Optimization problems in system-level synthesis*

Zoltán Ádám Mann

Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Magyar tudósok körútja 2,
H-1117 Budapest, Hungary
zoltan.mann@cs.bme.hu

András Orbán

Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Magyar tudósok körútja 2,
H-1117 Budapest, Hungary
andras.orban@cs.bme.hu

Abstract

System-level synthesis aims at partially automating the design and synthesis process of complex systems that consist of both hardware and software. This involves the usage of formal methods such as graph theory, as well as the formulation of some design steps explicitly as optimization problems.

This paper describes such a graph-theoretic model, and presents three important optimization problems: scheduling, allocation, and partitioning. All of these turn out be $\mathcal{NP}$-hard. However, some important sub-cases can be efficiently solved.

1 Introduction

This paper presents a graph-theoretic model and some related optimization problems in System-Level Synthesis (SLS, [1, 9]). The goal of SLS is to automatically design the optimal hardware and/or software structure from the high-level (yet formal) specification of a system. The optimality criteria may differ according to the particular application; in our model we will use the number of required processing units (PUs) in the case of hardware and the execution time in the case of software as main cost factors.

The central data structure is the elementary operation graph (EOG), which is an attributed data-flow graph. Its nodes represent elementary operations (EOs). An EO might be *e.g.* a simple addition but it might also be a complex function block. Each EO has a given duration $d_i \in \mathbb{N}$. The edges of the EOG represent data flow—and consequently precedences—between the operations.

One important problem is partitioning: deciding which EOs should be realized in hardware and which ones in software, taking into account hardware, software and communication costs. The EOs that should be realized in hardware are then scheduled—*i.e.* their starting times are determined—and allocated in physical PUs. (The goal of scheduling is to enable an efficient allocation in our case. The standard approach in the literature handles scheduling and allocation together [6, 7, 10]. Since both problems are computationally hard in our case, it can be advantageous to separate them.)

The above problems are complicated by the fact that we consider pipeline hardware to achieve maximum throughput. A pipeline system is characterized by two numbers: latency (denoted

*Published in the Proceedings of the 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications, Tokyo (Japan), 2003

by L) is the time needed to process one data item, while restart time (denoted by R) is the period of time before a new data item is introduced into the system. Generally $R \leq L$. Thus, non-pipeline systems can be regarded as a marginal case of pipeline systems, with $R = L$.

2 Scheduling and Allocation

As can be seen below, scheduling and allocation are tightly coupled. This is why we handle them in the same section. First we present the basic definitions; the results can be found in Section 2.2.

2.1 Definitions

Definition 1 *Let* TYPE *denote the set of all possible EO types.* dur : $TYPE \rightarrow \mathbb{N}$ *specifies the duration of EOs of a given type.*

Definition 2 *An* Elementary Operation Graph (EOG) *is a 4-tuple:* $EOG = (G, type, L, R)$, *where* $G = (V, E)$ *is a directed acyclic graph (its nodes are EOs, the edges represent data flow),* $type : V \rightarrow TYPE$ *is a function specifying the types of the EOs,* L *specifies the maximal latency of the system, and* R *is the restart time. The number of EOs is denoted by* n.

Definition 3 *The* duration *(execution time) of an EO is:* $d(EO) = dur(type(EO))$.

Note that L must not be smaller than the sum of the execution times on any execution path from input to output.

The following four axioms [1] provide a possible description of the correct operation of the system:

Axiom 1: EO_j must not start its operation until all of its direct predecessors (*i.e.* all EO_i-s, for which $(EO_i, EO_j) \in E$), have ended their operation;

Axiom 2: The inputs of EO_i must be constant during the total time of its operation ($d(EO_i)$);

Axiom 3: EO_i may change its output during the total time of its operation ($d(EO_i)$);

Axiom 4: The output of EO_i remains constant from the end of its operation until its next invocation.

Definition 4 $asap : V \rightarrow \mathbb{N}$ *and* $alap : V \rightarrow \mathbb{N}$ *denote the ASAP (As Soon As Possible) and ALAP (As Late As Possible) starting times of the EOs.*

As a consequence of Axiom 1, the ASAP and ALAP values satisfy the following equations:

$$asap(EO_i) = \max_{(EO_j, EO_i) \in E} (asap(EO_j) + d(EO_j)) \quad (1)$$

$$alap(EO_i) = \min_{(EO_i, EO_j) \in E} alap(EO_j) - d(EO_i) \quad (2)$$

The ASAP starting time of the inputs of the system is 0. (Note that the system is assumed to work synchronously, and clock cycles are numbered starting with 0.) Similarly, if EO_i is a system output, then $alap(EO_i) = L - d(EO_i)$. Based on this, and on the above equations, the ASAP and ALAP values can be easily calculated. This is done in another phase of the synthesis process, before scheduling.

Definition 5 *The* mobility domain *of an EO is:* $mob(EO) = [asap(EO), alap(EO)] \cap \mathbb{N}$. *The* starting time *of an EO is denoted by* $s(EO)$.

The mobility domain is the set of possible starting times from which scheduling has to choose, *i.e.* $s(EO) \in mob(EO)$.

Definition 6 *A* scheduling σ *assigns to every* EO_i *a starting time* $s_\sigma(EO_i) \in mob(EO_i)$. *The EOG together with the scheduling* σ *is called a* scheduled EOG, *denoted by* EOG_σ.

Definition 7 *A* valid *scheduling is a scheduling that fulfills the above four axioms.*

As we will show in Section 2.2, not every scheduling is valid. Consequently, the starting times of the EOs cannot be chosen arbitrarily in their mobility domains, but the axioms have to be assured explicitly.

Remark 8 *The scheduling defined by the ASAP starting times is valid (per definition, see also equation 1 above). Similarly, the scheduling defined by the ALAP starting times is also valid (see also equation 2 above).*

Definition 9 *Let* Σ *denote the set of all schedulings, and* $\Sigma' \subset \Sigma$ *the set of all valid schedulings.*

The scheduling problem should be defined as an optimization problem over Σ'. But in order to clarify the objective function, we first have to take a look at the allocation problem.

The aim of allocation is to map the EOs to the mimimum number of PUs. Clearly, EOs whose operation does not overlap in time, can be realized in the same PU. This depends on the restart time and the scheduling. More precisely:

Definition 10 *The* busy time interval *of an EO is (in a scheduled EOG):* $busy(EO_i) = [s(EO_i), s(EO_i) + d(EO_i) + \max(\{1\} \cup \{d(EO_j) : (EO_i, EO_j) \in E \text{ and } s(EO_j) = s(EO_i) + d(EO_i)\})]$.

If node i has no successor immediately scheduled after itself, then its busy time interval has length $d(EO_i) + 1$, otherwise $d(EO_i) + d(EO_j)$, where node j is the node with the highest duration scheduled directly after node i. This definition is the consequence of Axioms 2, 3, and 4. (For an explanation, see [1].)

Definition 11 *Two closed intervals* $[x_1, y_1]$ *and* $[x_2, y_2]$ intersect modulo R, *iff* $\exists z_1 \in [x_1, y_1]$ *and* $z_2 \in [x_2, y_2]$, *such that* $z_1 \equiv z_2 \pmod{R}$.

Definition 12 EO_i *and* EO_j *are called* compatible *iff* $type(EO_i) = type(EO_j)$ *and* $busy(EO_i)$ *and* $busy(EO_j)$ *do not intersect modulo* R. *Otherwise they are called* incompatible *(sometimes also called concurrent).*

It can be proven (see [1]) that two EOs can be realized in the same PU iff they are compatible. (Note that if EO_j is started immediately after EO_i has finished, then they are incompatible.)

Based on EOG_σ, we can define a new undirected graph:

Definition 13 *The* concurrency graph *of* EOG_σ *is* $G' = (V', E')$, *where* $V' = V$, *and* $(EO_i, EO_j) \in E'$ *iff* EO_i *and* EO_j *are incompatible in* EOG_σ.

It can be seen easily that finding a realization of EOG_σ using PUs corresponds to a vertex coloring of G'. Thus:

Definition 14 *The* allocation problem *consists of finding a vertex coloring in G' with the minimum number of colors.*

Correspondingly, the objective function of scheduling should be $\chi(G')$. However, our ultimate goal was to use general-purpose optimization heuristics for the scheduling problem. Consequently, we wanted to decouple it from the allocation problem; in addition, we wanted to use an objective function that can be calculated quickly. Therefore, we settled for another objective function, namely the *number of compatible pairs* (that is, the number of edges in the complement of the concurrency graph). We had two reasons for this:

1. Calculating the number of compatible pairs (NCP) is much easier than calculating the number of required PUs;

2. The above two numbers correlate significantly, *i.e.* if the NCP is high, this usually results in a lower number of required PUs.

As we will see in Section 2.2, it is difficult to calculate the number of required PUs. On the other hand, the CONCHECK algorithm [1] can determine the compatibility of two EOs in $\mathcal{O}(1)$ steps, and so the NCP can be calculated in $\mathcal{O}(n^2)$ time.

Now we will try to formally elaborate on the second claim.

Intuitively it seems to be logical that the chromatic number of graphs with many edges is higher than that of graphs with few edges, but this is clearly not always true. There are examples of graphs with many edges and relatively low chromatic number and vice versa. However, the above intuitive claim is true in a statistical sense.

Definition 15 *Let $\mathcal{G}_{n,M}$ denote the set of all graphs with n vertices and M edges. This can be regarded as a probability space, in which every graph has the same probability. $\mathcal{G}_{n,p}$ denotes the set of all graphs with n vertices, provided with the following probability structure: every edge is present with probability p, independently from the others.*

Definition 16 *Let Q be a graph property (that is, a set of graphs). We say that $G \in Q$* almost surely*, iff $\lim_{n\to\infty} Prob(G \in Q \mid G \in \mathcal{G}_{n,p}) = 1$.*

It is known [2], that

$$\chi(G) = \Theta\left(\frac{n}{\log_d n}\right)$$

almost surely, where $d = 1/(1-p)$.

Definition 17 *The graph property Q is said to be* convex*, iff $(G_1 \in Q,\ G_2 \in Q,\ V(G_1) = V(G) = V(G_2),\ E(G_1) \subseteq E(G) \subseteq E(G_2)) \Rightarrow G \in Q$.*

It is also known [3] that if Q is almost sure in the above sense, *i.e.* in $\mathcal{G}_{n,p}$, Q is convex, and $p = p(n)$ is such that $\lim_{n\to\infty} p \cdot \binom{n}{2} = \infty$, and $\lim_{n\to\infty}(1-p)\binom{n}{2} = \infty$, then Q is also almost sure in $\mathcal{G}_{n,M}$, where $M = p \cdot \binom{n}{2}$ (*i.e.* the expected number of edges).

Clearly, the property that $\chi(G)$ equals a given value is convex, so we can write with the appropriate p and M values:

$$\chi(G) = \Theta\left(\frac{n}{\log_d n}\right) = \Theta\left(\frac{n}{\ln n} \ln \frac{1}{1 - \frac{M}{\binom{n}{2}}}\right)$$

almost surely.

It can be seen easily that this function is monotonously increasing in the number of edges. This shows that maximizing the NCP almost surely induces solutions requiring fewer PUs.

Definition 18 *The* scheduling problem *consists of finding a valid scheduling with a maximum number of compatible pairs, given an EOG* $(G, type, L, R)$.

2.2 Results

Proposition 19 *Not every scheduling is valid.*

Proof: Consider the EOG in Figure 1.

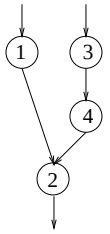


Figure 1: Example EOG

The duration of the EOs is the following: $d(EO_1) = 3$, $d(EO_2) = 1$, $d(EO_3) = 1$, $d(EO_4) = 1$. Let the latency be $L = 4$ (which is actually the lowest possible latency for this system). Consequently, the mobility domains are the following: $mob(EO_1) = [0, 0]$, $mob(EO_2) = [3, 3]$, $mob(EO_3) = [0, 1]$, $mob(EO_4) = [1, 2]$.

However, if both EO_3 and EO_4 were started in cycle 1, this would violate the axioms, since EO_4 needs the output of EO_3. □

Theorem 20 *In the special case when pipeline processing is not allowed ($R = L$), the allocation problem can be solved in polynomial time.*

Proof: Let V_t be the set of EOs of a given type t. If pipeline processing is not allowed, then the subgraph of G' spanned by V_t is an interval graph, and the chromatic number of interval graphs can be found in polynomial time [5]. Clearly, all types can be handled this way, independently of each other. (However, it is not true that G' itself would be an interval graph, but rather a set of interval graphs, between which all edges are present.) □

Theorem 21 *The allocation problem is $\mathcal{NP}$-hard, even if only EOGs with a single type and no edges are considered.*

PROOF: Because of pipeline processing, the class of possible G'-s is not that of interval graphs, but that of circular arc graphs, and the coloring of circular arc graphs is $\mathcal{NP}$-hard. (For a proof, see [4].) It only has to be proven that all circular arc graphs can be constructed as the concurrency graph of an EOG (with one type and no edges).

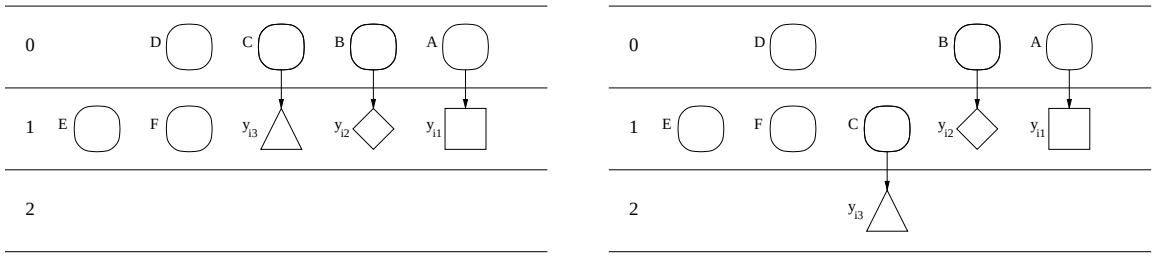
Suppose we have a set of arcs on a circle whose starting and end points have rationale coordinates (measured on the circle from an arbitrary origin). Let us choose a length unit in such a way that the length of all arcs be an integer greater than 1. Now consider an EOG, in which the EOs correspond to the arcs, and the duration of each EO is 1 smaller than the corresponding length; this way the busy time of the EO equals the length of the arc. The starting time of the EO should be the coordinate of the starting point of the arc, and R should be equal to the perimeter of the circle. This way G' will be exactly the corresponding circular arc graph. □

Theorem 22 *The scheduling problem is $\mathcal{NP}$-hard, even if pipeline processing is not allowed.*

PROOF: We show a Karp-reduction of the 3-SAT problem to this problem.

Suppose we have a Boolean satisfiability problem with variables x_l of the form $F = (y_{11} + y_{12} + y_{13})(y_{21} + y_{22} + y_{23}) \dots (y_{t1} + y_{t2} + y_{t3})$ where y_{ij} stands for either some x_l or $\neg x_l$. (If both x_l and $\neg x_l$ occur in the same term, then we can neglect that term, because it has always the value 1.) Now let us construct an EOG from this satisfiability problem. First make two nodes for each variable x_l: one for x_l and one for $\neg x_l$. The mobility range of these variables is the $[1, 2]$ interval. If one of these nodes is scheduled for the first time cycle, this means that the corresponding variable has the value 0, otherwise the value 1. The nodes corresponding to x_l and $\neg x_l$ will have the same type so that they are guaranteed to have different values in an optimal schedule.

Now take one term of the conjunction: $y_{i1}+y_{i2}+y_{i3}$. There are already 3 nodes corresponding to the variables; now we construct 6 more as shown in Figure 2.



(a) 8 compatible pairs

(b) 9 compatible pairs

Figure 2: The EOG belonging to a term of the 3-SAT formula

Here the same symbol means the same type, whereas different symbols mean different types. The mobility range of nodes A, B and C is the $[0, 1]$ interval, for D it is $[0, 0]$ and for E and F $[1, 1]$.

The value of the term should be 1, *i.e.* at least one of the variables y_{i1}, y_{i2}, y_{i3} should have the value 1. If all of them have the value 0 (which is the bad case) then we have the situation

of Figure 2(a) with 8 compatible pairs (concerning the type denoted by circles). If, on the other hand, at least one of the variables has the value 1, then one of the nodes A, B, C may be scheduled in cycle 1, making the NCP 9 (see Figure 2(b)). It can also be seen that the NCP cannot be more than 9.

So the reduction works as follows. First, we create the EOG using the rules just described. Suppose that there are v variables and t terms. Then we ask if the optimal number of compatible pairs is $v + 9t$. More than this is not possible because the number of compatible pairs corresponding to the variables is at most v and the number of compatible pairs corresponding to the terms is at most 9t. If the answer is yes, then the optimal schedule provides the solution of the satisfiability problem. If not, then this means that the satisfiability problem cannot be solved. □

These results show that it is infeasible to strive for a perfect solution. Rather, we have implemented two *heuristic* scheduling methods, which are described in [11]. In the case of allocation, we use a *best-fit* heuristic [8].

3 Partitioning

In this section we consider the partitioning problem, which aims at automatically deciding which operations should be realized in software, and which ones in hardware. Software and hardware costs, as well as communication between software and hardware have to be taken into account.

3.1 Problem definition

An undirected simple graph $G = (V, E)$, $V = \{v_1, \ldots, v_n\}$, $s, h : V \to \mathbb{R}^+$ and $c : E \to \mathbb{R}^+$ are given (G is the EOG of the problem—without directions). $s(v_i)$ (or s_i) and $h(v_i)$ (or h_i) denote the software and hardware cost of node v_i, respectively, while $c(v_i, v_j)$ denotes the communication cost between v_i and v_j if they are in different contexts (HW or SW).

P is called a hardware-software (HW-SW) partition if it is a bipartition of V, $V = V_H \uplus V_S$. The crossing edges are: $E_P = \{(v_i, v_j) : v_i \in V_S, v_j \in V_H \text{ or } v_i \in V_H, v_j \in V_S\}$. The hardware cost of P is: $H_P = \sum_{v_i \in V_H} h_i$; the software cost of P is: $S_P = \sum_{v_i \in V_S} s_i + \sum_{(v_i,v_j) \in E_P} c(v_i, v_j)$ (*i.e.* the software cost of the nodes and the communication cost; since both costs are time-dimensional, it makes sense to add them). The following optimization and decision problems can be defined (G, h, s, c are given in all problems):

Part1: H_0, $S_0 \in \mathbb{R}^+$ are given. Is there a P HW-SW partition so that $H_P \leq H_0$ and $S_P \leq S_0$?

Part2: $H_0 \in \mathbb{R}^+$ is given. Find a P HW-SW partition so that $H_P \leq H_0$ and S_P is minimal.

Part3: $S_0 \in \mathbb{R}^+$ is given. Find a P HW-SW partition so that $S_P \leq S_0$ and H_P is minimal.

3.2 Results

3.2.1 NP-completeness

Theorem 23 PART1 *is $\mathcal{NP}$-complete even if only graphs with no edges are considered.*

PROOF: PART1$\in \mathcal{NP}$, since P is a good proof for that.

To prove the $\mathcal{NP}$-hardness, we reduce the KNAPSACK problem [12] to PART1. Let an instance of the KNAPSACK problem be given. (There are n objects, the weights of the objects are denoted by w_i, the price of the objects by p_i, the weight limit by W and the price limit by K. The task is to decide, whether there is a subset X of objects, so that $\sum_{v_i \in X} w_i \leq W$ and $\sum_{v_i \in X} p_i \geq K$.) We define a graph to that as follows: $V = \{v_1, \ldots, v_n\}$, $E = \{\}$. Let $h_i = p_i$, $s_i = w_i$. (Since E is empty, there is no need to define c.) Introducing $A = \sum_{v_i \in V} p_i$, let $S_0 = W$, $H_0 = A - K$.

Now we solve PART1 with these parameters. We state that it has a solution iff the given KNAPSACK problem has a solution.

Assuming that PART1 has a solution: $V = V_H \uplus V_S$. It means that

$$S_P = \sum_{v_i \in V_S} w_i \leq W \tag{3}$$

and

$$H_P = \sum_{v_i \in V_H} p_i \leq A - K = \sum_{v_i \in V} p_i - K$$

the last one can also be formulated as:

$$K \leq \sum_{v_i \in V} p_i - \sum_{v_i \in V_H} p_i = \sum_{v_i \in V_S} p_i \tag{4}$$

(3) and (4) proves that $X = V_S$ is a solution of the original KNAPSACK problem.

Let now assume that X solves the KNAPSACK problem. Therefore:

$$\sum_{v_i \in X} s_i = \sum_{v_i \in X} w_i \leq W = S_0 \tag{5}$$

and

$$\sum_{v_i \in X} p_i \geq K = A - H_0 = \sum_{v_i \in V} p_i - H_0$$

that is

$$H_0 \geq \sum_{v_i \in V} p_i - \sum_{v_i \in X} p_i = \sum_{v_i \in V \setminus X} p_i = \sum_{v_i \in V \setminus X} h_i \tag{6}$$

(5) and (6) verifies that $V = (V \setminus X) \uplus X$ solves PART1. □

Theorem 24 PART2 *and* PART3 *are* $\mathcal{NP}$*-hard.* □

Although the general partitioning problem seems to be too hard to solve for large inputs, some special cases are easier. If communication is cheap, *i.e.* $c(v_i, v_j) \equiv 0 \quad \forall i, j$, then the partitioning problem reduces according to the proof of Theorem 23 to the well known knapsack problem, for which quasi-polynomial algorithms are known [4]. On the other hand, if the communication is the only significant part, *i.e.* $s_i \equiv 0, h_i \equiv 0 \quad \forall i$, then the trivial optimal solution is to put every node to software. However, if there are some predefined constraints considering the context of some nodes (*i.e.* the nodes in $\emptyset \neq \overline{V_S} \subseteq V$ are prescribed to be in software and the ones in $\overline{V_H} \subseteq V$, to be in hardware, $\overline{V_H} \cap \overline{V_S} = \emptyset$) the problem reduces to find the minimal weighted s-h-cut in a graph, where s and h represent the $\overline{V_S}$ and $\overline{V_H}$ sets, respectively. (If $\overline{V_H} = \emptyset$, then it reduces to find a minimal weighted cut.) This can be solved in polynomial time[8].

3.2.2 ILP solution

The following ILP solution is appropriate for the PART3 problem, but it is straightforward to adopt it to the other versions of the partitioning problem.

$h, s \in \mathbb{R}^n, c \in \mathbb{R}^e$ are the vectors representing each function (n is the number of nodes, e is the number of edges). $E \in \{-1, 0, 1\}^{e \times n}$ is the transposed incidence matrix of G, that is (using the EOG as a directed graph for technical reasons)

$$E[i,j] := \begin{cases} -1 & \text{if the } i\text{th edge starts in node } j \\ 1 & \text{if the } i\text{th edge ends in node } j \\ 0 & \text{if the } i\text{th edge is not connected to node } j \end{cases}$$

Let $x \in \{0,1\}^n$ be a binary vector indicating the partition, that is

$$x[i] := \begin{cases} 1 & \text{if the } i\text{th node is realized in hardware} \\ 0 & \text{if the } i\text{th node is realized in software} \end{cases}$$

It can be seen that $|Ex|$ indicates whether an edge crosses the two contexts or not. So the problem can be formulated as follows:

$$\min hx \tag{7a}$$

$$\begin{aligned} s(\mathbf{1} - x) + c|Ex| &\leq S_0 && (7b) \\ x &\in \{0,1\}^n && (7c) \end{aligned}$$

In Equation (7b) $\mathbf{1}$ means the n-dimensional $(1, \ldots, 1)$ vector. The (7a)-(7c) problem can be transformed to an ILP equivalent by introducing the variables $y \in \mathbb{R}^e$ to eliminate the $|\cdot|$:

$$\min hx \tag{8a}$$

$$\begin{aligned} s(\mathbf{1} - x) + cy &\leq S_0 && (8b) \\ Ex &\leq y && (8c) \\ -Ex &\leq y && (8d) \\ x &\in \{0,1\}^n && (8e) \end{aligned}$$

The last two programs are equivalent. If x solves (7b)-(7c), then $(x, |Ex|)$ solves (8b)-(8e). On the other hand, if (x, y) solves (8b)-(8e), then x will solve (7b)-(7c) too, since $y \geq |Ex|$ and $c \geq 0$.

Solving (8a)-(8e) is still $\mathcal{NP}$-hard, but our empirical results show that with LP-relaxation and branch-and-bound technique it can be solved for up to 300 nodes in acceptable time.

4 Conclusion

In this paper we have presented a graph-theoretic model commonly used in system-level synthesis. We defined the scheduling and allocation problems for pipeline systems, as well as the hardware-software partitioning problem.

All of these problems turned out to be $\mathcal{NP}$-hard in the general case; however, some sub-cases could be identified in which efficient algorithms are known. In particular, the allocation problem can be solved in polynomial time for non-pipeline systems, and the partitioning problem can be solved efficiently for both communication-dominated and processing-dominated systems. Also, the ILP solution for the general partitioning problem could solve large real-world problems.

5 Acknowledgements

This work was supported by Timber Hill LLC and by the PRCH Student Science Foundation. We would also like to thank Gbor Simonyi for pointing us to some useful literature.

References

[1] P. Arató, T. Visegrády, and I. Jankovits. *High-Level Synthesis of Pipelined Datapaths.* John Wiley & Sons, Chichester, United Kingdom, first edition, 2001.

[2] B. Bollobás. The chromatic number of random graphs. *Combinatorica*, 8(1):49–55, 1988.

[3] M. Daws. Probabilistic combinatorics, part III. `http://members.tripod.com/matt_daws/maths/pc.ps`, 2001. Based on the lectures of Dr. Thomason, Cambridge University.

[4] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic and Discrete Methods*, (1):216–227, 1980.

[5] M. Ch. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.

[6] R. L. Graham. Combinatorial scheduling theory. In Lynn Arthur Steen, editor, *Mathematics Today: Twelve Informal Essays*, pages 183–211. Springer, New York, 1978.

[7] R. L. Graham, E. L. Lawler, and J. K. Lenstra. Optimization and approximation in deterministic seqencing and scheduling: a survey. In P. L. Hammer, E. L. Johnson, and B. Korte, editors, *Discrete Optimization II*, pages 287–326. North-Holland, Amsterdam, 1979.

[8] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing, Boston, MA, 1997.

[9] A. A. Jerraya, M. Romdhani, C. Valderrama, Ph. Le Marrec, F. Hessel, G. Marchioro, and J. Daveau. Models and languages for system-level specification and design. In *NATO ASI on System-Level Synthesis, Proceedings*, 1998.

[10] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, editors, *Handbooks in Operations Research and Management Science*, volume 4. Elsevier, Amsterdam, 1993.

[11] Z. Á. Mann and A. Orbán. Integrating formal, soft and diagrammatic approaches in high-level synthesis and hardware-software co-design. In *Proceedings of Informatik 2001*, 2001.

[12] C. H. Papadimitriou. *Computational complexity.* Addison Wesley, 1994.