architectural synthesis problems of operation scheduling, resource allocation, and resource binding, can be significantly complicated by the presence of multiple word-length arithmetic.

An ILP construction for the multiple word-length architectural synthesis problem has been formulated. The ILP formulation provides solutions which are optimal with respect to the area-based cost function, but suffers from long run-times which scale up rapidly with relaxation of the latency constraint.

An heuristic solution has been developed based on intertwined scheduling, resource binding/word-length selection, and word-length refinement. This involves techniques for scheduling with incompletely defined word-length information, combining binding and word-length selection, and latency-based word-length refinement based on critical path analysis.

The results from an implementation of both the ILP and the heuristic approach show that significant improvements to system area can be made by allowing noncritical operations to share large word-length resources. The heuristic solution has been shown to be within 17% of the optimal area, over a range of minimum-latency problem sizes for which between one and three orders of magnitude speedup over an ILP solver has been achieved. For nonminimum-latency problems, the execution speedup is even greater, and the heuristic lies within 25% of the optimum solution quality for all but one benchmark.

Excessive sharing of resources could cause timing violations due to the timing overhead of the multiplexers and extra routing introduced. These timing considerations are currently not modeled either in the ILP or the heuristic solution. Experimentation shows, however, that for the target platform used to collect results, such timing violations have not occurred for any of the benchmark circuits in this paper. Similarly, resource sharing causes an area overhead, which has not been modeled by the proposed techniques. Experimentation shows that, while this overhead is likely to reduce the gains presented in this paper somewhat, the savings due to the proposed technique outweigh the overheads of further resource sharing for the target technology and the benchmarks studied. Should the overhead issues be of concern in a different implementation technology, it would be straight-forward to incorporate a simple upper-bound on the sharing of a single resource as an extra set of linear constraints in the ILP formulation, or as a modification of Algorithm 6 in the heuristic solution.

As deep submicron effects become more dominant in digital system design, resource sharing becomes more dependent on physical placement. As well as overcoming the restrictions mentioned above, in the future we aim to incorporate floorplan-based routing measures into the formulation. In addition, we are actively investigating the interaction between the synthesis problem described in this paper and the word-length optimization problem in digital signal processing. The aim in this work is to target power consumption in addition to area optimization [7].

ACKNOWLEDGMENT

The authors wish to acknowledge the support of Hewlett-Packard Laboratories and the Engineering and Physical Sciences Research Council, U.K. They also thank the anonymous reviewers for their extensive and helpful suggestions.

REFERENCES

[1] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens, "A methodology and design environment for DSP ASIC fixed point refinement," in *Design Automation and Test Europe*, Germany, 1999.
[2] K.-I. Kum and W. Sung, "Combined word-length optimization and high-level synthesis of digital signal processing systems," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 921–930, Aug. 2001.
[3] G. A. Constantinides, "High Level Synthesis and Word Length Optimization of Digital Signal Processing Systems," Ph.D. dissertation, University of London, England, U.K., 2001.
[4] Y. Cao and H. Yasuura, "Quality-driven design by bitwidth optimization for video applications," in *Proc. IEEE/ACM Asia and South Pacific Design Automation Conf.*, 2003, pp. 532–537.
[5] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "The multiple word-length paradigm," in *IEEE Symp. Field-Programmable Custom Computing Machines*, Rohnert Park, CA, Apr.–May 2001.
[6] S. A. Wadekar and A. C. Parker, "Accuracy sensitive word-length selection for algorithm optimization," in *Proc. Int. Conf. Computer Design*, Austin, TX, Oct. 1998, pp. 54–61.
[7] G. A. Constantinides, "Perturbation analysis for word-length optimization," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, Napa, CA, Apr. 2003.

[8] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee, "Precision and error analysis of MATLAB applications during automated hardware synthesis for FPGAs," in *Proc. Design Automation and Test Europe*, Munich, Germany, 2001, pp. 722–728.

[9] M. W. Stephenson, "Bitwise: Optimizing Bitwidths Using Data-Range Propagation," Master's thesis, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, May 2000.

[10] G. DeMicheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994.

[11] R. Jain, A. Parker, and N. Park, "Module selection for pipelined synthesis," in *Proc. 25th ACM/IEEE Design Automation Conf.*, 1988, pp. 542–547.

[12] R. Jain, "MOSP: Module selection for pipelined designs with multicycle operations," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1990, pp. 212–215.

[13] M. Ishikawa and G. D. Micheli, "A module selection algorithm for high-level synthesis," in *Proc. IEEE Int. Symp. Circuits and Systems*, 1991, pp. 1777–1780.

[14] I. G. Harris and A. Orailoğlu, "Intertwined scheduling, module selection, and allocation in time-and-area constrained synthesis," in *Proc. IEEE Int. Conf. Circuits and Systems*, 1993, pp. 1682–1685.

[15] K. Kum and W. Sung, "Word-length optimization for high-level synthesis of digital signal processing systems," in *Proc. IEEE Int. Workshop on Signal Processing Systems*, 1998, pp. 569–678.

[16] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Multiple-word-length resource binding," in *Field-Programmable Logic: The Roadmap to Reconfigurable Systems*. ser. Lecture Notes in Computer Science, H. Gruenbacher and R. Hartenstein, Eds. New York: Springer-Verlag, 2000.

[17] M. C. Molina, J. M. Mendias, and R. Hermida, "Multiple-precision circuits allocation independent of data-objects length," in *Proc. Design Automation and Test Europe*, Paris, Mar. 2002, pp. 909–913.

[18] J. Choi, H. Jun, and S. Hwang, "Efficient hardware optimization algorithm for fixed-point digital signal processing ASIC design," *IEE Electron. Lett.*, vol. 32, no. 11, pp. 992–994, May 1996.

[19] T. R. Jensen and B. Toft, *Graph Coloring Problems*. New York: Wiley, 1995.

[20] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Optimum word-length allocation," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, Napa, CA, Apr. 2002.

[21] K. K. Parhi, C.-Y. Wang, and A. P. Brown, "Synthesis of control circuits in folded pipelined DSP architectures," *IEEE J. Solid-State Circuits*, vol. 27, pp. 29–43, Jan. 1992.

[22] S. D. Haynes, J. Stone, P. Y. K. Cheung, and W. Luk, "Video image processing with the Sonic architecture," *IEEE Comput.*, vol. 33, pp. 50–57, Apr. 2000.

[23] R. S. Garfinkel and G. L. Nemhauser, *Integer Programming*. New York, NY: Wiley, 1972.

[24] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 464–475, Apr. 1991.

[25] H. Achatz, "Extended 0/1 LP formulation for the scheduling problem in high-level synthesis," *Proc. EURODAC with EURO-VHDL*, 1993.

[26] L. E. Lucke and K. K. Parhi, "Generalized ILP scheduling and allocation for high-level DSP synthesis," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1993, pp. 5.4.1.–5.4.4..

[27] B. Landwehr, P. Marwedel, and R. Dömer, "OSCAR: Optimum simultaneous scheduling, allocation, and resource binding," in *Proc. European Design Automation Conf.*, 1994, pp. 90–95.

[28] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Optimal datapath allocation for multiple-word-length systems," *IEE Electron. Lett.*, vol. 36, pp. 1508–1509, Aug. 2000.

[29] H. Schwab. (1997) lp_solve. [Online]. Available: ftp://ftp.es.ele.tue.nl/pub/lp_solve

[30] K. Parhi, *VLSI Digital Signal Processing Systems*. New York: Wiley, 1999.

[31] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Heuristic datapath allocation for multiple-word-length systems," in *Design Automation and Test Europe*, Mar. 2001.

[32] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. New York: Academic, 1980.

[33] J. Nestor and D. Thomas, "Behavioral synthesis with interfaces," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1986, pp. 112–115.

[34] C. Chantrapornchai, E. H.-M. Sha, and X. S. Hu, "Efficient design exploration based on module utility selection," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 19–29, Jan. 2000.

[35] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 661–679, Jun. 1989.

[36] V. Chvatal, "A greedy heuristic for the set-covering problem," *Math. Operations Res.*, vol. 4, no. 3, pp. 233–235, Aug. 1979.

[37] S. K. Mitra, *Digital Signal Processing*. New York: McGraw-Hill, 1998.

[38] C. Lee, D. Kirovski, I. Hong, and M. Potkonjak, "DSP QUANT: Design, validation, and applications of DSP hard real-time benchmark," in *Proc. IEEE Int. Conf. Acoustics Speech and Signal Processing*, vol. 1, 1997, pp. 671–682.

[39] K.-I. Kum, J. Kang, and W. Sung, "AUTOSCALER for C: An optimizing floating-point to integer C program convertor for fixed-point digital signal processors," *IEEE Trans. Circuits Syst. II*, vol. 47, pp. 840–848, Sep. 2000.

[40] J. Frigo, M. Gokhale, and D. Lavenier, "Evaluation of the Streams-C C-to-FPGA compiler: An applications perspective," in *Proc. ACM/SIGDA Int. Symp. on FPGAs*, Napa, CA, 2001.

[41] B. L. Evans, Raster Image Processing on the TMS320C7X VLIW DSP.

[42] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free," in *Proc. CODES/CASHE*, 1998, pp. 97–101.

[43] B. Landwehr, P. Marwedel, I. Markhof, and R. Dömer, "Exploiting isomorphism for speeding-up instance-binding in an integrated scheduling, allocation and assignment approach to architectural synthesis," in *Proc. Conf. Computer Hardware Description Languages and their Applications*, Toledo, Spain, Apr. 1997.

George A. Constantinides (S'96–M'01) received the M.Eng. degree in information systems engineering and the Ph.D. degree in electrical and electronic engineering from Imperial College, London, U.K., in 1998 and 2001, respectively.

Since 2002, he has been a Lecturer in digital systems, Electrical and Electronic Engineering Department, Imperial College. He is the author of 31 refereed conference and journal papers. He is also author of *Synthesis and Optimization of DSP Algorithms* (Dordrecht, Germany: Kluwer, 2004). His research interests include reconfigurable computing and electronic design automation, with a particular focus on digital signal processing algorithms.

Dr. Constantinides was program Co-Chair of the International Conference on Field-Programmable Logic and Applications in 2003, and serves on the Program Committees of FPL, FPT, ISCAS, ERSA, and ARC. He was the Founding Chair of the U.K. SIGDA Chapter, serving as General Chair in 2001, and Technical Chair from 2002 to 2003 of the annual workshop. He is a member of ACM.

Peter Y. K. Cheung (M'85–SM'04) received the B.Sc. degree in electrical engineering from Imperial College, London, U.K., in 1973.

After working at Hewlett-Packard for a number of years, he returned to Imperial College as a Research Assistant and was appointed as a Lecturer in 1980. Currently, he is a Professor of digital systems and Deputy Head of the Electrical and Electronic Engineering Department at Imperial College. His research interests include VLSI architectures for DSP and video processing, reconfigurable computing, embedded systems, and high-level synthesis and optimization of digital systems, particularly those containing field programmable logic. He has coauthored over 100 publications and two research monographs in these areas.

Prof. Cheung has served on the Technical Program Committee of many international conferences including ISCAS, FPL, FPT, and DATE.

Wayne Luk (S'85–M'85) is Professor of Computer Engineering, Department of Computing, Imperial College London, U.K., where he leads the Custom Computing Research Group. Much of his current work involves high-level compilation techniques and tools for parallel computers and embedded systems, particularly those containing reconfigurable devices such as field-programmable gate arrays. His research interests include theory and practice of customizing hardware and software for specific application domains, such as graphics and image processing, multimedia, and communications.