Efficient Method for Periodic Task Scheduling with Storage Requirement Minimization

Karine Deschinkel, Sid Touati

Efficient Method for Periodic Task Scheduling with Storage Requirement Minimization

Karine DESCHINKEL, Sid-Ahmed-Ali TOUATI

University of Versailles, France

Abstract. In this paper, we study an efficient approximate integer linear programming formulation of the general problem of one-dimensional periodic task scheduling under storage requirement, irrespective of machine constraints. We have already presented in [8] a theoretical framework that allows an optimal optimization of periodic storage requirement in a periodic schedule. This problem is used to optimise processor register usage in embedded systems. Our storage optimisation problem being NP-complete [7], solving an exact integer linear programming formulation as proposed in [8] is too expensive in practice. In this article, we give an efficient approximate model that allows fast resolution times while providing nearly optimal results. Our solution has been implemented and included inside a compiler for embedded processors.

1 Introduction

This article addresses the problem of storage optimization in cyclic data dependence graphs (DDG), which is for instance applied in the practical problem of periodic register allocation for innermost loops on modern Instruction Level Parallelism (ILP) processors[9]. The massive introduction of ILP processors since the last two decades makes us re-think new ways of optimizing register/storage requirement in assembly codes before starting the instruction scheduling process under resource constraints. In such processors, instructions are executed in parallel thanks to the existence of multiple small computation units (adders, multipliers, load-store units, etc.). The exploitation of this new fine grain parallelism (at the assembly code level) asks to completely revisit the old classical problem of register allocation initially designed for sequential processors. Nowadays, register allocation has not only to minimize the storage requirement, but has also to take care of parallelism and total schedule time. In this research article, we do not assume any resource constraints (except storage requirement); Our aim is to analyze the trade-off between memory (register pressure) and parallelism in a periodic task scheduling problem. Note that this problem is abstract enough to be considered in other scheduling disciplines that worry about conjoint storage and time optimization in repetitive tasks (manufacturing, transport, networking, etc.).

Existing techniques in this field usually apply a periodic instruction scheduling under resource constraints that is sensitive to register/storage requirement. Therefore a great amount of work tries to schedule the instructions of a loop (under resource and time constraints) such that the resulting code does not use more than R values simultaneously alive. Usually they look for a schedule that minimizes the storage requirement under a fixed scheduling period [3–5, 1]. In this paper, we satisfy register/storage

constraints early before instruction scheduling under resource constraints: we directly handle and modify the DDG in order to fix the storage requirement of any further subsequent periodic scheduling pass while taking care of not altering parallelism exploitation if possible. This idea uses the concept of reuse vector used for multi-dimensional scheduling [10, 11].

This article is on continuation on our previous work on register allocation [9]. In that paper, we showed that register allocation implies a loop unrolling. However, the general problem of storage optimisation does not require such loop unrolling. So the current paper is an abstraction of our previous results on register optimisation. Furthermore, it extends it with new heuristics and experimental results.

Our article is organized as follows. Section 2 recalls our task model and notations already presented in [8]. Section 3 recalls the exact problem of optimal periodic scheduling under storage constraints with integer linear programming: our detailed results on the optimal resolution of this problem have been presented in [8]. Since the exact model is not practical (too expensive in terms of resolution time), our current article provides a new look by writing an efficient approximate model in [**?**], that we call *SIRALINA*. Before concluding, Section 5 presents the results of our experimental evaluation of SIRALINA, providing practical evidence of its efficiency.

2 Tasks Model

Our task model is similar to [2]. We consider a set of l generic tasks (instructions inside a program loop) $T_0, \ldots, T_{l-1}$. Each task T_i should be executed n times, where n is the number of loop iterations. n is a an unknown, unbounded, but finite integer. This means that each task T_i has n instances. The k^{th} occurrence of task T_i is noted $T\langle i, k\rangle$, which corresponds to task executed at the k^{th} iteration of the loop, with $0 \leq k < n$.

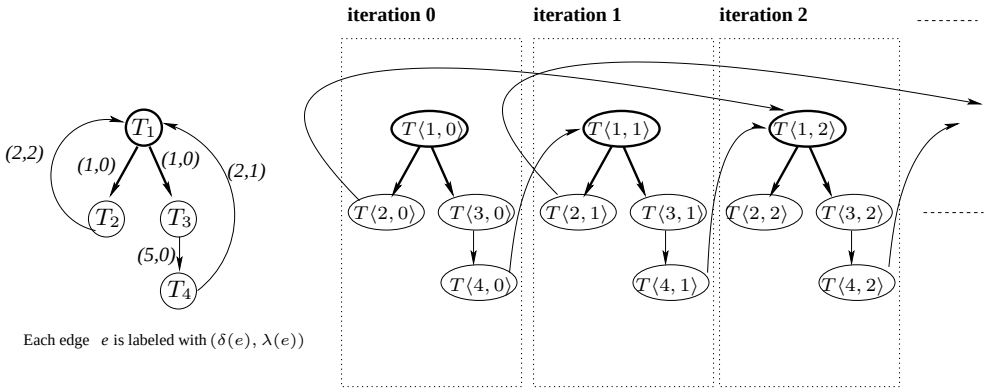
The tasks (instructions) may be executed in parallel. Each task may produce a result that is read/consumed by other tasks. The considered loop contains some data dependences represented with a graph G such that:

- V is the set of the generic tasks of the loop body, $V = \{T_0, \ldots, T_{l-1}\}$.
- E is the set of edges representing precedence constraints (flow dependences or other serialization constraints). Any edge $e = (T_i, T_j) \in E$ has a latency $\delta(e) \in \mathbb{N}$ in terms of processor clock cycles and a distance $\lambda(e) \in \mathbb{N}$ in terms of number of loop iterations. The distance $\lambda(e)$ means that the edge $e = (T_i, T_j)$ is a dependence between the task $T\langle i, k\rangle$ and $T\langle j, k + \lambda(e)\rangle$ for any $k = 0, \ldots, n - 1 - \lambda(e)$.

We make a difference between tasks and precedence constraints depending whether they refer to data to be stored into registers or not

1. V_R is the set of tasks producing data to be stored into registers.
2. E_R is the set of flow dependence edges through registers. An edge $e = (T_i, T_j) \in E_R$ means that the task $T\langle i, k\rangle$ produces a result stored into a register and read/consumed by $T\langle j, k + \lambda(e)\rangle$. The set of consumers (readers) of a generic task T_i is then the set:
$$Cons(T_i) = \{T_j \in V \mid e = (T_i, T_j) \in E_R\}$$

Example 1. Figure 1 is an example of a data dependence graph (DDG) where bold circles represent V_R the set of generic tasks producing data to be stored into registers. Bold edges represent flow dependences (each sink of such edge reads/consumes the data produced by the source and stored in a register). Tasks that are not in bold circles are instructions that do not write into registers (write the data into memory or simply do not produce any data). Non-bold edges are other data or precedence constraints different from flow dependences. Every edge e in the DDG is labeled by the pair $(\delta(e), \lambda(e))$.



(a) Example of a DDG with Generic Taks (b) Loop iterations and instruction/task instances

Fig. 1. Example of Data Dependence Graphs with Recurrent Tasks

In our generic processor model, we assume that the reading and writing from/into registers may be delayed from the starting time of task execution. Let assume $\sigma(T\langle i,k\rangle) \in \mathbb{N}$ as the starting execution time of task $T\langle i,k\rangle$. We thus define two delay functions δ_r and δ_w in which

$$\begin{aligned}
&\delta_w : V_R \rightarrow \mathbb{N} \\
&\quad T_i \mapsto \delta_w(T_i) |\ 0 \leq \delta_w(T_i) \\
&\quad \text{the writing time of data produced by } T\langle i,k\rangle \text{ is } \sigma(T\langle i,k\rangle) + \delta_w(T_i) \\
&\delta_r : V \rightarrow \mathbb{N} \\
&\quad T_i \mapsto \delta_r(T_i) |\ 0 \leq \delta_r(T_i) \\
&\quad \text{the reading time of the data consumed by } T\langle i,k\rangle \text{ is } \sigma(T\langle i,k\rangle) + \delta_r(T_i)
\end{aligned}$$

These two delays functions depend on the target processor and model almost all regular hardware architectures (VLIW, EPIC/IA64 and superscalar processors).

3 Exact Problem Formulation

This section recalls the exact integer linear model for solving the problem of Periodic Scheduling with Storage Minimisation (PSSM). It is built for a fixed desired period

$p \in \mathbb{N}$. Our PSSM exact model uses the linear formulation of the logical implication ($\Longrightarrow$) by introducing binary variables [7]. For more details on this problem, please refer to [8].

3.1 Basic Variables

- a schedule variable $\sigma_i \geq 0$ for each task $T_i \in V$, including σ_{K_i} for each killing node K_i. We assume a finite upper bound L for such schedule variables (L sufficiently large, $L = \sum_{e \in E} \delta(e)$); The schedule variables are integer.
- a binary variables $\theta_{i,j}$ for each $(T_i, T_j) \in V_R^2$. It is set to 1 iff (T_i, T_j) is a reuse edge;
- a reuse distance $\mu_{i,j}$ for all $(T_i, T_j) \in V_R^2$; $\mu_{i,j}$ is an integer variable.

3.2 Linear Constraints

- **Data dependences:** The schedule must at least satisfy the precedence constraints defined by the DDG.

$$\forall e = (T_i, T_j) \in E : \sigma_i - \sigma_j \leq -\delta(e) + p \times \lambda(e)$$

- **Flow dependences:** Each flow dependence $e = (T_i, T_j) \in E_R$ means that the task occurrence $T\langle j, k + \lambda(e)\rangle$ reads the data produced by $T\langle i, k\rangle$ at time $\sigma_j + \delta_r(T_j) + (\lambda(e) + k) \times p$. Then, we should schedule the killing node K_i of the task T_i after all T_i's consumers.

$$\forall T_i \in V_R,\ \forall T_j \in Cons(T_i) | e = (T_i, T_j) \in E_R :\ \sigma_{K_i} \geq \sigma_j + \delta_r(T_j) + p \times \lambda(e)$$

- **Storage dependences**
  There is a storage dependence between K_i and T_j if (T_i, T_j) is a reuse edge:

$$\forall (T_i, T_j) \in V_R^2 : \theta_{i,j} = 1 \Longrightarrow \sigma_{K_i} - \delta_w(T_j) \leq \sigma_j + p \times \mu_{i,j}$$

  This involvement can result in the following inequality :

$$\forall (T_i, T_j) \in V_R^2 : \sigma_j - \sigma_{K_i} + p \times \mu_{i,j} + M_1(1 - \theta_{ij}) \geq -\delta_w(T_j)$$

  where M_1 is an arbitrarily large constant.
  If there is no register reuse between two tasks T_i and T_j, then $\theta_{i,j} = 0$ and the storage dependence distance $\mu_{i,j}$ must be set to 0.

$$\forall (T_i, T_j) \in V_R^2 : \theta_{i,j} = 0 \Longrightarrow \mu_{i,j} = 0$$

  This involvement can result in the following inequality :

$$\forall (T_i, T_j) \in V_R^2 : \mu_{i,j} \leq M_2 \theta_{i,j}$$

  where M_2 is an arbitrarily large constant.

– **Reuse Relations**
The reuse relation must be a bijection from V_R to V_R:
A register can be reused by one task:

$$\forall T_i \in V_R : \sum_{T_j \in V_R} \theta_{i,j} = 1$$

A task can reuse one released register:

$$\forall T_i \in V_R : \sum_{T_j \in V_R} \theta_{j,i} = 1$$

3.3 Objective Function

As proved in [9], the storage requirement is equal to $\sum \mu_{i,j}$. In our periodic scheduling problem, we want to minimize the storage requirement:

$$\text{Minimize} \sum_{(T_i,T_j) \in V_R^2} \mu_{i,j}$$

Using the above integer liner program to solve an NP-problem problem as PSSM is not efficient in practice. We are only able to solve small instances (DDG sizes), in practice arround 12 nodes. For this reason, we propose a more efficient solution by using an approximate model as follows.

4 SIRALINA Approximate Model

4.1 Preliminaries

If edge $e = (T_i, T_j) \in V_R^2$ is not a reuse edge ($(T_i, T_j) \notin E_r$) then $\mu_{i,j} = 0$ else :

$$\forall (T_i, T_j) \in E_r : p \times \mu_{i,j} \geq \sigma_{K_i} - \delta_w(T_j) - \sigma_j$$

Denote $\mu^*_{i,j}$ the optimal reuse distance for $(T_i, T_j) \in E_r$. In this manner :

$$\begin{aligned} Min \sum_{(T_i,T_j) \in V_R^2} \mu_{i,j} &= Min \sum_{(T_i,T_j) \in E_r} \mu_{i,j} \\ &= \sum_{(T_i,T_j) \in E_r} \mu^*_{i,j} \end{aligned}$$

As the reuse relation is a bijection from V_R to V_R we have :

$$\begin{aligned} p \sum_{(T_i,T_j) \in E_r} \mu^*_{i,j} &\geq \sum_{(T_i,T_j) \in E_r} \sigma_{K_i} - \delta_w(T_j) - \sigma_j \\ &\geq \sum_{i \in V_R} \sigma_{K_i} - \sum_{j \in V_R} (\delta_w(T_j) + \sigma_j) \\ &\geq \sum_{i \in V_R} \sigma_{K_i} - \sum_{j \in V_R} \sigma_j - \sum_{j \in V_R} \delta_w(T_j) \end{aligned}$$

We deduce from this inequality that $\sum_{i \in V_R} \sigma_{K_i} - \sum_{j \in V_R} \sigma_j - \sum_{j \in V_R} \delta_w(T_j)$ denoted by SUM is a lower bound for the number of required registers. In this context, it may be useful to find an appropriate scheduling for which this value is minimal. As $\sum_{j \in V_R} \delta_w(T_j)$ is a constant for the problem, we could ignore it in the following optimization problem. We consider the following scheduling problem (P):

$$\begin{cases} \min \sum_{i \in V_R} \sigma_{K_i} - \sum_{j \in V_R} \sigma_j \\ \text{subject to :} \\ \sigma_j - \sigma_i \geq \delta(e) - p \times \lambda(e), \qquad \forall e = (T_i, T_j) \in E \\ \sigma_{K_i} - \sigma_j \geq \delta_r(T_j) + p \times \lambda(e), \forall T_i \in V_R, \forall T_j \in Cons(T_i) | e = (T_i, T_j) \in E_R \end{cases} \tag{1}$$

As the constraints matrix of the integer linear program of System 1 is totally unimodular, *i.e.*, the determinant of each square sub-matrix is equal to 0 or to $\pm$ 1, we can use polynomial algorithms to solve this problem [6]. This would allow us to consider huge DDG. The resolution of problem (P) by a simplex method will provide optimal values σ_i^* for each task $T_i \in V_R$ and the optimal values $\sigma_{K_i}^*$ for each killing node K_i. At this step, we are able to compute the cost $\overline{\mu_{ij}} = \lceil \frac{\sigma_{K_i}^* - \delta_w(T_j) - \sigma_j^*}{p} \rceil$ for each edge $e = (T_i, T_j) \in V_R^2$. Knowing the reuse distance values $\overline{\mu_{ij}}$ if T_j reuses the register freed by T_i, the storage allocation which consists of choosing which task reuses which released register can be modeled as a linear assignment problem. assignment problem.

We consider *the linear assignment problem (A)*:

$$\begin{cases} \min \quad p \quad \sum_{(T_i, T_j) \in V_R^2} \overline{\mu_{i,j}} \theta_{ij} \\ \text{Subject to} \\ \sum_{T_j \in V_R} \theta_{i,j} = 1, \qquad \forall T_i \in V_R \\ \sum_{T_i \in V_R} \theta_{i,j} = 1, \qquad \forall T_j \in V_R \\ \theta_{ij} \in \{0, 1\} \end{cases} \tag{2}$$

where $\overline{\mu_{i,j}}$ is a fixed value for each edge $e = (T_i, T_j) \in V_R^2$.

4.2 Heuristics

We suggest to solve the problem with the following heuristic :

- Solve the problem (P) to deduce the optimal values σ_i^* for each task $T_i \in V_R$ and the optimal values $\sigma_{K_i}^*$ for each killing node K_i,
- Compute the cost $\overline{\mu_{ij}} = \lceil \frac{\sigma_{K_i}^* - \delta_w(T_j) - \sigma_j^*}{p} \rceil$ for each edge $e = (T_i, T_j) \in V_R^2$,
- Solve the linear assignment problem (A) with the Hungarian algorithm which solves assignment problems in polynomial time ($O(n^3)$) to deduce the optimal values $\theta_{i,j}^*$),
- If $\theta_{i,j}^* = 1$ for the edge $e = (T_i, T_j) \in V_R^2$, then (T_i, T_j) is a reuse edge and the reuse distance is equal to $\overline{\mu_{ij}}$.

5 Experiments

We now present the results obtained on several DDG extracted from many well known benchmarks (Spec95, whetstone, livermore, lin-ddot, DSP filters, etc.). The data dependence graphs of all these loops are present in [7]. The small test instances have 2 nodes and 2 edges, the large instances have multiples hundreds of nodes and edges. In order to check the efficiency of our heuristic we compare its results against the optimal ones. We use the ILOG CPLEX 10.2 to solve the integer linear program. The experiment was run on PC under linux, equipped with a Pentium IV 2.13 Ghz processor, and 2 Giaga bytes of memory.

In practice, the optimal method [8] can solve small instances, arround 2 nodes. As far as we know about our problem, there is not simple instances larger than 12 nodes, because our problem is still NP-complete even DDG corresponding to chains or to trees [7]. So, our experiments are decomposed into two parts. A first part uses small DDG instances to investigate the efficiency of SIRALINA compared to the optimal integer linear model. A second part investigates the efficiency of SIRALIA to solve large instances.

5.1 SIRALINA and Optimal Results

Table 1 presents the results of SIRALINA against optimal methods using common benchmarks. This table presents the results for the minimal period of each benchmark. Note that every benchmark has its own minimal period, defined as the critical circuit of the DDG, which is inherent to the data dependences [2]. The first column represents the name of the benchmarks. The second and third column represent the instance size (numbers of DDG nodes and edges). Columns number 4 and 5 give the storage requirement (objective function values) computed by the optimal and SIRALINA methods (some instances could not be solved). The two lasts columns give the resolution times in seconds. As can be seen in this table, SIRALINA is fast and nearly optimal. Sometimes SIRALINA is slightly longer than the optimal method for two reasons: 1) the timer is too precise (milliseconds) and the interactions with operating system disturbs our timing measurements, and 1) SIRALINA performs in two steps while the optimal method performs in one step (resolving a unique integer linear program). In our context, we consider that a time difference which is less than 0.1 seconds is negligible. Another interesting remark is that the processing time of SIRALINA is relatively constant compared to the processing time of the optimal method.

Another improvment of SIRALINA compared to the optimal method is that SIRALINA performs is relatively a constant time irrespective of the considered period p. Figure 2 illustrates some examples, where we can see that the optimal method performs in a high variable processing time in function of the period p while SIRALINA is more stable. Figure 3 shows that SIRALINA is still nearly optimal with various period values. This remark has been checked for all other benchmarks and periods: in almost all benchmarks, SIRALINA computes nearly optimal results for all periods in a satisfactory (fast) processing time.

Benchmark	$\|V\|$	$\|E\|$	S_{opt}	$S_{siralina}$	T_{opt}	$T_{siralina}$
lin-ddot	4	4	7	7	0.007	0.066
liv-loop1	9	11	5	5	0.364	0.067
liv-loop5	5	5	3	3	0.005	0.066
liv-loop23	20	26	10	12	605.548	0.069
spec-dod-loop1	13	15	5	6	198.472	0.067
spec-dod-loop2	10	10	3	3	0.084	0.067
spec-dod-loop3	11	11	3	4	0.257	0.067
spec-dod-loop7	4	4	35	35	0.004	0.066
spec-fp-loop1	5	6	2	2	0.006	0.067
spec-spice-loop1	2	2	3	3	0.004	0.067
spec-spice-loop2	9	10	15	15	2.757	0.067
spec-spice-loop3	4	5	2	2	0.005	0.067
spec-spice-loop4	12	51	8	8	0.088	0.068
spec-spice-loop5	2	2	1	1	0.003	0.067
spec-spice-loop6	6	7	14	14	0.016	0.067
spec-spice-loop7	5	5	40	40	0.005	0.067
spec-spice-loop8	4	4	7	7	0.005	0.067
spec-spice-loop9	11	17	7	7	26.242	0.067
spec-spice-loop10	4	4	2	2	0.005	0.069
spec-tom-loop1	15	18	5	7	604.278	0.068
test-christine	18	17	230	230	600.847	0.068
Elliptic	36	59	NA	10	NA	0.074
whet-cycle4-1	4	4	1	1	0.005	0.066
whet-cycle4-2	4	4	2	2	0.006	0.067
whet-cycle4-4	4	4	4	4	0.01	0.067
whet-cycle4-8	4	4	8	8	0.013	0.069
whet-loop1	16	28	5	6	0.2	0.068
whet-loop2	7	10	5	5	0.006	0.067
whet-loop3	5	16	4	4	0.006	0.067

Table 1. SIRALINA and Optimal Results

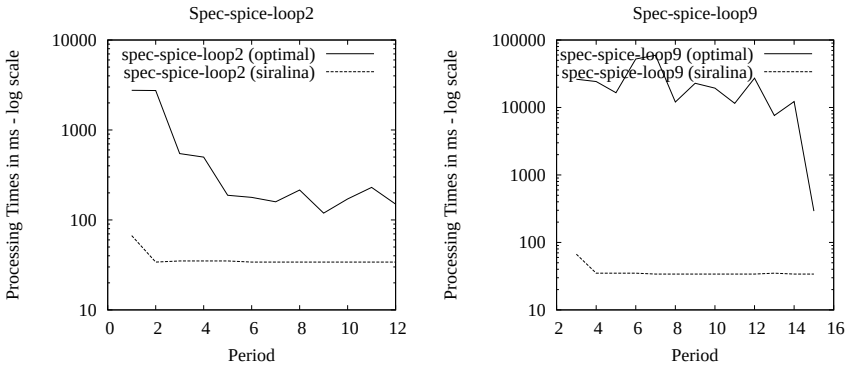


Fig. 2. Processing Times of some Benchmarks vs. Period

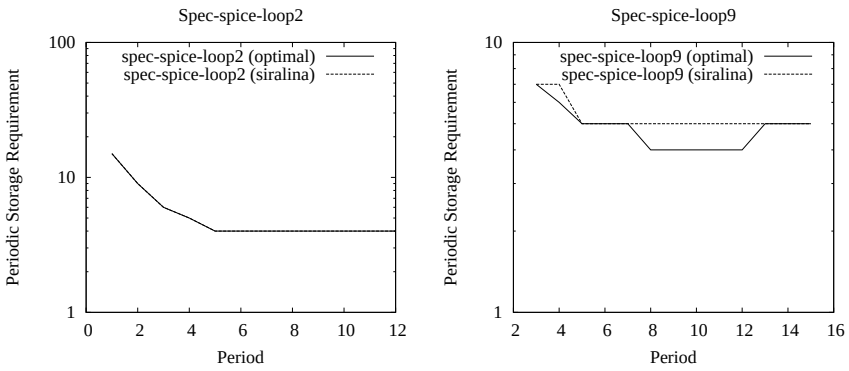


Fig. 3. Storage Requirement of some Benchmarks vs. Period

5.2 SIRALINA with Large Instances

6 Conclusion

This article presents efficient heuristics for the periodic task scheduling problem under storage constraints. Our heuristics are based on the theoretical approach of reuse graphs studied in [8]. Storage allocation is expressed in terms of reuse edges and reuse distances to model the fact that two tasks use the same storage location.

Since computing an optimal periodic storage allocation is intractable in large data dependence graphs (larger than 12 nodes for instance), we have identified one approximate subproblem. We call this simplified problem as SIRALINA. It proceeds in two optimal steps. A first optimal step (totally unimodular constraints matrix) computes scheduling variables. Then a second optimal step solves an assignment problem using the Hungarian method in order to compute the integer μ variables. This second step may alter optimality because μ variables are ceiled.

Our practical experiments on many DDGs show that SIRALINA provides satisfactory solutions with fast resolution times. Consequently, this method is included inside a compiler for embedded systems (in collaboration with STmicroelectronics).

Finally, our future work will loop for an efficient exact models. In some cases, we still require optimal solution because compilation times are less critical. Our exact model published in [8] is not satisfactory enough to be used in practice. A better exact integer formulation is required to be able to solve large instances, even if the problem is NP-complete.

References

1. Benoit Dupont de Dinechin. Parametric Computation of Margins and of Minimum Cumulative Register Lifetime Dates. In David C. Sehr and Utpal Banerjee and David Gelernter and Alexandru Nicolau and David A. Padua, editor, *LCPC*, volume 1239 of *Lecture Notes in Computer Science*, pages 231–245. Springer, 1996.

2. Claire Hanen and Alix Munier. A Study of the Cyclic Scheduling Problem on Parallel Processors. *Discrete Applied Mathematics*, 57(2-3):167–192, 1995.
3. Alexandre E. Eichenberger, Edward S. Davidson, and Santosh G. Abraham. Minimizing Register Requirements of a Modulo Schedule via Optimum Stage Scheduling. *International Journal of Parallel Programming*, 24(2):103–132, April 1996.
4. D. Fimmel and J. Muller. Optimal Software Pipelining Under Resource Constraints. *International Journal of Foundations of Computer Science (IJFCS)*, 12(6):697–718, 2001.
5. Johan Janssen. *Compilers Strategies for Transport Triggered Architectures*. PhD thesis, Delft University, Netherlands, 2001.
6. Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, New York, 1987.
7. Sid-Ahmed-Ali Touati. *Register Pressure in Instruction Level Parallelisme*. PhD thesis, Université de Versailles, France, June 2002. ftp.inria.fr/INRIA/Projects/a3/touati/thesis.
8. Sid-Ahmed-Ali Touati. Periodic Task Scheduling under Storage Constraints. In *Proceedings of the Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA?07)*, August 2007.
9. Sid-Ahmed-Ali Touati and Christine Eisenbeis. Early Periodic Register Allocation on ILP Processors. *Parallel Processing Letters*, 14(2), June 2004. World Scientific.
10. Michelle Mills Strout, Larry Carter, Jeanne Ferrante, and Beth Simon. Schedule-Independent Storage Mapping for Loops. *ACM SIG-PLAN Notices*, 33(11):24–33, November 1998.
11. William Thies, Frederic Vivien, Jeffrey Sheldon, and Saman Amarasinghe. A Unified Framework for Schedule and Storage Optimization. *ACM SIGPLAN Notices*, 36(5):232–242, May 2001.