Worst case analysis of decomposed software pipelining for cyclic unitary RCPSP with precedence delays

Abir Benabid · Claire Hanen

Abstract In this paper, we address a cyclic scheduling problem of finding a periodic schedule with minimal period for the unitary resource constrained cyclic scheduling problem. The main originality is in being able to cope with both precedence delays and complex resource settings which make the problem \(\mathcal{NP}\)-complete, in general.

A guaranteed approach, called Decomposed Software Pipelining, has been proposed by Gasperoni and Schwiegelshohn (Parallel Process. Lett. 4:391–403, 1994), followed by the retiming method by Calland et al. (IEEE Trans. Parallel Distrib. Syst. 9(1):24–35, 1998) to solve the problem assuming parallel identical processors and ordinary precedence constraints. In this paper, an extension of this approach to unitary resource-constrained cyclic scheduling problems with precedence delays, is analyzed and its worst case performance ratio is provided.

Keywords Scheduling · Cyclic scheduling · Software pipelining · Time-lags · Precedence delays · Resource constrained scheduling problem · Approximated algorithm · List algorithm

1 Introduction

Cyclic scheduling problems (Hanen and Munier 1994) have numerous practical applications in production systems (Proth and Xie 1995; Alcaide et al. 2007; Levner et al. 2007) as well as in embedded systems (Dupont de Dinechin et al. 2008; Robert and Vivien 2009; Hanen and Munier 1994) in devices used in mobile telephones, vehicles and consumer electronics. Our research is motivated by the advances in hardware technology, but our results are also valid for mass production systems. In fact, instruction scheduling, also known as software pipelining (Allan et al. 1995), is produced by the compiler in embedded architectures and aims at reducing the operating frequency given real-time processing requirements. The high-quality schedules produced is then a performance critical optimization that has a direct impact on the overall system cost and energy consumption. Most of today’s high performance applications use instruction level parallel processors such as (VLIW) processors (Rau 1993). To benefit from parallelism, software pipelining techniques aims to exploit it across more than one basic block of the code. Since most of the programs are composed of loops with a very large number of iterations (Image processing is a typical example), the parallelism should be extracted from loops. Hence, to achieve efficiency, instruction scheduling can be modeled by a cyclic scheduling problem given by a bivalued (uniform) cyclic precedence graph and resource constraints.

Among the different cyclic scheduling frameworks, modulo (periodic) scheduling (Rau 1994; Hanen and Munier 1994; Dupont de Dinechin et al. 2008) is the most successful in compilers (for example, the LAO compiler of STMicroelectronics) and production systems (Kats and Levner 2003; Alcaide et al. 2007; Levner et al. 2007) since it induces a short description of schedules that optimizes the mem-
ory requirements of the loop code provided by the compiler, as well as the way schedules are executed in production systems. In this approach, each task is repeated periodically every \( \lambda \) time units. The aim is to provide a periodic schedule with minimal period \( \lambda \). This problem is \( \mathcal{NP} \)-hard even if the processing times are unitary, if there are no precedence delays and parallel processors are assumed (Hanen and Munier 1994). If no resource constraints are involved, then the problem can be solved in polynomial time – all references can be found in (Robert and Vivien 2009, pp. 103–128). Polynomial subproblems have been investigated when the precedence graph is acyclic (Hanen and Munier 1994), or when some decisions are made on the ordering of tasks on resources (Alcaide et al. 2007; Levner et al. 2007).

In order to model the software pipelining problem, (Dupont de Dinechin et al. 2008) introduces the RCMSP (Resource Constrained Modulo Scheduling Problem) as an extension of the usual resource constrained scheduling problem RCPSP (Brucker et al. 1999) with precedence delays and provides an integer programming model. This choice is driven by the fact that RCMSP is able to model both resource and precedence constraints induced by the features of VLIW processors. These processors are composed of functional units of different types. An instruction may use several types of functional units at the same time (for example, memory access and floating point unit). We consider the special case, denoted as unitary RCMSP, where the resource demands are unitary. This restriction is motivated by the VLIW structure where usually no more than one functional unit of each type is needed by an instruction. Moreover, VLIW functional units are usually pipelined. This means that they can start a new instruction even though the previous instruction is still in progress. Hence the precedence constraints induced by data dependences between instructions are subject to precedence delays in the RCMSP.

Although periodic scheduling heuristics are commonly used in instruction scheduling of modern VLIW architectures, in the classic modulo scheduling framework (Rau 1994; Huff 1993; Llosa et al. 1996; Lam 1988), the theoretical efficiency of the corresponding schedules is not established. However, the noticeable Decomposed Software Pipelining (DSP) approach introduced in (Gasperoni and Schwiegelshohn 1994; Wang et al. 1994; Calland et al. 1998) has been proved to be among the most efficient in practice (Darte and Huard 2000), and provides the only algorithms with a worst case performance bound. The idea of DSP is to decompose the periodic scheduling problem into two parts: a polynomially solvable graph problem, and a resource constrained non-cyclic scheduling problem. The solutions of the two problems can then be recombined to obtain a feasible periodic schedule. Notice that the performance bound given in (Calland et al. 1998) assumes simple precedence constraints and parallel machines, then it is extended, in (Darte and Huard 2000), to the case of precedence constraints with delays.

The goal of this paper is to explore more deeply the Decomposed Software Pipelining approach and to extend it to deal with unitary RCMSP resource settings and arbitrary non-negative start to start precedence delays. We then provide a worst case performance analysis that generalizes those of (Gasperoni and Schwiegelshohn 1994) and (Calland et al. 1998) which have been developed for parallel processors and simple precedences. Our performance guarantee outperforms the bound given by (Darte and Huard 2000) and extends it to the case of specialized processors. We show that Decomposed Software Pipelining is still a guaranteed algorithm in this context, although the theoretical worst case ratio increases with the number of resource types.

Section 2 presents the mathematical formulation of the cyclic unitary resource constrained modulo scheduling problem RCMSP with precedence delays, illustrated by a loop issued from the ST200 (ST Microelectronics) compiler. In Sect. 3, we introduce the Decomposed Software Pipelining approach and we define a generic algorithm, Extended DSP, to solve our problem. Section 4 presents to the worst case analysis of the extended DSP algorithm. Section 5 concludes the paper.

2 Problem formulation

We now formally define the problem addressed in this paper. We use similar notations as other classical cyclic scheduling papers (Hanen and Munier 1994).

2.1 Resources, tasks, precedences

An instance of a unitary resource-constrained cyclic scheduling problem can be defined by:

1. An architecture model is characterized by \( k \) resource types. The availability of resource type \( r \in \{1, \ldots, k\} \) is denoted by \( m_r \). The resource type \( r \) may be viewed as \( m_r \) parallel processors.

2. A set \( T \) of \( n \) tasks or instructions \( \{T_i\}_{1 \leq i \leq n} \) with integer processing time \( \{p_i\}_{1 \leq i \leq n} \). To each task \( T_i \) we associate a binary vector \( \{b_{ir}\}_{1 \leq r \leq k} \) over the resource types, such that \( T_i \) uses \( b^r_i \) units of type \( r \) resource during its execution. Notice that a task might use several processors but of different types. Each task \( T_i \) must be performed an infinite number of times. We call \( T_i \) at iteration \( q \) and denote by \( (T_i, q) \) the \( q \)th execution of \( T_i \).

3. An inner loop modeled by a cyclic precedence bivalued graph \( G = (T, E) \) where \( E \) is a set of edges defining uniform dependence relations. An edge between two tasks \( (T_i, T_j) \in E \) is characterized by two integers: