Constrained Scheduling of VLSI Algorithms

Emile Aarts[1] [2]
Jan Korst[1]
Jan Karel Lenstra[3] [4]

Abstract

The problem is studied of scheduling VLSI algorithms on signal-processor chips. The objective is to find schedules that minimize in a hierarchical way (1) the makespan and (2) the number of required storage units. The paper concentrates on a formulation of the corresponding constrainèd scheduling problem. In addition, some related complexity issues are addressed, and some preliminary results are discussed obtained with a simulated annealing based approximation algorithm.

Keywords: Combinatorial Optimization, Constrained Scheduling, Simulated Annealing, VLSI Design.

1. Introduction

The complexity of *Very Large Scale Integrated* (VLSI) systems necessitates the use of *Computer Aided Design* (CAD) methods. These methods generally start off with a high level specification of a VLSI algorithm which is transformed into a low level implementation (a chip layout) through a sequence of synthesis steps, where in each successive step more detail is added. This process is often called *silicon compilation*, by analogy with *software compilation*; for an introductory review see the special issue of *Computer* [2].

The various synthesis steps can be viewed as complex decision processes, i.e. the "best implementation" must be chosen among a potentially very large number of alternatives. A number of problems within these decision processes can be mathematically formulated as combinatorial optimization problems. Well-known examples are placement and routing problems [2].

In this paper, we consider the class of combinatorial optimization problems in VLSI design that is related to the problem of finding an efficient and effective mapping of a VLSI algorithm onto a special class of VLSI circuits, i.e. signal processors.

[1] Philips Research Laboratories, P.O. Box 80.000, 5600 JA Eindhoven, the Netherlands

[2] Eindhoven University of Technology, Department of Mathematics and Computer Science, P.O. Box 513, 5600 MB Eindhoven, the Netherlands

[3] Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, the Netherlands

[4] Econometric Institute, Erasmus University, P.O. Box 1738, 3000 DR Rotterdam, the Netherlands

Such a signal processor can be viewed as a general data processing structure containing an *arithmetic section* and a *storage section* [10]. The arithmetic section is composed of a limited number of concurrently operating arithmetic units which carry out elementary operations such as multiplications, additions and subtractions. The storage section accommodates a number of storage units that are used for intermediate storage of the variables used in the VLSI algorithm.

Mapping a VLSI algorithm onto a signal processor poses two minimization objectives:

1. As a result of the typical field of application (audio, video, and telecommunication), data processing takes place at high frequencies. Consequently, the mapping should take as much advantage as possible of the inherent parallelism of the VLSI algorithm, resulting in a minimal execution time.
2. The economic yield of a VLSI circuit strongly depends on the silicon area occupied by it. In practice it turn out that the area of a signal processor is mainly determined by the storage section. Hence the mapping should yield a minimal number of storage units.

The approach pursued in this paper assumes a hierarchy between these two objectives. The execution time is of prime importance: the number of storage units is minimized subject to minimum execution time.

The remainder of this paper focusses on a mathematical formulation of the constrained minimization problem introduced above and a discussion of some related complexity issues (§2). Furthermore, we present some results of a first approach to solve the problem (approximately) by simulated annealing (§3).

2. Problem Formulation

Consider a signal processor consisting of a collection $\mathcal{A} = \{1, 2, \ldots, n\}$ of n types of arithmetic units, each type j with multiplicity n_j, and a VLSI algorithm represented by a *computation scheme* $(\mathcal{O}, t, R, \rightarrow)$, where

- $\mathcal{O}$ is the set of N (indivisible) *operations* in the VLSI algorithm,
- t specifies for each operation $i \in \mathcal{O}$ the execution time $t_i \in \mathbf{Z}^+$,
- R specifies for each operation $i \in \mathcal{O}$ the type of arithmetic unit $R_i \in \mathcal{A}$ necessary to execute operation i, and
- $\rightarrow$ is a precedence relation which induces a partial ordering on $\mathcal{O}$. More specifically, $i \rightarrow j$ implies that i must be completed before j can be started

For a given signal processor, a given computation scheme $(\mathcal{O}, t, R, \rightarrow)$ and a given deadline D, a *feasible schedule* is defined as a set $S = \{S_1, \ldots, S_N\}$ of starting times $S_i \in \mathbf{Z}^+$ for all $i \in \mathcal{O}$ satisfying the following constraints:

(i) the resource constraints, i.e. the maximum number of available arithmetic units of each type j may not be exceeded at any time instance k, i.e.

$$|\{i \in \mathcal{O} \mid (R_i = j) \ \wedge \ (S_i \leq k < S_i + t_i)\}| \leq n_j,$$

(ii) the precedence constraints, i.e. $S_j \geq (S_i + t_i)$ whenever $i \rightarrow j$, for all $i, j \in O$, and

(iii) the deadline D, i.e. $\{S_i + t_i\} \leq D$ for all $i \in O$.

The *makespan* M of a feasible schedule S is defined as the maximum completion time of the operations, i.e. $M = \max_{i \in O}\{S_i + t_i\}$.

The problem imposed by objective 1, can now be formulated as follows:

PRECEDENCE AND RESOURCE CONSTRAINED SCHEDULING
Given a computation scheme $(O, t, R, \rightarrow)$, a deadline D, and bounds n_j on the number of arithmetic units of type j, find a feasible schedule.

With respect to the complexity of PRECEDENCE AND RESOURCE CONSTRAINED SCHEDULING, we can state the following theorem:

Theorem 1
PRECEDENCE AND RESOURCE CONSTRAINED SCHEDULING is NP-hard.

Proof
We consider two special cases, by relaxing firstly the resource constraints, i.e. for $n = 1$ and $n_1 = m$, and secondly the precedence constraints, i.e. for $\rightarrow$ empty. These special cases are called PRECEDENCE CONSTRAINED SCHEDULING and RESOURCE CONSTRAINED SCHEDULING, respectively, and both are known to be NP-complete [3]. □

Clearly, we aim at solving this problem with deadline D equal to M_{min}, the minimum makespan.

At this point we mention that for most instances of the constrained scheduling problem described above, even if $D = M_{min}$, the number of feasible schedules is large. This is intuitively clear from the following arguments. Each feasible schedule with a minimum makespan contains one or more critical paths. Any change in the start times of one of the operations on a critical path leads to an increment of the makespan. For the operations not on a critical path, there usually exist time intervals such that the starting times of the operations may vary within the limits of these intervals without changing the makespan or violating the resource and precedence constraints. This so-called *slack* can be used to minimize the number of storage units, our secondary objective.

Before discussing the problem imposed by both objectives 1 and 2, we consider the storage requirements of a given feasible schedule. To model the storage requirements of a VLSI algorithm, we introduce the concept of *lifetime intervals*. Let $\mathcal{X}$ denote the set of variables used in the algorithm. An operation $i \in O$ uses the values of one or more variables and assigns new values to one or more variables. Here, we distinguish between *single assignment*, i.e. each variable $j \in \mathcal{X}$ receives at most one value during the execution of the algorithm, and *multiple assignment*, i.e. variables may receive more than one value. For a given feasible schedule, a variable i is said to be *alive* between the completion time $S_j + t_j$ of operation j that assigns a (new) value to j and the starting time S_k of operation k that uses this value for the last time. In the case of single assignment, the lifetime of variable i is denoted by the lifetime interval $(S_j + t_j, S_k]$. In the case of multiple assignment, with each variable i a set of lifetime intervals is associated, each interval corresponding to a different value of i. If a variable is alive, then a storage unit is required to store its value. To simplify the control of the signal processor as much as possible, it is required that the subsequent values of one variable are all stored in the same storage unit.

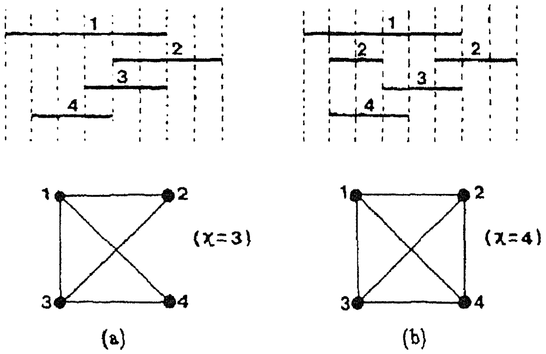

Figure 1: *Lifetime analysis and corresponding interval graph in the case of single assignment (a), and multiple assignment (b).*

Evidently, two variables that are simultaneously alive, cannot be stored in the same storage unit. Hence, the problem of finding a minimal number of storage units for a given schedule S can be formulated as follows:

VARIABLE STORAGE
For a given feasible schedule S, find a mapping $f : \mathcal{X} \to \{1, 2, \ldots, p\}$ such that $f(i) \neq f(j)$ whenever the variables i and j $(i \neq j)$ are simulateously alive, and p, the number of required storage units, is minimal.

To discuss the complexity of VARIABLE STORAGE, we distinguish between single and multiple assignment.

Theorem 2
VARIABLE STORAGE can be solved in polynomial time in the case of single assignment. In the case of multiple assignment, the problem is NP-hard.

Proof
For single assignment, VARIABLE STORAGE can be solved in $O(|\mathcal{X}| \log |\mathcal{X}|)$ by the *left edge algorithm* [5]. In the case of multiple assignment, we consider a special case of VARIABLE STORAGE where for each variable the number of different values is at most two. The problem REGISTER SUFFICIENCY FOR LOOPS,

which is known to be NP-complete [3], reduces to this special case. □

Now, the problem imposed by objective 2 can be formulated as follows:

SCHEDULING TO MINIMIZE VARIABLE STORAGE
Given the set of feasible schedules, specified by a computation scheme $(\mathcal{O}, t, R, \rightarrow)$, a deadline D, and bounds n_j on the number of arithmetic units of type j, find a feasible schedule that minimizes the number of required storage units.

We assume that the set of feasible schedules is *not* listed explicitly but that it can be specified by some compact representation. With respect to the complexity of SCHEDULING TO MINIMIZE VARIABLE STORAGE, we can state the following theorem:

Theorem 3
SCHEDULING TO MINIMIZE VARIABLE STORAGE is NP-hard.

Proof
We consider the special case where $n = 1$ and $n_1 = |\mathcal{O}|$ (no resource constraints) and $\mathcal{X} = \{1, \ldots, m+1\}$. For each variable $i \in \mathcal{X}$ three operations are defined, say i_1, i_2 and i_3, where operation i_1 assigns a value to i and operations i_2 and i_3 use this value (single assignment). Evidently, we have $|\mathcal{O}| = 3(m+1)$. Furthermore, $i_1 \rightarrow i_2$, $i_2 \rightarrow i_3$, $t_{i_1} = t_{i_3} = 0$, and $t_{i_2} = a_i$. By choosing $a_{m+1} = D$, and $\sum_{i=1}^{m} a_i = 2D$, it is straightforward to show that PARTITION, which is known to be NP-complete [3], reduces to this special case of SCHEDULING TO MINIMIZE VARIABLE STORAGE. □

Now, the problem imposed by both objectives 1 and 2 can be formulated as follows:

COMPUTATION SCHEME SCHEDULING
Given a computation scheme $(\mathcal{O}, t, R, \rightarrow)$ and a deadline D, and bounds n_j on the number of arithmetic units of type j, find a feasible schedule that minimizes the number of required storage units.

Theorem 4
COMPUTATION SCHEME SCHEDULING is NP-hard.

Proof
Evidently, COMPUTATION SCHEME SCHEDULING is NP-hard since finding a schedule that is feasible is already NP-hard (Theorem 1) and even if the set of feasible schedules is given the problem remains NP-hard (Theorem 3). □

We end this section with the following remarks. VARIABLE STORAGE reduces to GRAPH COLOURING [3] by constructing the *interval graph* $G = (V, E)$ where $V = \mathcal{X}$ and $\{i, j\} \in E$ if variables i and j are simultaneously alive. The number of required storage units is then given by the *chromatic number* χ of G (see Figure 1). Furthermore, it is easy to show that GRAPH COLOURING reduces to VARIABLE STORAGE with multiple assignment.

3. Solution Approaches

So far the literature does not present algorithms (neither optimization nor approximation algorithms) for solving the combined VLSI scheduling problem described above. However, a number of approaches have been reported to the separate problems. For problems related to PRECEDENCE AND RESOURCE CONSTRAINED SCHEDULING see [4,7,8,10]; for VARIABLE STORAGE see [8,10].

Here we report on a first approach to the combined problem, i.e. COMPUTATION SCHEME SCHEDULING with D equal to the minimal makespan. Our approach is based on a combination of heuristic algorithms, approximating an optimal solution of the problem by following the two step decomposition imposed by the two minimization objectives given in the introduction.

Step 1: A near-optimal feasible schedule is determined for the PRECEDENCE AND RESOURCE CONSTRAINED SCHEDULING problem by using a simple dispatch rule [7], which schedules operations according to the following priority criterion: *Levelling* [1] of the precedence relation yields for each operation the earliest possible starting time (neglecting the constraints induced by the limited number of arithmetic units of each type). These operations are placed in a waiting list, which gives for each instance of time the operations that can be started at that time. Next, the time instances of the waiting list are examined consecutively, starting off at time zero. If the operations in the waiting list at a given time instance do not lead to conflicts, i.e. if the number of available arithmetic units is not exceeded, then the operations are scheduled at the time indicated by the waiting list. Otherwise, a conflict free subset of operations from the waiting list is selected, whilst the operations that are not selected are moved to the next time instance in the waiting list. The rule for selecting a conflict free subset is based on the length of the critical path of the operations in the corresponding precedence graph [4,8,10], such that operations with a long critical path are scheduled first.

Step 2: Given a near-optimal feasible schedule with makespan M, then the COMPUTATION SCHEME SCHEDULING problem is solved approximately by using the principle of *simulated annealing*. This is a randomized version of *neighbourhood search*, applying a stochastic *acceptance rule*, which allows the algorithm to eventually escape from local optima; for an overview see [6]. For this, the *solution space* is given by all feasible schedules with makespan M. The cost of each solution is given by the chromatic number of the interval graph that goes with a given solution, and is approximated by the well-known graph colouring algorithm of Welsh and Powell [9]. A *neighbour* of a given solution is determined by taking a noncritical operation and selecting a new starting time within its slack interval such that the schedule remains feasible.

Preliminary results obtained with our approximation algorithm show that reductions of the number of storage units up to 50% can be obtained (relative to the number of units required by the schedule obtained in step 1). Computation times, however, are large, i.e. up to several hours on a VAX11/750. This is mainly due to the use of the simulated annealing algorithm.

Further research will concentrate on investigating the potentials for speeding up the algorithm, predominantly by using more sophisticated neighbourhood structures. Other research efforts will concentrate on a combined approach in which the makespan and the number of storage units are minimized simultaneously, i.e. instead of following the hierarchical decomposition imposed by objectives 1 and 2, a strategy is pursued that uses a weighted sum of the makespan and the number of storage units as a cost function.

References

[1] Aho, A.V., and J.D. Ullman, *Principles of Compiler Design*, Addison-Wesley, Reading Mass.

[2] *Computer*, 19(1986).

[3] Garey, M.R., and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Fransisco, 1979.

[4] Goossens, G., J. Rabeay, J. Vanhoof, J. Vandewalle and H. De Man, An Efficient Microcode-Compiler for Custom Multiprocessor DSP-Systems, *Proc. Int. Conf. on Computer Aided Design*, Santa Clara, California, 1987, pp. 24-57.

[5] Hashimoto, A., and J. Stevens, Wire Routing by Optimizing Channel Assignment within Large Appertures, *Proc. 8th Design Automation Workshop*, 1971, pp. 155-169.

[6] Laarhoven, P.J.M. van, and E.H.L. Aarts, *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Company, Dordrecht, The Netherlands, 1987.

[7] Lawler, E.L., J.K. Lenstra and A.H.G. Rinnooy Kan, Recent Developments in Deterministic Sequencing and Scheduling: A Survey, in: M.A.H. Demster, J.K. Lenstra and A.H.G. Rinnooy Kan (Eds.), *Deterministic and Stochastic Scheduling*, D. Reidel Publishing Company, Dordrecht, The Netherlands, 1982.

[8] Tseng, C.J., and D.P. Siewiorek, Automated Synthesis of Data Paths in Digital Systems, *IEEE Trans. on Computer-Aided Design* **5**, 379 (1986).

[9] Welsh, D.J.A., and M.B. Powell, An Upper Bound for the Chromatic Number of a Graph and its Application to Timetabling Problems, *The Computer Journal* **10**, 85 (1967).

[10] Zeman, J., and G.S. Moschytz, Systematic Design and Programming of Signal Processors, Using Project Management Techniques, *IEEE Trans. on Acoustics, Speech and Signal Processing* **31**, 1536 (1983).