Scheduling Monotone Interval Orders on Typed Task Systems

Benoît Dupont de Dinechin

STMicroelectronics AST Embedded Systems Research Laboratory
Via Cantonale 16E 6928 Manno Switzerland
benoit.dupont-de-dinechin@st.com

Abstract

We present a modification of the Leung-Palem-Pnueli parallel machine scheduling algorithm and prove its optimality for scheduling monotone interval orders with release dates and deadlines on UET typed task systems in polynomial time.

Key words: monotone interval orders, typed task systems, modified deadlines, backward scheduling

Introduction

The Leung-Palem-Pnueli algorithm (LPPA) [5] is a parallel machine scheduling algorithm based on deadline modification and the use of lower modified deadline first priority in a Graham list scheduling algorithm (GLSA). The Leung-Palem-Pnueli algorithm solves the following feasibility problems in polynomial time (see §1.1 for problem denotations):

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

Scheduling problems on *typed task systems* [3] generalize the parallel machine 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 [1]. Jansen [4] gives a polynomial time algorithm for the problem $\Sigma^k P|intOrder; p_i = 1|C_{max}$.

In the present work, we modify the algorithm of Leung, Palem and Pnueli [5] in order to solve $\Sigma^k P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|-$ feasibility problems in polynomial time. Presentation is as follows. In Section 1, we introduce extensions to the $\alpha|\beta|\gamma$ scheduling problem denotation and we discuss the Graham list scheduling algorithm (GLSA) for typed task systems. In Section 2, 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.

1 Deterministic Scheduling Background

1.1 Machine Scheduling Problem Denotation

In parallel machine 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 *dependence* $O_i \prec O_j$ between two operations constrains the schedule with $\sigma_i + p_i \leq \sigma_j$. The *dependence graph* has one arc (O_i, O_j) for each dependence $O_i \prec O_j$.

Machine scheduling problems are denoted by a triplet notation $\alpha|\beta|\gamma$ [1], where α describes the processing environment, β specifies the operation properties and γ defines the optimality criterion. For the deterministic machine scheduling problems, the common values of α, β, γ are:

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:

$prec(l_i^j)$ Dependence delays $l_i^j \geq -p_i$, where $O_i \prec O_j$ implies $\sigma_i + p_i + l_i^j \leq \sigma_j$.
$prec(l_i^j = l)$ All the dependence delays l_i^j have the same value l.
$inTree$ The dependence graph is an in-tree.
$intOrder(mono\ l_i^j)$ The dependence graph weighted by $w(O_i, O_j) \stackrel{\text{def}}{=} p_i + l_i^j$ is a monotone interval order (see below).

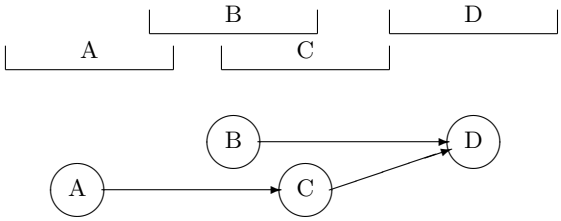

Fig. 1. Set of intervals and the corresponding interval order graph.

As introduced by Papadimitriou & Yannakakis [7], an *interval order* is defined by an incomparability graph that is chordal. An interval order is also the transitive orientation of the complement of an interval graph [7] (see Figure 1). A *monotone interval order* graph [6] is an interval order graph (V, E) with a non-negative weight function w on the arcs such that, given any $(v_i, v_j), (v_i, v_k) \in E : \operatorname{pred} v_j \subseteq \operatorname{pred} v_k \Rightarrow w(v_i, v_j) \leq w(v_i, v_k)$. Here $\operatorname{pred} v_j$ and $\operatorname{pred} v_k$ respectively denote the direct predecessors of v_j and v_k.

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 *dependence-consistent release dates* $\{r'_i\}_{1 \leq i \leq n}$ are recursively defined as $r'_i \stackrel{\text{def}}{=} \max(r_i, \max_{O_j \in \operatorname{pred} O_i}(r'_j + p_j + l^i_j))$. Likewise, the *dependence-consistent deadlines* $\{d'_i\}_{1 \leq i \leq n}$ are recursively defined as $d'_i \stackrel{\text{def}}{=} \min(d_i, \min_{O_j \in \operatorname{succ} O_i}(d'_j - p_j - l^j_i))$.

1.2 Extension of the Graham List Scheduling Algorithm

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 [1]. An operation is available if the current time step is not earlier than its release date and all its direct predecessors have completed their execution early enough to satisfy the entering dependences of this operation.

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 [1] (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 $P|r_i; p_i = 1|L_{max}$), there are no dependences between operations. Therefore, optimal scheduling can

[1] On typed task systems, the operation type must match the processor type.

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 dependences delays $l_i^j = -p_i \Rightarrow \sigma_i \leq \sigma_j$, that is, dependences with zero cycle start-start time lags. Thus we assume the GLSA is extended as follows: in cases of available operations with equal priorities, schedule first the earliest operations in the dependence topological sort order.

2 The Modified Leung-Palem-Pnueli Algorithm

2.1 Algorithm Description

The Leung-Palem-Pnueli algorithm (LPPA) is similar to classic UET scheduling algorithms on parallel processors like Garey & Johnson [2], in that it uses lower modified deadlines first priority in a GLSA. Generally speaking, 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 main feature of the LPPA is its computation of *fixpoint modified deadlines* [2] by solving *backward scheduling problems* denoted $B(O_i, S_i)$. Precisely, the LPPA computes the latest possible schedule date σ'_i of operation O_i in each $B(O_i, S_i)$ and updates its current modified deadline 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 [5].

Our modification of the Leung-Palem-Pnueli algorithm (LPPA) computes the fixpoint modified deadlines $\{d^*_i\}_{1\leq i\leq n}$ by executing the following procedure:

(i) Compute the dependence-consistent release dates $\{r'_i\}_{1\leq i\leq n}$ and initialize the modified deadlines $\{d'_i\}_{1\leq i\leq n}$ as the dependence-consistent deadlines.
(ii) Define an iteration order over the operation set $\{O_i\}_{1\leq i\leq n}$.
(1) Let O_i be the current operation in the iteration.
(2) Compute the optimal backward schedule date σ'_i of O_i in $B(O_i, S_i)$.
(3) Update the modified deadline of O_i as $d'_i \leftarrow \sigma'_i + 1$.
(4) Update the modified deadlines of each $O_k \in \operatorname{pred} O_i$ with $d'_k \leftarrow \min(d'_k, d'_i - 1 - l^i_k)$ ($\operatorname{pred} O_i$ is the set of direct predecessors of O_i).
(5) Go to (1) until a fixpoint of the modified deadlines $\{d'_i\}_{1\leq i\leq n}$ is reached.

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

In our modified algorithm, we also define the *backward scheduling problem* $B(O_i, S_i)$ with $S_i \stackrel{\text{def}}{=} \operatorname{succ} O_i \cup \operatorname{indep} O_i$ ($\operatorname{succ} O_i$ is the set of direct successors of O_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) $\forall O_j \in \{O_i\} \cup S_i : r'_j \leq \sigma'_j < d'_j$

An *optimal backward schedule* for O_i maximizes σ'_i in $B(O_i, S_i)$.

Constraints (a) state that only the dependences 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 the UET typed task systems. Constraints (c) ensure that operations are backward scheduled within the dependence-consistent release dates and the current modified deadlines.

Let $\{r'_j\}_{1 \leq i \leq n}$ be the dependence-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 linear search 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 problem can be solved in polynomial time, optimum backward scheduling would require pseudo-polynomial time.

In order to avoid the pseudo-polynomial time complexity of optimum backward scheduling, the LPPA relies on two dichotomy searches on the feasibility of the constrained backward scheduling problem $\sigma'_i \in [p, q] \wedge B(O_i, S_i)$. For convenience, assume $l_i^j = -\infty$ if $O_i \not\prec O_j$. The feasibility of $\sigma'_i \in [p, q] \wedge B(O_i, S_i)$ is checked by converting the dependences from O_i to each direct successor $O_j \in S_i$ into release dates. This defines a relaxation $\Sigma^k P | \hat{r}_j ; \hat{d}_j ; p_j = 1 | -$ as:

$$\begin{cases} \hat{r}_j \stackrel{\text{def}}{=} \text{ if } j \neq i \text{ then } \max(r'_j, q + 1 + l_i^j) & \text{else } p \\ \hat{d}_j \stackrel{\text{def}}{=} \text{ if } j \neq i \text{ then } d'_j & \text{else } q + 1 \end{cases}$$

This $\Sigma^k P | \hat{r}_j ; \hat{d}_j ; p_j = 1 | -$ relaxation is optimally solved by the GLSA and the earliest $\hat{d}_j$ first priority (Theorem 1). If infeasible, so is the constrained backward scheduling problem $\sigma'_i \in [p, q] \wedge B(O_i, S_i)$.

The first dichotomy search of the LPPA initializes $p = r'_i$ and $q = d'_i - 1$. Then it proceeds to find the latest q such that the above relaxation is feasible over operation set S_i. The second search of the LPPA finds the latest p such that the above relaxation is feasible over operation set $\{O_i\} \cup S_i$. When 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 to find a feasible relaxation, $B(O_i, S_i)$ is assumed infeasible.

Lemma 2 *The optimal backward scheduling procedure applied to $B(O_i, S_i)$ computes the latest σ'_i that satisfies conditions (a), (b), (c).*

PROOF. First assume that the two dichotomy searches are replaced by linear searches. 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 relaxation of $\sigma'_i \in [p, q] \wedge B(O_i, S_i)$. We have $\hat{\sigma}_i = p$ because 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).

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 $p < q$. Relaxation of problem $\sigma'_i \in [p, q] \wedge B(O_i, S_i)$ is feasible while relaxation of problem $\sigma'_i \in [p+1, q] \wedge B(O_i, S_i)$ is infeasible imply there are no scheduling slots available in range $[p+1, q]$, else O_i would use one of them. The relaxation of problem $\sigma'_i \in [p+1, s] \wedge B(O_i, S_i)$ is also infeasible $\forall s \in [p+1, q]$, as it can only differ from the former relaxation on some release dates $\hat{r}_j$ with $\hat{r}_j > p+1$. Therefore, relaxation of problem $\sigma'_i \in [s, s] \wedge B(O_i, S_i)$ is infeasible $\forall s \in [p+1, q]$.

We now prove that dichotomy searches yield the same results as linear searches. The first dichotomy search keeps p constant and moves q. So it yields the same results as linear search if a feasible relaxation of $\sigma'_i \in [p, q] \wedge B(O_i, S_i)$ implies a feasible relaxation of $\sigma'_i \in [p, q-1] \wedge B(O_i, S_i)$. This is the case, since decrementing q may only reduce some of the release dates in the relaxation. The second dichotomy search keeps q constant and moves p. So it yields the same results as linear search if a feasible relaxation of $\sigma'_i \in [p, q] \wedge B(O_i, S_i)$ implies a feasible relaxation of $\sigma'_i \in [p-1, q] \wedge B(O_i, S_i)$. This is the case, since decrementing p only reduces the release date of O_i in the relaxation. □

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

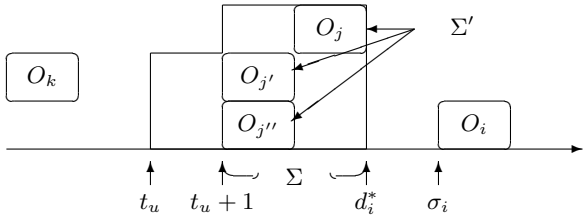

Fig. 2. The core proof of the Leung-Palem-Pnueli algorithm.

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 $B(O_i, S_i)$. 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 [5], 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 2.

Let $r = \tau_i$ be the resource 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 operations within the deadlines. Else, some scheduling slots at t_u are either empty or filled with unsaturated operations. 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 $\Sigma' \stackrel{\text{def}}{=} \{O_j \in \Sigma : r'_j \leq t_u\}$. Subset Σ' only contains dependent operations, as independent operations would have been scheduled at date t_u or earlier by the core GLSA.

Consider problem $P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|-$. In an interval order, given two operations O_i and O_j, either $\operatorname{pred} O_i \subseteq \operatorname{pred} O_j$ or $\operatorname{pred} O_j \subseteq \operatorname{pred} O_i$. This is easily understood by referring to the underlying intervals that define the interval order. Select $O_{j'}$ among $O_j \in \Sigma'$ such that $|\operatorname{pred} O_j|$ 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 $\sigma_k + 1 + l_k^{j'} = \sigma_{j'} > t_u \Rightarrow \sigma_k + 1 > t_u - l_k^{j'}$. By construction, $\operatorname{pred} O_{j'}$ are the direct predecessors of all operations $O_j \in \Sigma'$ and no predecessor of $O_{j'}$ is in Σ'. Thus $O_k \notin \Sigma'$ 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 $S_k \stackrel{\text{def}}{=} \operatorname{succ} O_k \cup \operatorname{indep} O_k$, 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. From Lemma 2, at least one operation $O_j \in \Sigma'$ is scheduled at date t_u or earlier by any stable backward schedule of $B(O_k, S_k)$.

Since $\exists O_j$ scheduled at date t_u or earlier in a stable backward schedule of $B(O_k, S_k)$, from Lemma 2 $\sigma'_k + 1 + l_k^j \leq t_u$ so $d_k^* - 1 + 1 + l_k^j \leq t_u$. By the monotone property, $\operatorname{pred} O_{j'} \subseteq \operatorname{pred} O_j \Rightarrow p_k + l_k^{j'} \leq p_k + l_k^j \Rightarrow l_k^{j'} \leq l_k^j$ 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 algorithm 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 final GLSA. Leung, Palem and Pnueli [5] 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 algorithm has polynomial time complexity.

Summary and Conclusions

We present a modification of the algorithm of Leung, Palem and Pnueli (LPPA) [5] that schedules monotone interval orders with release dates and deadlines on UET typed task systems [3] in polynomial time. Using an extended $\alpha|\beta|\gamma$ denotation, this is scheduling problem $\Sigma^k P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|-$.

Compared to the original LPPA [5], 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 the typed task systems and simplified thanks to the properties of monotone interval orders. Note that including the transitive successors of operation O_i in $B(O_i, S_i)$ can still be useful when using our modified LPPA as an heuristic for scheduling typed task systems. This would not change its optimality in case of monotone interval orders.

Like the original LPPA, our modified algorithm optimally solves a feasibility problem: after scheduling, one needs to check if the schedule meets the deadlines. By embedding our proposed algorithm in a dichotomy search for the smallest L_{max} such that the scheduling problem with deadlines $d_i + L_{max}$ is fea-

sible, one can also solve $\Sigma^k P|intOrder(mono\ l_i^j); r_i; p_i = 1|L_{max}$ in polynomial time. This is a significant generalization over the $\Sigma^k P|intOrder; p_i = 1|C_{max}$ problem solved by Jansen [4] in polynomial time.

In their work, Leung, Palem and Pnueli [5] also describe several techniques that enable to lower the overall complexity of their algorithm. The first is a proof that applying optimum 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 could be applied to typed task systems as well.

References

[1] P. Brucker. Scheduling Algorithms -4th edition. *Springer Verlag*, 2004.

[2] M. R. Garey, David S. Johnson. Scheduling Tasks with Nonuniform Deadlines on Two Processors. *J of the ACM 23(3), pp. 461-467*, 1976.

[3] J. M. Jaffe. Bounds on the Scheduling of Typed Task Systems. *SIAM Journal on Computing, 9, 3*, pages 541–551, 1980.

[4] K. Jansen. Analysis of Scheduling Problems with Typed Task Systems. *Discrete Applied Mathematics 52*, pages 223–232, 1994.

[5] A. Leung, K. V. Palem, A. Pnueli. Scheduling Time-Constrained Instructions on Pipelined Processors. *ACM TOPLAS 23, 1*, pages 73–103, Jan. 2001.

[6] K. V. Palem, B. Simons. Scheduling Time-Critical Instructions on RISC Machines. *ACM Transactions on Programming Languages and Systems (TOPLAS) 15, 4*, pages 632–658, Sept. 1993.

[7] C. Papadimitriou, M. Yannakakis. Scheduling Interval-Ordered Tasks. *SIAM J. of Computing, 8, 3*, pages 405–409, 1979.