Scheduling Monotone Interval Orders on Typed Task Systems

Benoît Dupont de Dinechin
STMicroelectronics STS/CEC
12, rue Jules Horowitz - BP 217. F-38019 Grenoble
benoit.dupont-de-dinechin@st.com

Abstract

We present a modification of the Leung-Palem-Pnueli parallel processors scheduling algorithm and prove its optimality for scheduling monotone interval orders with release dates and deadlines on Unit Execution Time (UET) typed task systems in polynomial time. This problem is motivated by the relaxation of Resource-Constrained Project Scheduling Problems (RCPSP) with precedence delays and UET operations.

Introduction

Scheduling problems on *typed task systems* (Jaffe 1980) generalize the parallel processors scheduling problems by introducing k types $\{\tau_r\}_{1\leq r\leq k}$ and $\sum_{1\leq r\leq k} m_r$ processors with m_r processors of type τ_r. Each operation O_i has a type $\tau_i \in \{\tau_r\}_{1\leq r\leq k}$ and may only execute on processors of type τ_i. We denote typed task systems with $\Sigma^k P$ in the α-field of the $\alpha|\beta|\gamma$ scheduling problem denotation (Brucker 2004).

Scheduling typed task systems is motivated by two main applications: resource-constrained scheduling in high-level synthesis of digital circuits (Chaudhuri, Walker, & Mitchell 1994), and instruction scheduling in compilers for VLIW processors (Dupont de Dinechin 2004). In high-level synthesis, execution resources correspond to the synthesized functional units, which are partitioned by classes such as adder or multiplier with a particular bit-width. Operations are typed by these classes and may have non-unit execution time. In compiler VLIW instruction scheduling, operations usually have unit execution time (UET), however on most VLIW processors an operation requires several resources for execution, like in the Resource-Constrained Project Scheduling Problems (RCPSP) (Brucker *et al.* 1999). In both cases, the pipelined implementation of functional units yield scheduling problems with precedence delays, that is, the time required to produce a value is larger than the minimum delay between two activations of a functional unit.

We are aware of the following work in the area of typed task systems. Jaffe (Jaffe 1980) introduces them to formalize instruction scheduling problems that arise in high-performance computers and data-flow machines, and studies the performance bounds of list scheduling. Jansen (Jansen 1994) gives a polynomial time algorithm for problem $\Sigma^k P|intOrder; p_i = 1|C_{max}$, that is, scheduling interval-ordered typed UET operations. Verriet (Verriet 1998) solves problem $\Sigma^k P|intOrder; c_i^j = 1; p_i = 1|C_{max}$ in polynomial time, that is, interval-ordered typed UET operations subject to unit communication delays.

Interval orders are a class of precedence graphs where UET scheduling on parallel processors is polynomial-time, while non-UET scheduling on 2 processors is strongly NP-hard (Papadimitriou & Yannakakis 1979). In particular, Papadimitriou and Yannakakis solve $P|intOrder; p_i = 1|C_{max}$ in polynomial-time. Scheduling interval orders with communication delays on parallel processors is also polynomial-time, as the algorithm by Ali and El-Rewini (Ali & El-Rewini 1992) solves $P|intOrder; c_i^j = 1; p_i = 1|C_{max}$. Verriet (Verriet 1996) further proposes a deadline modification algorithm that solves $P|intOrder; c_i^j = 1; r_i; p_i = 1|L_{max}$ in polynomial-time.

Scheduling interval orders with precedence delays on parallel processors was first considered by Palem and Simons (Palem & Simons 1993), who introduced monotone interval orders and solve $P|intOrder(mono\ l_i^j); p_i = 1|L_{max}$ in polynomial-time. This result is generalized by Leung-Palem-Pnueli algorithm (Leung, Palem, & Pnueli 2001).

In the present work, we modify the algorithm of Leung, Palem and Pnueli (Leung, Palem, & Pnueli 2001) in order to solve $\Sigma^k P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|-$ feasibility problems in polynomial time. The resulting algorithm thus operates on typed tasks, allows precedence delays, and handles release dates and deadlines. Thanks to these properties, it provides useful relaxations of the RCPSP with UET operations and precedence delays.

The Leung-Palem-Pnueli algorithm (Leung, Palem, & Pnueli 2001) is a parallel processors scheduling algorithm based on deadline modification and the use of lower modified deadline first priority in a Graham list scheduling algorithm. The Leung-Palem-Pnueli algorithm (LPPA) solves the following feasibility problems in polynomial time:

- $1|prec(l_i^j \in \{0,1\}); r_i; d_i; p_i = 1|-$
- $P2|prec(l_i^j \in \{-1,0\}); r_i; d_i; p_i = 1|-$
- $P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|-$
- $P|inTree(l_i^j = l); d_i; p_i = 1|-$

Here, the l_i^j are precedence delays with $p_i + l_i^j \geq 0$.

Presentation is as follows. In the first section, we extend the $\alpha|\beta|\gamma$ scheduling problem denotation and we discuss the Graham list scheduling algorithm (GLSA) for typed task systems. In the second section, we present our modified Leung-Palem-Pnueli algorithm (LPPA) and prove its optimality for scheduling monotone interval orders with release dates and deadlines on UET typed task systems in polynomial time. In the third section, we discuss the application of this algorithm to VLIW instruction scheduling.

Deterministic Scheduling Background

Machine Scheduling Problem Denotation

In parallel processors scheduling problems, an operation set $\{O_i\}_{1\leq i\leq n}$ is processed on m identical processors. Each operation O_i requires the exclusive use of one processor for p_i time units, starting at its *schedule date* σ_i. Scheduling problems may involve *release dates* r_i and *due dates* d_i. This constrains the schedule date σ_i of operation O_i as $\sigma_i \geq r_i$ and there is a penalty whenever $C_i > d_i$, with C_i the *completion date* of O_i defined as $C_i \stackrel{\text{def}}{=} \sigma_i + p_i$. For problems where $C_i \leq d_i$ is mandatory, the d_i are called *deadlines*.

A *precedence* $O_i \prec O_j$ between two operations constrains the schedule with $\sigma_i + p_i \leq \sigma_j$. In case of *precedence delay* l_i^j between O_i and O_j, the scheduling constraint becomes $\sigma_i + p_i + l_i^j \leq \sigma_j$. The *precedence graph* has one arc (O_i, O_j) for each precedence $O_i \prec O_j$. Given an operation O_i, we denote $\mathsf{succ}O_i$ the set of direct successors of O_i and $\mathsf{pred}O_i$ the set of direct predecessors of O_i in the precedence graph. The set $\mathsf{indep}O_i$ contains the operations that are not connected to O_i in the undirected precedence graph.

Given a scheduling problem over operation set $\{O_i\}_{1\leq i\leq n}$ with release dates $\{r_i\}_{1\leq i\leq n}$ and deadlines $\{d_i\}_{1\leq i\leq n}$, the *precedence-consistent release dates* $\{r_i^+\}_{1\leq i\leq n}$ are recursively defined as $r_i^+ \stackrel{\text{def}}{=} \max(r_i, \max_{O_j\in\mathsf{pred}O_i}(r_j^+ + p_j + l_j^i))$. Likewise, the *precedence-consistent deadlines* $\{d_i^+\}_{1\leq i\leq n}$ are recursively defined as $d_i^+ \stackrel{\text{def}}{=} \min(d_i, \min_{O_j\in\mathsf{succ}O_i}(d_j^+ - p_j - l_i^j))$.

Machine scheduling problems are denoted by a triplet $\alpha|\beta|\gamma$ (Brucker 2004), where α describes the processing environment, β specifies the operation properties and γ defines the optimality criterion. Values of α, β, γ include:

- α : 1 for a single processor, P for parallel processors, Pm for the given m parallel processors. We denote typed task systems with k types by $\Sigma^k P$.
- β : r_i for release dates, d_i for deadlines (if $\gamma = -$) or due dates, $p_i = 1$ for Unit Execution Time (UET) operations.
- γ : $-$ for the feasibility, C_{max} or L_{max} for the minimization of these objectives.

The *makespan* is $C_{max} \stackrel{\text{def}}{=} \max_i C_i$ and the *maximum lateness* is $L_{max} \stackrel{\text{def}}{=} \max_i L_i : L_i \stackrel{\text{def}}{=} C_i - d_i$. The meaning of the additional β fields is:

$prec(l_i^j)$ Precedence delays l_i^j, assuming $l_i^j \geq -p_i$.

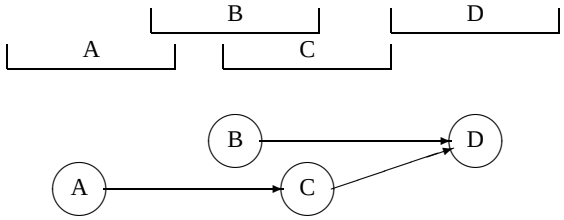

Figure 1: Set of intervals and the corresponding interval order graph.

$prec(l_i^j = l)$ All the precedence delays l_i^j equal l.

$inTree$ The precedence graph is an in-tree.

$intOrder(mono\ l_i^j)$ The precedence graph weighted by $w(O_i, O_j) \stackrel{\text{def}}{=} p_i + l_i^j$ is a monotone interval order.

An *interval order* is the transitive orientation of the complement of an interval graph (Papadimitriou & Yannakakis 1979) (see Figure 1). The important property of interval orders is that given any two operations O_i and O_j, either $\mathsf{pred}O_i \subseteq \mathsf{pred}O_j$ or $\mathsf{pred}O_j \subseteq \mathsf{pred}O_i$ (similarly for successors). This is easily understood by referring to the underlying intervals that define the interval order. Adding or removing operations without predecessors and successors to an interval order is still an interval order. Also, interval orders are transitively closed, that is, any transitive successor (predecessor) must be a direct successor (predecessor).

A *monotone interval order* graph (Palem & Simons 1993) is an interval order whose precedence graph (V, E) is weighted with a non-negative function w on the arcs such that, given any $(O_i, O_j), (O_i, O_k) \in E$: $\mathsf{pred}O_j \subseteq \mathsf{pred}O_k \Rightarrow w(O_i, O_j) \leq w(O_i, O_k)$. Monotone interval orders are motivated by the application of interval orders properties to scheduling problems with precedence delays.

Indeed, in scheduling problems with interval orders, the precedence arc weight considered between any two operations O_i and O_j is $w(O_i, O_j) \stackrel{\text{def}}{=} p_i$ with p_i the processing time of O_i. In case of monotone interval orders, the arc weights are $w(O_i, O_j) \stackrel{\text{def}}{=} p_i + l_i^j$ with l_i^j the precedence delay between O_i and O_j. An interval order graph where all arcs leaving any given node have the same weight is obviously monotone, so interval order precedences without precedence delays imply monotone interval order graphs.

Graham List Scheduling Algorithm Extension

The Graham list scheduling algorithm (GLSA) is a classic scheduling algorithm where the time steps are considered in non-decreasing order. For each time step, if a processor is idle, the highest priority operation available at this time is scheduled An operation is available if the current time step is not earlier than the release date and all direct predecessors have completed their execution early enough to satisfy the precedence delays. On typed task systems, the operation type must match the type of an idle processor.

The GLSA is optimal for $P|r_i; d_i; p_i = 1|-$ and $P|r_i; p_i = 1|L_{max}$ when using the earliest deadlines (or due

dates) d_i first as priority (Brucker 2004) (Jackson's rule). This property directly extends to typed task systems:

Theorem 1 *The GLSA with Jackson's rule optimally solves* $\Sigma^k P|r_i;d_i;p_i=1|-$ *and* $\Sigma^k P|r_i;p_i=1|L_{max}$.

Proof: In typed task systems, operations are partitioned by processor type. In problem $\Sigma^k P|r_i;d_i;p_i=1|-$ (respectively $\Sigma^k P|r_i;p_i=1|L_{max}$), there are no precedences between operations. Therefore, optimal scheduling can be achieved by considering operations and processors of each type independently. For each type, the problem reduces to $P|r_i;d_i;p_i=1|-$ (respectively $P|r_i;p_i=1|L_{max}$), which is optimally solved with Jackson's rule. □

In this work, we allow precedences delays $l_i^j=-p_i \Rightarrow \sigma_i \leq \sigma_j$, that is, precedences with zero start-start time lags. Thus we extend the GLSA as follows: in cases of available operations with equal priorities, schedule first the earliest operations in the precedence topological sort order.

The Modified Leung-Palem-Pnueli Algorithm

Algorithm Description

The Leung-Palem-Pnueli algorithm (LPPA) is similar to classic UET scheduling algorithms on parallel processors like Garey & Johnson (Garey & Johnson 1976), in that it uses a lower modified deadlines first priority in a GLSA. Given a scheduling problem with deadlines $\{d_i\}_{1\leq i\leq n}$, *modified deadlines* $\{d'_i\}_{1\leq i\leq n}$ are such that $\forall i \in [1,n]$: $\sigma_i+p_i\leq d'_i\leq d_i$ for any schedule $\{\sigma_i\}_{1\leq i\leq n}$. The distinguishing feature of the LPPA is the computation of its modified deadlines, which we call *fixpoint modified deadlines*[1].

Precisely, the LPPA defines a *backward scheduling problem* denoted $B(O_i,S_i)$ for each operation O_i. An *optimal backward scheduling* procedure computes the latest possible schedule date σ'_i of operation O_i in each $B(O_i,S_i)$. Optimal backward scheduling of $B(O_i,S_i)$ is used to update the current modified deadline of O_i as $d'_i \leftarrow \sigma'_i+p_i$. This process of deadline modification is iterated over all problems $B(O_i,S_i)$ until a fixpoint of the modified deadlines $\{d^*_i\}_{1\leq i\leq n}$ is reached (Leung, Palem, & Pnueli 2001).

We modify the Leung-Palem-Pnueli algorithm (LPPA) to compute the fixpoint modified deadlines $\{d^*_i\}_{1\leq i\leq n}$ by executing the following procedure:

(i) Compute the precedence-consistent release dates $\{r_i^+\}_{1\leq i\leq n}$, the precedence-consistent deadlines $\{d_i^+\}_{1\leq i\leq n}$ and initialize the modified deadlines $\{d'_i\}_{1\leq i\leq n}$ with the precedence-consistent deadlines.

(ii) For each operation O_i, define the backward scheduling problem $B(O_i,S_i)$ with $S_i \stackrel{\text{def}}{=} \mathsf{succ}O_i \cup \mathsf{indep}O_i$.

(1) Let O_i be the current operation in some iteration over $\{O_i\}_{1\leq i\leq n}$.

(2) Compute the optimal backward schedule date σ'_i of O_i by optimal backward scheduling of $B(O_i,S_i)$.

(3) Update the modified deadline of O_i as $d'_i \leftarrow \sigma'_i+1$.

(4) Update the modified deadlines of each $O_k \in \mathsf{pred}O_i$ with $d'_k \leftarrow \min(d'_k, d'_i-1-l_k^i)$.

(5) Go to (1) until a fixpoint of the modified deadlines $\{d'_i\}_{1\leq i\leq n}$ is reached.

In our modified LPPA, we define the *backward scheduling problem* $B(O_i,S_i)$ as the search for a set of dates $\{\sigma'_j\}_{O_j\in\{O_i\}\cup S_i}$ that satisfy:

(a) $\forall O_j \in S_i : O_i \prec O_j \Rightarrow \sigma'_i+1+l_i^j \leq \sigma'_j$

(b) $\forall t \in \mathbb{N}, \forall r \in [1,k] : |\{O_j \in \{O_i\}\cup S_i \wedge \tau_j = r \wedge \sigma'_j = t\}| \leq m_r$

(c) $\forall O_j \in \{O_i\}\cup S_i : r_j^+ \leq \sigma'_j < d'_j$

Constraints (a) state that only the precedences between O_i and its direct successors are kept in the backward scheduling problem $B(O_i,S_i)$. Constraints (b) are the resources limitations of typed task systems with UET operations. Constraints (c) ensure that operations are backward scheduled within the precedence-consistent release dates and the current modified deadlines. An *optimal backward schedule* for O_i maximizes σ'_i in $B(O_i,S_i)$.

Let $\{r_j^+\}_{1\leq i\leq n}$ be the precedence-consistent release dates and $\{d'_j\}_{1\leq i\leq n}$ be the current modified deadlines. The simplest way to find the optimum backward schedule date of O_i in $B(O_i,S_i)$ is to search for the latest $s\in[r_i^+,d'_i-1]$ such that the constrained backward scheduling problem $(\sigma'_i=s)\wedge B(O_i,S_i)$ is feasible. Even though each such constrained problem can be solved in polynomial time by reducing to some $\Sigma^k P|r_j;d_j;p_j=1|-$ over $\{O_i\}\cup S_i$, optimal backward scheduling of $B(O_i,S_i)$ would require pseudo-polynomial time, as there are up to $d'_i-r_i^+$ constrained backward scheduling problems to solve. Please note that a simple dichotomy search for the latest feasible $s\in[r_i^+,d'_i-1]$ does not work, as $(\sigma'_i=s)\wedge B(O_i,S_i)$ is infeasible does not imply that $(\sigma'_i=s+1)\wedge B(O_i,S_i)$ is infeasible.

In order to avoid the pseudo-polynomial time complexity of optimal backward scheduling, we rely instead on a procedure with two successive dichotomy searches for feasible relaxations of constrained backward scheduling problems, like in the original LPPA. Describing this procedure requires further definitions. Assume $l_i^j=-\infty$ if $O_i \not\prec O_j$. Given a constrained backward scheduling problem $(\sigma'_i\in[p,q])\wedge B(O_i,S_i)$, we define a relaxation $\Sigma^k P|\hat{r}_j;\hat{d}_j;p_j=1|-$ over the operation set $\{O_i\}\cup S_i$ such that:

$$\begin{cases} & \hat{r}_i \stackrel{\text{def}}{=} p \\ & \hat{d}_i \stackrel{\text{def}}{=} q+1 \\ O_j\in S_i \Longrightarrow & \hat{r}_j \stackrel{\text{def}}{=} \max(r_j^+, q+1+l_i^j) \\ O_j\in S_i \Longrightarrow & \hat{d}_j \stackrel{\text{def}}{=} d'_j \end{cases}$$

In other words, the precedences from O_i to each direct successor $O_j\in S_i$ are converted into release dates assuming the release date and deadline of O_i respectively equal p and $q+1$. We call *type 2 relaxation* the resulting scheduling problem $\Sigma^k P|\hat{r}_j;\hat{d}_j;p_j=1|-$ and *type 1 relaxation* this

[1] Leung, Palem and Pnueli call them "consistent and stable modified deadlines".

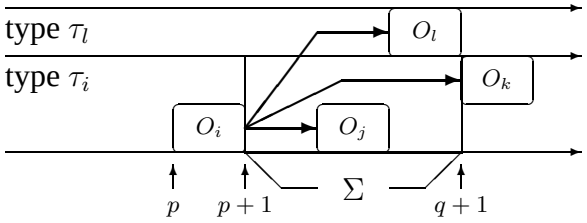

Figure 2: Optimal backward scheduling proof.

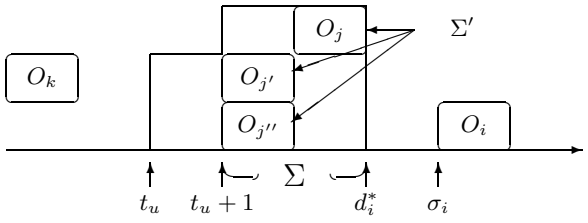

Figure 3: Modified Leung-Palem-Pnueli algorithm proof.

problem when disregarding the resource constraints of O_i. Both type 1 and type 2 relaxations are optimally solved by the GLSA with the earliest $\hat{d}_j$ first priority (Theorem 1). If any relaxation is infeasible, so is the constrained backward scheduling problem $(\sigma'_i \in [p,q]) \wedge B(O_i,S_i)$.

Observe that the type 1 relaxation is increasingly constrained as q increases, independently of the value of p. And for any fixed q, the type 2 relaxation is increasingly constrained as p increases. Therefore, it is correct to explore the feasibility of any of these relaxations using dichotomy search. So the optimal backward scheduling procedure is based on two dichotomy searches as follows.

The first dichotomy search initializes $p = r_i^+$ and $q = d'_i - 1$. Then it proceeds to find the latest q such that the type 1 relaxation is feasible. The second dichotomy search keeps q constant and finds the latest p such that the type 2 relaxation is feasible. Whenever both searches succeed, the optimum backward schedule date of O_i is taken as $\sigma'_i = p$ so the new modified deadline is $d'_i = p+1$. If any dichotomy search fails, $B(O_i,S_i)$ is assumed infeasible.

Algorithm Proofs

Theorem 2 *The optimal backward scheduling procedure computes the latest schedule date σ'_i of O_i among the schedules that satisfy conditions (a), (b), (c) of $B(O_i,S_i)$.*

Proof: The two dichotomy searches are equivalent to linear searches, respectively by increasing q and by increasing p. If no feasible relaxation $\Sigma^k P|\hat{r}_j;\hat{d}_j;p_j=1|-$ exist in any of these linear searches, the backward scheduling problem $B(O_i,S_i)$ is obviously infeasible.

If a feasible relaxation exists in the second linear search, this search yields a backward schedule with $\sigma'_i = p$. Indeed, let $\{\hat{\sigma}_j\}_{O_j\in\{O_i\}\cup S_i}$ be schedule dates for the type 2 relaxation of $(\sigma'_i \in [p,q]) \wedge B(O_i,S_i)$. We have $\hat{\sigma}_i = p$ because the type 2 relaxation of problem $(\sigma'_i \in [p+1,q]) \wedge B(O_i,S_i)$ is infeasible and the only difference between these two relaxations is the release date of O_i. Moreover, the dates $\{\hat{\sigma}_j\}_{O_j\in\{O_i\}\cup S_i}$ satisfy (a), (b), (c). Condition (a) is satisfied from the definition of $\hat{r}_j$ and because $\hat{\sigma}_i = p \leq q$. Conditions (b) and (c) are satisfied by the GLSA.

Let us prove that the backward schedule found by the second search is in fact optimal, that is, there is no $s \in [p+1,q]$ such that problem $(\sigma'_i \in [s,s]) \wedge B(O_i,S_i)$ is feasible. This is obvious if $p = q$, so consider cases where $p < q$. The type 2 relaxation of problem $(\sigma'_i \in [p,q]) \wedge B(O_i,S_i)$ is feasible while the type 2 relaxation of problem $(\sigma'_i \in [p+1,q]) \wedge B(O_i,S_i)$ is infeasible imply there is a set Σ of operations that fill all slots of type τ_i in range $[p+1,q]$ and prevents the GLSA from scheduling of O_i in that range (Figure 2). So $O_j \in \Sigma \Rightarrow \hat{d}_j \leq \hat{d}_i = q+1 \wedge \hat{r}_j \geq p+1$.

Now assume exists some $s \in [p+1,q]$ such that problem $(\sigma'_i \in [s,s]) \wedge B(O_i,S_i)$ is feasible. This imply that problem $(\sigma'_i \in [p+1,s]) \wedge B(O_i,S_i)$ is also feasible. The type 2 relaxation of $(\sigma'_i \in [p+1,s]) \wedge B(O_i,S_i)$ differs from the type 2 relaxation of $(\sigma'_i \in [p+1,q]) \wedge B(O_i,S_i)$ only by the decrease of the release dates $\hat{r}_j$ of some operations $O_j \in S_i$, yet $\hat{r}_j \geq p+1$ as $\hat{r}_j \stackrel{\text{def}}{=} \max(r_j^+, s+1+l_i^j) \geq p+1+1+l_i^j$. As all the operations of Σ must still be scheduled in range $[p+1,q]$ in the type 2 relaxation of $(\sigma'_i \in [p+1,s]) \wedge B(O_i,S_i)$, there is still no scheduling slot for O_i in that range. So problem $(\sigma'_i \in [p+1,s]) \wedge B(O_i,S_i)$ and problem $(\sigma'_i \in [s,s]) \wedge B(O_i,S_i)$ are infeasible. □

Theorem 3 *The modified algorithm of Leung, Palem and Pnueli solves any feasible problem $\Sigma^k P|intOrder(mono\ l_i^j);r_i;d_i;p_i=1|-$.*

Proof: The correctness of this modified Leung-Palem-Pnueli algorithm (LPPA), like the correctness of the original LPPA, is based on two arguments. The first argument is that the fixpoint modified deadlines are indeed deadlines of the original problem. This is apparent, as each backward scheduling problem $B(O_i,S_i)$ is a relaxation of the original scheduling problem and optimal backward scheduling computes the latest schedule date of O_i within $B(O_i,S_i)$ by Theorem 2. Let us call *core* the GLSA that uses the earliest fixpoint modified deadlines first as priorities. The second correctness argument is a proof that the core GLSA does not miss any fixpoint modified deadlines.

Precisely, assume that some O_i is the earliest operation that misses its fixpoint modified deadline d_i^* in the core GLSA schedule. In a similar way to (Leung, Palem, & Pnueli 2001), we will prove that an earlier operation O_k necessarily misses its fixpoint modified deadline d_k^* in the same schedule. This contradiction ensures that the core GLSA schedule does not miss any fixpoint modified deadline. The details of this proof rely on a few definitions and observations illustrated in Figure 3.

Let $r = \tau_i$ be the type of operation O_i. An operation O_j is said *saturated* if $\tau_j = r$ and $d_j^* \leq d_i^*$. Define $t_u < d_i^*$ as the latest time step that is not filled with saturated operations on the processors of type r. If $t_u < 0$, the problem is infeasible, as there are not enough slots to schedule opera-

tions of type r on m_r processors within the deadlines. Else, some scheduling slots of type r at t_u are either empty or filled with operations $O_u : d_u^* > d_i^*$ of lower priority than saturated operations in the core GLSA. Define the operation set $\Sigma \stackrel{\text{def}}{=} \{O_j \text{ saturated} : t_u < \sigma_j < d_i^*\} \cup \{O_i\}$. Define the operation subset $\Sigma' \stackrel{\text{def}}{=} \{O_j \in \Sigma : r_j^+ \leq t_u\}$.

Consider problem $P^k|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|-$. In an interval order, given two operations O_i and O_j, either $\mathsf{pred}O_i \subseteq \mathsf{pred}O_j$ or $\mathsf{pred}O_j \subseteq \mathsf{pred}O_i$. Select $O_{j'}$ among $O_j \in \Sigma'$ such that $|\mathsf{pred}O_j|$ is minimal. As $O_{j'} \in \Sigma'$ is not scheduled at date t_u or earlier by the core GLSA, there must be a constraining operation O_k that is a direct predecessor of operation $O_{j'}$ with $\sigma_k + 1 + l_k^{j'} = \sigma_{j'} > t_u \Rightarrow \sigma_k + 1 > t_u - l_k^{j'}$. Note that O_k can have any type. Operations in $\mathsf{pred}O_{j'}$ are the direct predecessors of all operations $O_j \in \Sigma'$ and no predecessor of $O_{j'}$ is in Σ'. Thus $O_k \notin \Sigma'$ and O_k is a direct predecessor of all operations $O_j \in \Sigma'$.

We call *stable backward schedule* any optimal backward schedule of $B(O_k, S_k)$ where the modified deadlines equal the fixpoint modified deadlines. Since $S_k \stackrel{\text{def}}{=} \mathsf{succ}O_k \cup \mathsf{indep}O_k$, we have $\Sigma \subseteq S_k$. By the fixpoint property, we may assume that a stable backward schedule of $B(O_k, S_k)$ exists. Such stable backward schedule must slot the $m_r(d_i^* - 1 - t_u) + 1$ operations of Σ before d_i^* on m_r processors, so at least one operation $O_j \in \Sigma'$ is scheduled at date t_u or earlier by any stable backward schedule of $B(O_k, S_k)$.

Theorem 2 ensures that optimal backward scheduling of $B(O_k, S_k)$ satisfies the precedence delays between O_k and O_j. Thus $\sigma'_k + 1 + l_k^j \leq t_u$ so $d_k^* - 1 + 1 + l_k^j \leq t_u$. By the monotone interval order property, $\mathsf{pred}O_{j'} \subseteq \mathsf{pred}O_j \Rightarrow w(O_k, O_{j'}) \leq w(O_k, O_j) \Rightarrow 1 + l_k^{j'} \leq 1 + l_k^j \Rightarrow l_k^{j'} \leq l_k^j$ for $O_{j'}$ selected above and $O_j \in \Sigma'$, so $d_k^* \leq t_u - l_k^{j'}$. However in the core GLSA schedule $\sigma_k + 1 > t_u - l_k^{j'}$, so O_k misses its fixpoint modified deadline d_k^*. □

The overall time complexity of this modified LPPA is the sum of the complexity of initialization steps (i-ii), of the number of iterations times the complexity of steps (1-5) and of the complexity of the core GLSA. Leung, Palem and Pnueli (Leung, Palem, & Pnueli 2001) observe that the number of iterations to reach a fixpoint is upper bounded by n^2, a fact that still holds for our modified algorithm. As the time complexity of the GLSA on typed task systems with k types is within a factor k of the time complexity of the GLSA on parallel processors, our modified LPPA has polynomial time complexity.

In their work, Leung, Palem and Pnueli (Leung, Palem, & Pnueli 2001) describe further techniques that enable to lower the overall complexity of their algorithm. The first is a proof that applying optimal backward scheduling in reverse topological order of the operations directly yields the fixpoint modified deadlines. The second is a fast implementation of list scheduling for problems $P|r_i; d_i; p_i = 1|-$. These techniques apply to typed task systems as well.

Table 1: ST200 VLIW processor resource availabilities and operation class resource requirements

Resource	Issue	Memory	Control	Align
Availability	4	1	1	2
ALU	1	0	0	0
ALUX	2	0	0	1
MUL	1	0	0	1
MULX	2	0	0	1
MEM	1	1	0	0
MEMX	2	1	0	1
CTL	1	0	1	1

Application to VLIW Instruction Scheduling

ST200 VLIW Instruction Scheduling Problem

We illustrate VLIW instruction scheduling problems on the ST200 VLIW processor manufactured by STMicroelectronics. The ST200 VLIW processor executes up to 4 operations per time unit with a maximum of one control operation (goto, jump, call, return), one memory operation (load, store, prefetch), and two multiply operations per time unit. All arithmetic operations operate on integer values with operands belonging either to the General Register file (64 × 32-bit) or to the Branch Register file (8 × 1-bit). In order to eliminate some conditional branches, the ST200 VLIW architecture also provides conditional selection instructions. The processing time of any operation is a single time unit ($p_i = 1$), while the precedence delays l_i^j between operations range from -1 to 2 time units.

The resource availabilities of the ST200 VLIW processor and the resource requirements of each operation are displayed in Table 1. The resources are: Issue for the instruction issue width; Memory for the memory access unit; Control for the control unit. An artificial resource Align is also introduced to satisfy some encoding constraints. Operations with identical resource requirements are factored into *classes*: ALU, MUL, MEM and CTL correspond respectively to the arithmetic, multiply, memory and control operations. The classes ALUX, MULX and MEMX represent the operations that require an extended immediate operand. Operations named `LDH`, `MULL`, `ADD`, `CMPNE`, `BRF` belong respectively to classes MEM, MUL, ALU, ALU, CTL.

A sample C program and the corresponding ST200 VLIW processor operations for the inner loop are given in Figure 4. The operations are numbered in their appearance order. In Figure 5, we display the precedence graph between operations of the inner loop of Figure 4 after removing the redundant transitive arcs. As usual in RCPSP, the precedence graph is augmented with dummy nodes O_0 and $O_{n+1} : n = 7$ with null resource requirements. Also, the precedence arcs are labeled with the corresponding start-start time-lag, that is, the values of $p_i + l_j^i$. The critical path of this graph is $O_0 \rightarrow O_1 \rightarrow O_2 \rightarrow O_3 \rightarrow O_7 \rightarrow O_8$ so the makespan is lower bounded by 7.

This example illustrates that null start-start time-lags, or precedence delays $l_j^i = -p_i$, occur frequently in actual VLIW instruction scheduling problems. Moreover, the start-

```
int
prod(int n, short a[], short b) {
   int s=0, i;
   for (i=0;i<n;i++) {
      s += a[i]*b;
   }
   return s;
}
```

```
L?__0_8:
    LDH_1    g131 = 0, G127
    MULL_2   g132 = G126, g131
    ADD_3    G129 = G129, g132
    ADD_4    G128 = G128, 1
    ADD_5    G127 = G127, 2
    CMPNE_6  b135 = G118, G128
    BRF_7    b135, L?__0_8
```

Figure 4: A sample C program and the corresponding ST200 operations

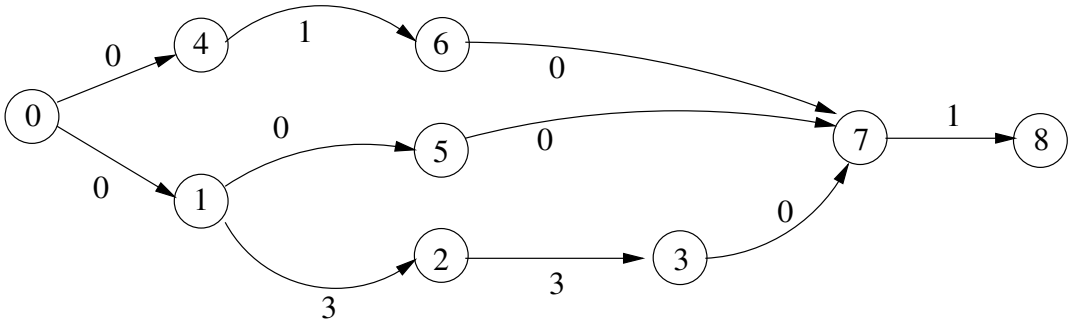

Figure 5: Precedence graph of the inner loop instruction scheduling problem

start time-lags are non-negative, so classic RCPSP schedule generation schemes (Kolisch & Hartmann 1999) (list scheduling) are guaranteed to build feasible (sub-optimal) solutions for these VLIW instruction scheduling problems. In this setting, the main value of VLIW instruction scheduling problem relaxations such as typed task systems is to strengthen the bounds on operation schedule dates including the makespan. Improving bounds benefits scheduling techniques such as solving time-indexed integer linear programming formulations (Dupont de Dinechin 2007).

ST200 VLIW Compiler Experimental Results

We implemented our modified Leung-Palem-Pnueli algorithm in the instruction scheduler of the production compiler for the ST200 VLIW processor family. In order to apply this algorithm, we first relax instances of RCPSP with UET operations and non-negative start-start time-lags to instances of scheduling problems on typed task systems with precedence delays, release dates and deadlines:

- Expand each operation that requires several resources to a chain of sub-operations that use only one resource type per sub-operation. Set the chain precedence delays to -1 (zero start-start time-lags).
- Assign to each sub-operation the release date and deadline of its parent operation.

The result is a UET typed task system with release dates and deadlines, whose precedence graph is arbitrary.

Applying our modified Leung-Palem-Pnueli algorithm to an arbitrary precedence graph implies that optimal scheduling is no longer guaranteed. However, the fixpoint modified deadlines are still deadlines of the UET typed task system considered, as the proof of Theorem 2 does not involve the precedence graph properties. From the way we defined the relaxation to typed task systems, it is apparent that these fixpoint modified deadlines are also deadlines of the original problem (UET RCPSP with non-negative time-lags).

In Table 2, we collect the results of lower bounding the makespan of ST200 VLIW instruction scheduling problems with our modified LPPA for typed task systems. These results are obtained by first computing the fixpoint modified deadlines on the reverse precedence graph, yielding strengthened release dates. The modified LPPA is then applied to the precedence graph with strengthened release dates, and this computes fixpoint modified deadlines including a makespan lower bound. The benchmarks used to extract these results include an image processing program, and the `c-lex` SpecInt program.

The first column of Table 2 identifies the code block that defined the VLIW instruction scheduling problem. Column n gives the number of operations to schedule. Columns Resource, Critical, MLPPA respectively give the makespan lower bound in time units computed with resource use, critical path, and the modified LPPA. The last column ILP gives the optimal makespan as computed by solving a time-indexed linear programming formulation (Dupont de Dinechin 2007). According to this experimental data, there exists cases where using the modified LPPA yields a significantly stronger relaxation than critical path computation.

Summary and Conclusions

We present a modification of the algorithm of Leung, Palem and Pnueli (LPPA) (Leung, Palem, & Pnueli 2001) that schedules monotone interval orders with release dates and deadlines on UET typed task systems (Jaffe 1980) in poly-

Table 2: ST200 VLIW compiler results of the modified Leung-Palem-Pnueli algorithm

Label	n	Resource	Critical	MLPPA	ILP
BB26	41	11	15	19	19
BB23	34	10	14	18	18
BB30	10	3	5	5	5
BB29	16	5	10	10	10
_1_31	34	9	14	18	18
BB9_Short	16	4	10	10	10
BB22	16	4	10	10	10
LAO021	22	6	6	7	7
LAO011	20	6	18	18	18
BB80	14	6	17	17	17
LAO033	41	11	31	32	32
4_1362	23	9	38	38	38
BB916	34	14	30	31	31
4_1181	15	8	18	19	19
4_1180	7	2	9	10	10
4_998	14	4	10	11	11
4_1211	9	2	9	9	9
4_1209	14	7	18	18	18
4_1388	6	2	8	9	9
4_949	13	5	12	13	13
BB740	11	4	13	14	14
LAO0160	17	7	7	11	11

nomial time. In an extended $\alpha|\beta|\gamma$ denotation, this is problem $\Sigma^k P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|-$.

Compared to the original LPPA (Leung, Palem, & Pnueli 2001), our main modifications are: use of the Graham list scheduling algorithm (GLSA) adapted to typed task systems and to zero start-start time-lags; new definition of the backward scheduling problem $B(O_i, S_i)$ that does not involve the transitive successors of operation O_i; core LPPA proof adapted to typed task systems and simplified thanks to the properties of monotone interval orders.

Like the original LPPA, our modified algorithm optimally solves a feasibility problem: after scheduling with the core GLSA, one needs to check if the schedule meets the deadlines. By embedding this algorithm in a dichotomy search for the smallest L_{max} such that the scheduling problem with deadlines $d_i + L_{max}$ is feasible, one also solves $\Sigma^k P|intOrder(mono\ \ l_i^j); r_i; p_i = 1|L_{max}$ in polynomial time. This is a significant generalization over the $\Sigma^k P|intOrder; p_i = 1|C_{max}$ problem solved by Jansen (Jansen 1994) in polynomial time.

Our motivation for the study of typed task systems with precedence delays is their use as relaxations of the Resource-Constrained Scheduling Problems (RCPSP) with Unit Execution Time (UET) operations and non-negative start-start time-lags. In this setting, precedence delays are important, yet no previous polynomial-time scheduling algorithms for typed task systems consider them. The facts that interval orders include operations without predecessors and successors, and that the LPPA enforces releases dates and deadlines, are also valuable for these relaxations.

References

Ali, H. H., and El-Rewini, H. 1992. Scheduling Interval Ordered Tasks on Multiprocessor Architecture. In *SAC '92: Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing*, 792–797. New York, NY, USA: ACM.

Brucker, P.; Drexl, A.; Möhring, R.; Neumann, K.; and Pesch, E. 1999. Resource-Constrained Project Scheduling: Notation, Classification, Models and Methods. *European Journal of Operational Research* 112:3–41.

Brucker, P. 2004. *Scheduling Algorithms, 4th edition*. SpringerVerlag.

Chaudhuri, S.; Walker, R. A.; and Mitchell, J. E. 1994. Analyzing and Exploiting the Structure of the Constraints in the ILP Approach to the Scheduling Problem. *IEEE Transactions on VLSI* 2(4).

Dupont de Dinechin, B. 2004. From Machine Scheduling to VLIW Instruction Scheduling. *ST Journal of Research* 1(2). http://www.st.com/stonline/press/magazine/stjournal/vol0102/.

Dupont de Dinechin, B. 2007. Time-Indexed Formulations and a Large Neighborhood Search for the Resource-Constrained Modulo Scheduling Problem. In *3rd Multidisciplinary International Scheduling conference: Theory and Applications (MISTA)*. http://www.cri.ensmp.fr/classement/2007.html.

Garey, M. R., and Johnson, D. S. 1976. Scheduling Tasks with Nonuniform Deadlines on Two Processors. *J. ACM* 23(3):461–467.

Jaffe, J. M. 1980. Bounds on the Scheduling of Typed Task Systems. *SIAM J. Comput.* 9(3):541–551.

Jansen, K. 1994. Analysis of Scheduling Problems with Typed Task Systems. *Discrete Applied Mathematics* 52(3):223–232.

Kolisch, R., and Hartmann, S. 1999. Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis. In J., W., ed., *Handbook on Recent Advances in Project Scheduling*. Kluwer Academic. chapter 7.

Leung, A.; Palem, K. V.; and Pnueli, A. 2001. Scheduling Time-Constrained Instructions on Pipelined Processors. *ACM Trans. Program. Lang. Syst.* 23(1):73–103.

Palem, K. V., and Simons, B. B. 1993. Scheduling Time-Critical Instructions on RISC Machines. *ACM Trans. Program. Lang. Syst.* 15(4):632–658.

Papadimitriou, C. H., and Yannakakis, M. 1979. Scheduling Interval-Ordered Tasks. *SIAM J. Comput.* 8(3):405–409.

Verriet, J. 1996. Scheduling Interval Orders with Release Dates and Deadlines. Technical Report UU-CS-1996-12, Department of Information and Computing Sciences, Utrecht University.

Verriet, J. 1998. The Complexity of Scheduling Typed Task Systems with and without Communication Delays. Technical Report UU-CS-1998-26, Department of Information and Computing Sciences, Utrecht University.