Hardware-Software Co-design for Reconfigurable Field Programmable Gate Arrays Using Mixed-Integer Programming

Faridah M. Ali
Department of Computer Engineering,
Kuwait University, P.O. Box 5969, Safat 13060, Kuwait
E-mail: faridah@puc.edu.kw

Helal Al-Hamadi
Department of Information Science,
Kuwait University, P.O. Box 5969, Safat 13060, Kuwait
E-mail: helal@cfw.kuniv.edu

Ahmed Ghoniem
Department of Finance and Operations Management,
Isenberg School of Management, University of Massachusetts Amherst,
Amherst, MA 01002, U.S.A.
E-mail: aghoniem@isenberg.umass.edu

Hanif D. Sherali
Grado Department of Industrial and Systems Engineering (0118),
Virginia Tech,
Blacksburg, VA 24061, U.S.A.
E-mail: hanifs@vt.edu

Keywords: hardware-software co-design, FPGAs, task partitioning, scheduling, mixed-integer 0-1 programming

Received: September 19, 2011

This paper presents a novel mixed-integer programming formulation for scheduling non-preemptive, aperiodic, hard real-time tasks with precedence constraints. It provides an integrated partitioning and scheduling co-synthesis approach. The problem formulation maps some n precedence-related, indivisible jobs having specified processing requirements, release times, and due-dates to a system involving a single Central Processing Unit (CPU) and up to m potential reconfigurable Field Programmable Gate Arrays (FPGAs). We provide a time-indexed mixed-integer 0-1 programming formulation that jointly assigns tasks to either the CPU or to one of the FPGAs, and determines the task sequence for each software or hardware component that is utilized, with the objective of minimizing a composite cost of task partitioning and scheduling. Computational experience is provided using randomly generated instances to demonstrate the applicability of the proposed methodology.

Povzetek: Predstavljen je algoritem za porazdeljevanje opravil pri snovanju programske in strojne opreme.

1 Introduction and Motivation

The task partitioning and scheduling problem bears practical significance in software/hardware co-design of hard real-time applications that arise in a host of applications such as flight and defense control, telecommunication, or nuclear power plants, to name a few. Specifically, we consider the problem of partitioning and scheduling n indivisible (no preemption), aperiodic (which could be considered as the body of a looped system), precedence-related jobs that are characterized by specific processing requirements, release times, and due-dates (which are deadlines that cannot be violated). The system architecture is depicted in Figure 1, and involves a single Central Processing Unit (CPU) and a maximum of some m potential reconfigurable Field Programmable Gate Arrays (FPGAs) controlled by a single controller unit. The system components are connected with two explicit communication buses (channels). The first bus is the system bus, which is used for input/output (I/O) data transfers, whereas the second is used for FPGA reconfiguration transfers. The task processing effort is primarily carried out by the CPU, in general, and the FPGAs are incrementally utilized if the CPU alone cannot conform to all due-date restrictions. In contrast with scheduling problems

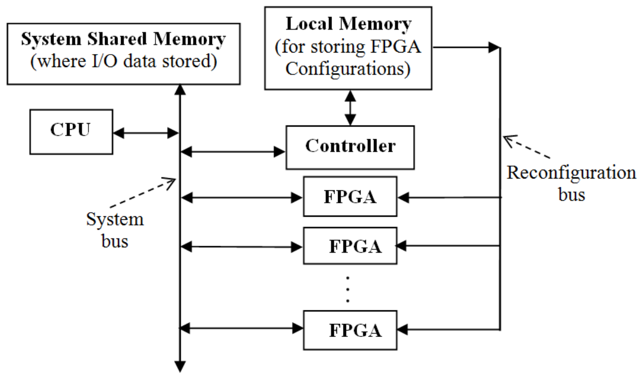

Figure 1: System Architecture

that arise in production and logistical systems where it is often desirable to meet imposed due-dates, it is *imperative* to comply with the specified due-dates in the problem under investigation.

Another reason for the use of such dual systems in practice resides in the benefits accruing from the cooperation between software and hardware components. As a consequence, the co-design problem has gained increasing attention over the last decade, (see [6], [7], [10], [12], [16], [19], and [20]). This process involves three main operations, namely, resource (software, hardware, and auxiliary components) *allocation* to the system, task *partitioning* among software and hardware processing components, and *scheduling* of the tasks over their assigned processing components (CPU or FPGA). Most works in the literature have addressed this joint problem via two-phase methods [14], where a task partitioning is achieved first, and then tasks are subsequently scheduled over the relevant processing components. It is important, however, to note that the task partitioning and scheduling operations are intertwined [9], and need to be dealt with concurrently in order to determine optimal operational solutions.

An algorithm that aims at partitioning and scheduling the tasks on two CPUs and several hardware components with the objective of minimizing the total execution time and the hardware cost was presented by Liu and Wong [13]. Arato et al. [2] designed a procedure for scheduling tasks on a software component, where tasks that violate their deadlines are partially assigned to an auxiliary hardware component. A mixed-integer 0-1 formulation for partitioning precedence-related tasks on a single-CPU-single-FPGA system, as well as a genetic algorithm to address larger problem instances was presented in [2]. Ali and Das [1] presented a heuristic algorithm that progressively assigns indivisible tasks to dynamically reconfigurable FPGAs when certain tasks cannot meet their deadlines on the CPU. In contrast, this challenging scheduling problem is tackled in the present paper using a mixed-integer 0-1 programming model with the objective of minimizing a composite cost of task partitioning and scheduling on a single CPU along with several potentially reconfigurable FPGAs.

Jeong et al. [11] proposed a mixed-integer 0-1 formulation and a heuristic for hardware-software partitioning in systems consisting of a single CPU and a single FPGA, with the objective of minimizing the reconfiguration overhead. A mixed-integer programming model was developed by Niemann and Marwedel [14] that employs a two-phase method for hardware/software partitioning, where a tentative schedule is first proposed and is subsequently verified; if the timing constraints are violated, the partitioning step is repeated with timing constraints that are tighter than the estimated scheduling horizon length. Bender [3] discussed an alternative mixed-integer programming approach for mapping real-time precedence graphs into a system of Application Specific Integrated Circuits (ASICs) that are used as hardware components, and pipelined microprocessors that are used as software components. Initially, only a limited number of hardware components are used, and these are incrementally increased until a feasible solution is obtained.

Recently, hardware/software partitioning for multimedia and wireless mobile applications has gained great importance. Brogioli et al. [4] proposed a set of criteria for partitioning real-time embedded multimedia applications between software programmable Digital Signal Processors (DSPs) and hardware-based FPGA coprocessors. Dasu and Panchanathan [5] investigated the design and development of a dynamically reconfigurable multimedia processor that involves an optimal hardware/software codesign methodology. Furthermore, a hardware/software partitioning for multimedia application that utilizes process-level pipelining and a heuristic technique based on simulating annealing was presented by Juan et al. [12].

A design space exploration tool that supports both explicit communication and reconfigurable hardware was addressed by Haubelt et al. [10]. The developed algorithm strictly separates functionality from the architecture, and maps a process graph onto components such that data dependencies given by the process graph can be handled in the resulting implementation. The work presented in the present paper bears some similarity to this approach in that we also use explicit communication channels and reconfigurable hardware in our system architecture.

Observe that the problem at hand is also related to the challenging class of unrelated parallel machine scheduling problems (see [15]). It is characterized by the presence of release dates, imperative due-dates, precedence constraints, inter-task data communication, reconfiguration of hardware resources (FPGAs), and a composite objective function that minimizes the processing and resource utilization costs. Also, it is specially structured due to the fact that all processing components are identical, except for the CPU.

The main contribution of the present paper is a novel formulation of a time-indexed mixed-integer 0-1 programming model for the co-synthesis hardware/software integrated partitioning and scheduling problem. The motivation for the proposed work is two-fold: first, it provides a tool for simultaneously partitioning (or mapping) and scheduling tasks onto hardware/software units, and second, it offers a design space exploration tool to ascertain the minimum number of FPGAs required for a particular application before a system is actually built.

The remainder of this paper is organized as follows. In Section 2, we formally describe the problem under investigation along with our notation. Thereafter, we introduce in Section 3 a mixed-integer 0-1 programming formulation that simultaneously captures the requirements pertaining to the partitioning and scheduling operations. Section 4 delineates our data generation scheme, and reports our computational experience using a set of random test instances to demonstrate the effectiveness of the proposed solution approach. We close the paper in Section 5 with a summary of our findings.

2 Problem Description and Notation

We address the problem of scheduling some n precedence-related jobs having specified processing requirements, release times, and inviolable due-dates in a system involving a single *Central Processing Unit* (CPU) and a maximum of some m potential reconfigurable *Field Programmable Gate Arrays* (FPGAs). Each job can be processed either by the CPU itself, or it can be scheduled for processing on one of the m available FPGAs. In either case, no preemption is permitted, and each resource (CPU or FPGA) can process at most one job at any point in time. However, whenever an FPGA begins processing a job, it must be reconfigured to perform the required operation by a single available reconfiguration *controller*. This reconfiguration process consumes a specified duration that is part of the total processing time required for performing the job on the associated FPGA. Note that while the controller is reconfiguring any FPGA to begin processing a job, it is occupied and cannot simultaneously reconfigure another FPGA. Again, no preemption is permitted in the reconfiguration process. Also, not all the m available FPGAs need be used; in fact, there is a fixed cost for using an FPGA that competes with the cost related to achieving scheduling efficiency. When an FPGA finishes executing any task, it becomes available for the next task if needed. This is described more in detail in the model formulation given in Section 3.

In our analysis, the FPGAs cannot be preconfigured since it is not known in advance which task will be executed on the FPGA rather than on the CPU. Moreover, if the system has more than one FPGA, it is not known which FPGA will be used until scheduling is complete. The proposed system (Figure 1) is generic and can be used for executing any precedence-related jobs. However, for a given real-time set of jobs, once an optimal system configuration and scheduling decisions are determined, then a partial run-time reconfiguration can be performed during implementation where only the configuration bits of the particular task are transferred to the FPGA in order to reduce the configuration time.

Notation:

- $j = 1, ..., n$: Index for jobs.
- For establishing precedences, we define $P = \{(j_1, j_2) : j_1 \to j_2$, i.e., the processing of job j_1 must precede that of job $j_2\}$.
- m = maximum potential number of FPGAs available for use.
- Index for *time-slots*: Let the time be discretized so that the scheduling time-line for the CPU and each FPGA contains s time-slots, where the end of time-slot s is estimated to be the maximum allowable makespan duration, based on due-dates. We index the time-slots over this maximum makespan duration as: $t = 1, ..., s$. (Note that the actual duration of the time-slots is arbitrary and rescalable, and typically ranges from 1 to 300 seconds in practice. Also, the time measurement (in discretized units) begins at time $t = 0$ at the beginning of slot 1.)
- Index for *slots*: These correspond to a sequential indexing of the foregoing time-slots over the resources, where the slots for the CPU are indexed as $k = 1, ..., s$, the slots for FPGA 1 are indexed as $k = s + 1, ..., 2s$, the slots for FPGA 2 are indexed as $k = 2s + 1, ..., 3s$, and so on, up to slots $k = ms + 1, ..., (m + 1)s$ for FPGA m. (Note that the *time-slots* are numbered $1, ..., s$, whereas the *slots* are indexed contiguously over the CPU and the m FPGAs as $k = 1, ..., (m + 1)s$.)
- Index for resources: $r = 0, 1, ..., m$, where $r = 0$ is the CPU and $r = 1, ..., m$ index the FPGAs.
- $r(k)$ = resource corresponding to slot k. (So $r(k) = 0$ for $k = 1, ..., s$, $r(k) = 1$ for $k = s + 1, ..., 2s$, and so on.)
- δ_{jr} = processing time of job j on resource r (in integral time units that conform with the time-slot duration).
- π_{jr} = reconfiguration time on resource FPGA r to process job j (in integral time units that conform with the time-slot duration). We assume that π_{jr} is included within $\delta_{jr}, \forall j, \forall r \geq 1$.
- α_j = release time of job j. That is, if a job j has no predecessors, then the earliest time-slot to start its processing is $\alpha_j + 1$.
- d_j = due-date of job j.
- lb_j = lower bound on the starting time-slot for job j. If a job has no predecessor, then $lb_j = \alpha_j + 1$; otherwise we may simply take $lb_j = \max\{\alpha_j + 1, \max_{j_1:(j_1,j)\in P}\{lb_{j_1} + \min_r\{\delta_{j_1 r}\}\}\}$.

– $k \bmod^+ (s)$ = remainder for the division k/s, except that this is taken as s if the remainder is zero.

Principal Decision Variables:

The principal decision variables are defined below:

– $x_{jk} = \begin{cases} 1 & \text{if job } j \text{ is assigned to start at slot } k \\ 0 & \text{otherwise}, \quad \forall j, k. \end{cases}$

– $y_r = \begin{cases} 1 & \text{if FPGA } r \text{ is utilized} \\ 0 & \text{otherwise}, \quad r = 1, ..., m. \end{cases}$

Auxiliary Decision Variables:

The following auxiliary variables are defined based on the x_{jk}-variables:

– s_j = time-slot at which the processing of job j starts.

– f_j = time-slot at which the processing of job j ends.

Key Sets:

We define certain key sets based on individual job processing times (which could, in general, be CPU- and FPGA-dependent), reconfiguration times (which could again be FPGA-dependent), job release/availability times, and job due-dates:

– $S_j \equiv$ {slots k: $x_{jk} = 1$ is a possible decision based on release, due-date, processing, and reconfiguration times}, $\forall j = 1, ..., n$.

Observe that we may express S_j as

$S_j = \{k : k \in \{1, ..., (m+1)s\}, lb_j \le k \bmod^+(s) \le d_j - \delta_{jr(k)} + 1\}, \forall j = 1, ..., n.$

– $S_{jr} = \{k \in S_j :$ slot k is associated with resource $r\}, \forall j = 1, ..., n, r = 0, ..., m.$

– $J_k = \{(j, \ell) : j \in \{1, ..., n\}, \ell \in S_j$, and $x_{j\ell} = 1$ would imply that slot k would be occupied by the reconfiguration/processing of job $j\}$, $\forall k = 1, ..., (m+1)s$.

Note that J_k can be expressed as

$J_k = \{(j, \ell) : j \in \{1, ..., n\}, \ell \in S_j, r(\ell) = r(k), \ell \bmod^+(s) \le k \bmod^+(s) \le \ell \bmod^+(s) + \delta_{jr(\ell)}\}.$

– $R_t = \{(j, \ell) : j \in \{1, ..., n\}, \ell \in S_j$, and $x_{j\ell} = 1$ would imply that during the time-slot t, the controller is busy performing a reconfiguration$\}$, $\forall t = 1, ..., s - \Delta$, where $\Delta = \min\limits_{j,r}\{\delta_{jr} - \pi_{jr}\}$.

We can formally state R_t as

$R_t = \{(j, \ell) : j \in \{1, ..., n\}, \ell \in S_j, \ell \ge s + 1, \ell \bmod^+(s) \le t \le \ell \bmod^+(s) + \pi_{jr(\ell)}\}.$

Note that $R_t \equiv \emptyset$ for $t = s - \Delta + 1, ..., s$.

Cost Parameters:

– c_{jk} = cost of commencing the operation of job j at the duration corresponding to slot k.

– λ = cost per FPGA used.

Remark 1. The cost of resources (number of FPGAs used) and efficiency (as predicated by the term $\sum\limits_{j=1}^{n} \sum\limits_{k \in S_j} c_{jk} x_{jk}$) compete in the objective function of the mathematical program formulated in Section 3. In addition, observe that it might be desirable to preclude alternative optimal solutions that allow idleness on the available resources. To this end, we may require the hierarchy of cost parameters c_{jk} associated with any job j to be strictly increasing with respect to the time-slot $k \bmod^+ (s)$. □

3 Mathematical Programming Formulation

We present below our proposed mixed-integer 0-1 programming formulation, denoted by **HWSW**, which ascertains the task partitioning and scheduling decisions in order to minimize the total processing and resource costs.

$$\textbf{HWSW}: \text{Minimize} \quad \sum_{j=1}^{n} \sum_{k \in S_j} c_{jk} x_{jk} + \lambda \sum_{r=1}^{m} y_r \tag{1a}$$

$$\text{subject to} \quad \sum_{k \in S_j} x_{jk} = 1, \quad \forall j = 1, ..., n \tag{1b}$$

$$\sum_{(j,\ell) \in J_k} x_{j\ell} \le 1, \quad \forall k = 1, ..., (m+1)s \tag{1c}$$

$$\sum_{(j,\ell) \in R_t} x_{j\ell} \le 1, \quad \forall t = 1, ..., s - \Delta \tag{1d}$$

$$s_j = \sum_{k \in S_j} [k \bmod^+(s)] x_{jk}, \quad \forall j = 1, ..., n \tag{1e}$$

$$f_j = \sum_{k \in S_j} [k \bmod^+(s) + \delta_{jr(k)} - 1] x_{jk}, \quad \forall j = 1, ..., n \tag{1f}$$

$$f_{j_1} + 1 \le s_{j_2}, \quad \forall (j_1, j_2) \in P \tag{1g}$$

$$y_r \ge \sum_{k \in S_{jr}} x_{jk}, \quad \forall j = 1, ..., n, \forall r = 1, ..., m \tag{1h}$$

$$1 \ge y_1 \ge y_2 \ge ... \ge y_m \ge 0 \tag{1i}$$

$$\sum_{j=1}^{n} \sum_{k \in S_{jr}} x_{jk} \ge \sum_{j=1}^{n} \sum_{k \in S_{j,r+1}} x_{jk}, \quad \forall r = 1, ..., m-1 \tag{1j}$$

$$x \text{ binary}, y \text{ continuous}. \tag{1k}$$

The objective function (1a) seeks to minimize the total processing and resource costs. Constraint (1b) requires each job to be feasibly scheduled on either the CPU or on an FPGA. Constraint (1c) asserts that no resource can be processing more than one job simultaneously during any associated slot. Likewise, Constraint (1d) enforces the restriction that the controller can be reconfiguring at most one job at any point in time. Note that whenever $x_{jk} = 1$

for some job $j \in \{1,...,n\}$, and $k \in S_j$, where slot k corresponds to FPGA r, say, then it is assumed that job j starts its reconfiguration by the controller on FPGA r at the time corresponding to the beginning of slot k, after which it immediately proceeds to be processed by FPGA r. Constraints (1e) and (1f) state the definitional identities for the start and finish time-slots for each job j in terms of the x-variables, and Constraint (1g) represents the precedence relationships. Constraint (1h), along with the second objective term, invokes that $y_r = 1$ if and only if some job is processed on FPGA r, and is zero otherwise, even when restricted to be a continuous variable on [0, 1]. Constraints (1i) and (1j) attempt to defeat the inherent symmetry in the problem with respect to the FPGAs, assuming that the FPGAs are identical with respect to processing times. (Note that if there are subgroups of identical FPGAs, then these types of constraints can be incorporated within each such subgroup.) Specifically, Constraint (1i) requires that the lower-indexed FPGAs be utilized first, and more importantly, Constraint (1j) attempts to impart an identity to the utilized FPGAs by imposing the hierarchy that FPGA r should process at least as many jobs as FPGA $r+1$. Without such hierarchical constraints, the inherent symmetry in the problem can hopelessly mire the solution process by requiring it to search among symmetric reflections of essentially the same sets of solutions (see Sherali and Smith [17]). Finally, (1k) represents the logical restrictions on the variables, where the y-variables would automatically turn out to be binary-valued at optimality, even when permitted to be continuous variables on the interval [0, 1].

The model is a linear mixed-integer 0-1 program (MIP), which can be solved using a commercial solver to any desired percentage of optimality.

Remark 2. It is possible to accommodate different alternative objective functions of practical interest within this modeling framework involving the makespan, resource usage (number of FPGAs, durations of usage of FPGAs, etc.), and the completion times of jobs, as desired. □

4 Computational Experience

In this section, we begin by delineating the data generation scheme for constructing random, small- to moderately-sized test instances. Next, we present our computational experience to test the efficiency of the proposed mathematical programming formulation. Our proposed formulation was coded in AMPL and solved using CPLEX 10.1 on a Dell Precision 650 workstation having a Xeon(TM) CPU 2.40 GHz processor and 1.50 GB of RAM.

4.1 Data generation

To demonstrate the usefulness of the optimization scheduling model, we have used randomly simulated, realistic graphs along with the associated data. Although the resulting test cases do not pertain to actual hardware data, they simulate what one might expect in practice.

In our test-bed, the number of jobs n was selected to be 10, 20, or 30, and the number of potential FPGAs, m, was specified to ensure the feasibility of the resulting instance upon generating the different processing times and key sets.

Random Parameters:

- The precedence relationships between the tasks were randomly generated according to the following scheme. Given $j_2 \in \{2,...,n\}$, and for all $j_1 \in \{1,...,j_2-1\}$, we generated $\varphi_{j_1 j_2}$ using a uniform distribution over the range [0, 1]. For some threshold ρ, if $\varphi_{j_1 j_2} > \rho$, then the arc (j_1, j_2) was added to P, that is, task j_1 was required to be a predecessor of task j_2. In our scheme, we took $\rho = 0.75$, which induced a desired density of the precedence arcs in the task graph. Also, redundant arcs were suppressed from the set P by invoking transitivity in the precedence relationships. That is, if the arc (j_β, j_γ) was generated while there also exists an alternative path from j_β to j_γ in the precedence graph, then the direct arc (j_β, j_γ) is redundant, and was consequently deleted from the set P.

- The δ_{j0}-parameters and the release dates, α_j, were generated using a uniform discrete distribution over the sets {1,...,10} and {0,...,15}, respectively.

- Following a scheme similar to that employed by Ali and Das [1], we took the reconfiguration times π_{jr} to be given by $\lfloor 0.015\zeta_j \rfloor$, where ζ_j was randomly generated using a uniform distribution over the range [0, 200], and where $\lfloor\cdot\rfloor$ denotes the rounding-down operation.

- We set $d_j = \lfloor 1.3lb_j + \theta_j \rfloor$, where θ_j is a randomly generated value using a uniform distribution over the interval $[\bar{\delta} - \tau - \frac{\Lambda}{2}, \bar{\delta} - \tau + \frac{\Lambda}{2}]$, and where $\bar{\delta}$ is the average processing time over the CPU, and τ and Λ are parameters that influence the tightness of the due-dates. Here, the term $[\bar{\delta} - \tau - \frac{\Lambda}{2}, \bar{\delta} - \tau + \frac{\Lambda}{2}]$ is based on Fisher's method [8]; we took $\tau = 0.2$ and $\Lambda = 1$ in our experiments.

- The processing costs were computed as $c_{jk} = \lfloor \aleph_j \rfloor + k \bmod^{+}(s)$, where $\aleph_j$ was generated using a uniform distribution over the interval [0, 5], and where s was computed as noted below.

Additional Deduced Parameters:

- $\delta_{jr} = \lceil 0.3\delta_{j0} \rceil + \pi_{jr}, \forall j, \forall r \geq 1.$

- $s = \max_{j=1,\dots,n} \{d_j\}.$

- $\lambda = 4 \max_{j=1,\dots,n} \{d_j\}.$

Remark 3. If the reconfiguration times, as well as certain processing times on FPGAs, are fractional, then all time-related parameters may be suitably rescaled to achieve data integrality. This process, however, could entail a significant growth in the size of the problem. An alternative approach would be to round up all fractional processing times (and to round down due-dates, as appropriate). The marginal time amounts that are introduced by this rounding process may be viewed as idleness-buffers on the relevant hardware or software components. By solving the problem instance with such integerized data, we would obtain a heuristic solution to the original problem. Further improvements can be achieved via a routine that shifts operations to the left to eliminate the marginal idleness that has been introduced by this rounding process. □

4.2 Illustrative example

As a prelude, we present below an example to illustrate the problem under investigation and to gain insights into the proposed formulation. Consider the problem instance where the jobs to be processed are related via the precedence graph depicted in Figure 2 and where the associated parameters are provided in Table 1 (with the remaining data being generated as prescribed in Section 4.1). The available resources include a single CPU and one FPGA for potential use. The discrete, time-indexed scheduling horizon has a projected length of $s = 39$ time-slots.

The solution produced by Model HWSW is summarized in Table 1, and is depicted in the Gantt-chart in Figure 3. This small-sized problem instance was solved to optimality in 0.04 seconds. Observe that the slot values k for which $x_{jk} = 1$, as specified in Table 1, indicate indirectly when operations start their processing, as well as the processing components on which these operations are scheduled (by observing the time-slot ranges attributed to each software/hardware component). For instance, both jobs 9 and 10 start at the beginning of time-slot 28 and are completed at the end of time-slot 31. However, since $x_{9,28} = x_{10,67} = 1$, job 9 is scheduled on the CPU, whereas job 10 is processed on FPGA 1.

Job j	α_j	δ_{j0}	π_{j1}	d_j	$k : x_{jk} = 1$	s_j	f_j
1	4	1	1	11	5	5	5
2	1	9	1	9	45	6	9
3	5	4	3	14	10	10	13
4	0	6	1	23	14	14	19
5	3	9	2	24	53	14	18
6	5	2	3	29	20	20	21
7	14	10	3	29	61	22	27
8	1	7	3	39	71	32	37
9	2	4	2	38	28	28	31
10	4	7	1	39	67	28	31

Table 1: Data and results for an illustrative example

4.3 Computational results

Table 2 summarizes the results obtained for instances having $n = 10$ and 20. The first column of this table specifies the instance number as well as the number of jobs and the maximum number of FPGAs involved. The second column provides the length of the scheduling horizon (number of time-slots). In the next three columns, we present the results obtained for solving the continuous or linear programming (LP) relaxation of Model HWSW, denoted $\overline{\textbf{HWSW}}$, by allowing the x-variables to assume continuous values between 0 and 1. The optimal objective value of the continuous relaxation ($\nu(\overline{\text{HWSW}})$), the ensuing computational time in seconds, and the % optimality gap, are reported for each instance. We define the % Gap as $= 100\frac{\nu(\text{HWSW})-\nu(\overline{\text{HWSW}})}{\nu(\text{HWSW})}$. The final two columns relate to solving Problem HWSW to optimality. Table 2 reveals that for these instances having up to 20 jobs, optimal solutions were obtained within manageable times.

Table 3 provides the results for the more challenging 30-job problem instances. Here, in addition to solving the problem to optimality, we demonstrate the effectiveness of employing two heuristics to derive good quality feasible solutions in a relatively timely fashion. In Table 3, in addition to the LP solution, we report the first MIP solution produced by CPLEX during its branch-and-bound (B&B) exploration, as well the best available MIP solution that the solver could obtain within a specified computational limit of 300 CPU seconds. We compare these two heuristic approaches to solving the problem to optimality by reporting the percentage deviation (% Dev.) between the heuristic solution value from the optimal solution value.

Whereas the LP relaxation objective values for $n = 10$ were particularly tight (within an average optimality gap of 0.7% as seen in Table 2), the LP relaxation for larger problem instances ($n \in \{20, 30\}$ over Tables 2 and 3) exhibited an average optimality gap of 19.5%. As a consequence, these larger problem instances required a significant amount of branching operations within the B&B algorithm employed by the solver. However, it is worthwhile mentioning that by considering the first MIP solution obtained by the solver, or by imposing a time-bound (of 300 CPU seconds) on the computational effort, the resulting heuristic solutions respectively sacrificed only 2% and 0.18% of optimality on average for $n = 30$, while achieving an average computational savings of 98.6% and 93.8%, respectively, as compared with determining optimal solutions. This indicates that near-optimal solutions are typically identified at early stages of the B&B exploration and, hence, motivates the use of such B&B-based heuristic approaches for larger problem instances. It is also important to highlight that the efficiency of such B&B-based heuristic approaches is predicated on the integration of a rounding heuristic scheme and a relaxation-induced neighborhood search (RINS) heuristic that is implemented within CPLEX. At a frequency that the user may control (based on the difficulty and the size of the test instance under investigation), the solver triggers the rounding scheme and/or the RINS heuristic at any node of the B&B tree in an attempt to identify a good quality MIP solution. In our quest for the first MIP solution and the best MIP solution within 300 CPU seconds, the rounding scheme and the RINS heuristic were triggered by the solver at every node explored in the B&B tree.

5 Conclusions

We have proposed a novel formulation for partitioning and scheduling precedence-related jobs on both hardware and software components over a time-indexed horizon. Our model effectively captures the nonpreemption assumption, the precedence constraints, the inviolable due-date restrictions, and the processing times required over hardware and software components. Computational experience reported using randomly generated test instances reveals that optimal solutions can be computed (for $n \leq 20$) within about 20 seconds. For $n = 30$, we demonstrated the effectiveness of two branch-and-bound-based heuristic approaches in producing near-optimal solutions. In particular, using the branch-and-bound algorithm of CPLEX 10.1 to output the first MIP solution and the best MIP solution within a timelimit of 300 CPU seconds, the resulting heuristic solutions respectively sacrificed only 2% and 0.18% of optimality on average, while achieving an average computational savings of 98.6% and 93.8%, respectively, as compared with determining optimal solutions. Thus, we recommend the use of such heuristic approaches for large instances in order to obtain near-optimal solutions with manageable effort. Also, we have focused our attention on minimizing the total processing and resource costs. However, the proposed model is flexible enough to accommodate different alternative objective functions of practical interest within this same modeling framework, which involve the makespan, resource usage (number of FPGAs, durations of usage of FPGAs, etc.), and the completion times of jobs, as desired.

Acknowledgement

This work has been partially supported by the *National Science Foundation* under Grant CMMI-0552676.

References

[1] Ali, F. M. and Das A. S. (2004), Hardware-software co-synthesis of hard real-time systems with reconfigurable FPGAs, *Computers and Electrical Engineering*, 471-489.

[2] Arato P., Juhasz S., Mann Z. A., Orban A. and Papp, D. (2003), Hardware-software partitioning in embedded system design, *IEEE International Symposium on Intelligent Signal Processing*, 197-202.

[3] Bender, A. (1996), Design of an optimal loosely coupled heterogeneous multiprocessor system, *Proceedings of European Design and Test Conference*, 275-281.

[4] Brogioli, M., Radosavljevic, P. and Cavallaro, J. R. (2006), A general hardware/software co-design methodology for embedded signal processing and multimedia workloads, *Fortieth Asilomar Conference on Signals, Systems and Computers*, 1486-1490.

[5] Dasu, A. and Panchanathan, S. (2001), Reconfigurable media processing, *Proceedings of International Conference on Information Technology: Coding and Computing*. 2-4 April 2001, 300 - 304.

[6] Dittmann, F., Gotz, M. and Rettberg, A. (2007), Model and methodology for the synthesis of heterogeneous and partially reconfigurable systems, *Parallel and Distributed Processing Symposium*, 2007, IPDPS 2007, IEEE International, 26-30 March 2007, 1-8.

[7] Edwards, M.D. and Forrest, J. (1995), Hardware/software partitioning for performance enhancement, *IEEE Colloquium on Partitioning in Hardware-Software Codesigns*, 2, 1-5.

[8] Fisher, M. L. (1976), A dual algorithm for the one machine scheduling problem, *Mathematical Programming*, 11, 229-251.

[9] Giovanni, D.M. and Rajesh, K. G. (1997), Hardware/software co-design, *Proceedings of the IEEE*, 85(3), 349-365.

[10] Haubelt, C., Otto, S., Grabbe C. and Teich, J. (2005), A system-level approach to hardware reconfigurable systems, *Proceedings of the Design Automation Conference ASP-DAC 2005*, 298-301.

[11] Jeong, B., Yoo, S., Lee, S. and Choi, K. (2000), Hardware-software cosynthesis for run-time incrementally reconfigurable FPGAs, *Proceedings of the Design Automation Conference ASP-DAC 2000*, 169-174.

[12] Juan P. C., David, S., Onassis, C. and Alvaro, S. (2000), Pipelining-based tradeoffs for hardware/software codesign of multimedia systems, *8th Euromicro Workshop on Parallel and Distributed Processing*, 383-390.

[13] Liu, H. and Wong, D. F. (1998), Integrated partitioning and scheduling for hardware/software co-design, *Proceedings of the International Conference on Computer Design*, 609-614.

[14] Niemann, R. and Marwedel, P. (1997), An algorithm for hardware/software partitioning using mixed integer linear programming, *Design Automation for Embedded Systems*, 2(2), 165-193.

[15] Pinedo, M. (1995), *Scheduling: Theory, Algorithms and Systems*, Prentice-Hall, NJ.

[16] Saul, J. M. (1999), Hardware/software codesign for FPGA-based systems, *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences*.

[17] Sherali, H. D. and Smith, J. C. (2001), Improving discrete model representations via symmetry considerations, *Management Science*, 47, 1396-1407.

[18] Shin, Y. and Choi, K. (1997), Enforcing schedulability of multi-task systems by hardware-software codesign, *International Workshop on Hardware/Software Co-Design*, 3-7.

[19] Sipper, M. and Sanchez, E. (2000), Configurable chips meld software and hardware, *IEEE Computer*, 33(1), 120-121.

[20] Takeuchi, Y., Shibata, K. and Kunieda, H. (1994), Codesign methodology on programmable hardware and software system, *IEEE Asia-Pacific Conference on Circuits and Systems*, 182-187.

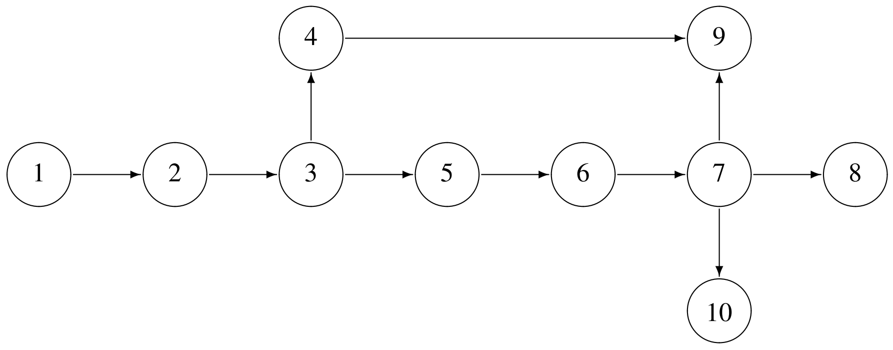

Figure 2: Task precedence graph for the illustrative example involving 10 tasks

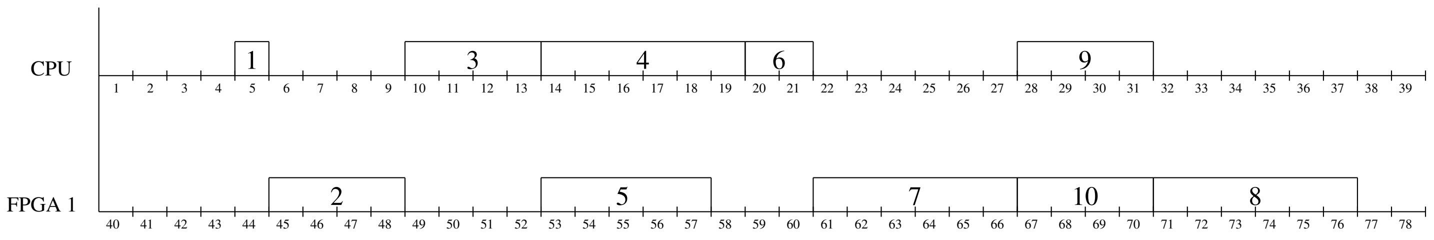

Figure 3: Gantt chart for an illustrative example involving 10 tasks and one FPGA

Instance	s	**LP Solution**			**MIP solution**	
#, (n, m)		ν(**HWSW**)	**Time (s)**	**% Gap**	ν(**HWSW**)	**Time (s)**
1, (10,3)	42	387	0.05	1.5	393	0.35
2, (10,3)	69	413	0.09	0	413	0.09
3, (10,3)	59	363.8	0.11	1.6	370	0.20
4, (10,3)	71	431	0.12	0.2	432	0.14
5, (10,3)	62	396	0.09	0.2	397	0.12
6, (20,2)	40	639.66	0.12	20.2	803	0.39
7, (20,2)	44	649.30	0.11	22.4	838	1.45
8, (20,2)	58	829.14	0.14	23.0	1079	17.12
9, (20,2)	72	1008	0.10	22.2	1296	0.54
10, (20,2)	63	841	0.17	23.5	1100	20.93

Table 2: Performance of Model HWSW for instances having $n = 10$ and 20

Instance	s	**LP Solution**			**First MIP**			**MIP within 300 s**		**Optimal MIP**	
#, (n, m)		ν(**HWSW**)	**Time (s)**	**% Gap**	ν(**HWSW**)	**Time (s)**	**% Dev.**	ν(**HWSW**)	**% Dev.**	ν(**HWSW**)	**Time (s)**
11, (30,2)	77	1444.7	0.18	17.9	1769	6.3	0.4	1761	0	1761	10.8
12, (30,2)	74	1228.11	0.21	20.0	1571	17.4	2.2	1537	0	1537	103.93
13, (30,2)	57	1166.49	0.23	17.7	1484	21.6	4.5	1433	0.9	1419	6647.9
14, (30,2)	92	1175.98	0.39	19.2	1485	165.6	1.9	1466	≈ 0	1456	2819.3
15, (30,2)	81	1639.24	0.39	9.6	1836	17.6	1.1	1815	0	1815	6943.7

Table 3: Performance of Model HWSW for instances having $n = 30$