System-Level Timing-Constrained Scheduling

En-Shou Chang and Daniel D. Gajski

Department of Information and Computer Science
University of California, Irvine, CA 92697-3425

Technical Report #98–15
January 17, 1998

Abstract

HLS scheduling algorithms can not be applied on system-level synthesis due to the following problems:

- The control-step is not available at system-level.
- Mixed concurrent and exclusive execution flows
- Synchronization among objects scheduled
- Execution time of objects scheduled may not be determined until run-time.

In this paper, we present a data-structure to specify the input for system-level scheduling, and a system-level timing-constrained scheduling algorithm. Static scheduling, which has no OS overhead and better system WCET, is used. The algorithm presented can obtain near-optimal solutions within acceptable and predictable CPU time.

1 Introduction

Years ago, when the total number of transistors on a chip hit thousands, to design a system transistor by transistor became a tough job for human. The need to design with abstraction urged the birth of high-level synthesis (HLS)[1, 2], where operations and functional units (FUs) can be used to design a system. Nowaday, the total number of transistors on a chip are hitting millions. Again, to design a system operation by operation becomes painful. Methodologies to design a system at system-level[3, 4, 5], where tasks, processing elements (PEs), and intellectual properties (IPs) can be used, is urgent.

The system synthesis flow[5] consists of a series of well-defined design steps which will eventually map the system specification to the target architecture. These design steps include partitioning, scheduling, and communication refinement. The task of partitioning distributes the behaviors (or processes) that comprise the system specification amongst PEs, ASICs, and IPs. Scheduling determines the order of execution of the behavior to meet certain constraints like response time, throughput, maximum resource available, etc. The task of communication refinement selects the appropriate protocols and resources to implement the abstract communications between the behaviors. The result of the synthesis flow is hand-off to software compilers, HLS synthesis tools, and interface synthesis tools to synthesize the target machine.

Effective scheduling algorithms for HLS are available for years. For almost every HLS scheduling problem, excellent algorithms have been presented to serve. However, most of the algorithms can not be applied on system-level synthesis due to the following problems:

1. Scheduling in HLS is based on control-step, which is no longer available at system-level. The objects need to be schedule at system-level are tasks, whose duration is a real number, no longer an integer.

2. Task graphs for system-level scheduling usually combine concurrent and exclusive execution flows. In HLS, control/data-flow graph (CDFG) has concurrent execution flows only in DFG and exclusive execution flows only in CFG.

3. In addition to sequentiality, task graphs for system-level scheduling also have synchronization between tasks.

4. Execution time of a task may not be determined until the task is executed.

Basically, there are two ways to do system-level scheduling, namely, dynamic scheduling and static scheduling. **Dynamic scheduling**[6] leaves the actual execution sequence of tasks to be determined by operating system (OS) at run-time. The synthesis tools only has to focus on allocation. By using dynamic scheduling, high utilization can be easily obtained whereas the cost of OS overhead, including CPU time and memory, has to be paid. On the other hand, **static scheduling** lets the schedule done by synthesis tools whereas the execution sequence of tasks is determined before run-time. Static scheduling can be favorite under either of the following conditions:

1. **Worst case execution time (WCET) and average execution time of each subtask are similar.** Usually, OS consumes 5% to 40% of system resource. In case the difference between WECT and average execution time is smaller than the OS overhead, static scheduling can have higher performance than dynamic scheduling.

2. **System WCET is more important than utilization.** Worst case execution sequences for both dynamic scheduling and static scheduling are the same. Since dynamic scheduling has to pay for OS overhead, static scheduling can obtain better system WECT.

In this paper, we present a system-level timing-constrained scheduling algorithm. Static scheduling approach is used. The algorithm presented can obtain near-optimal solutions within acceptable and predictable CPU time. We define the time-constrained scheduling problem in Section 2. Our scheduling algorithm and the mathematical theorems which the algorithm is based on are presented in Section 3. Several system design examples are used to illustrate the efficiency and effectiveness of the algorithm in Section 4.

2 Timing-constrained scheduling at system-level

Input for system-level scheduling is an ETG and a timing-constraint. The ETG is a task graph representing sub-behaviors which comprise the system and relations between the sub-behaviors.

Definition 1 An **ETG** is a graph $G = (V, E)$ where

- $V = V_T \cup V_F \cup V_J$;
- $E = E_{SEQ} \cup E_{SYNC}$;
- V_T is a set of **tasks** ;
- V_F is a set of **forks** ;
- V_J is a set of **joints** ;
- E_{SEQ} is a set of **sequentialities**, which are directed arcs;

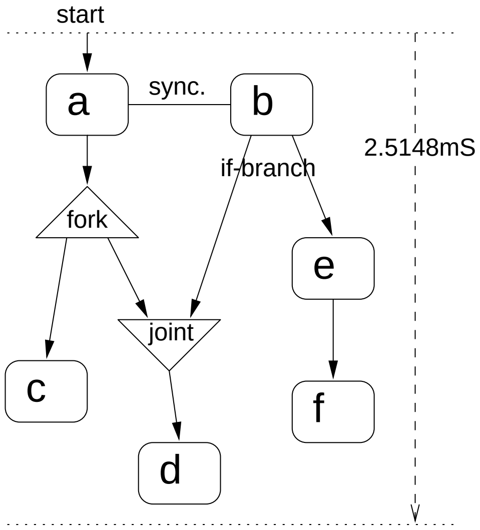

Figure 1: An example of ETG

- E_{SYNC} is a set of **synchronizations**, which are undirected arcs.

Each task t is associated with a PE type p_t and execution time m_t. In case the execution time can only be determined at run-time, WECT can be used as m_t. Only one arc is allowed between any two nodes, either a sequentiality or a synchronization. When multiple sequentialities source a task, only one of the arcs can be true at run-time. When multiple sequentialities source a fork, all of them are true. The sink of a joint can not be true until all sources of the joint become true. Tasks connected by synchronizations have to be executed at the same time.

The nodes connected by synchronizations can be treated as a supernode. Thus, (V, E_{SEQ}) of an acyclic ETG is a partial order.

Definition 2 For a given ETG G, a **schedule** $\sigma : V_T \to \mathcal{R}$ assigns to each tasks t a start-time.

A schedule is called feasible if and only if no sequentiality is violated.

Definition 3 For a given ETG G, **compatible graph** is an undirected graph $C = (V_T, F)$ such that $(t_1, t_2) \in F$ if and only if t_1 and t_2 can always be scheduled on the same PE. t_1 and t_2 are said to be compatible if $(t_1, t_2) \in F$.

Figure 1 shows an example of ETG. In the ETG, task `d` is compatible with both `a` and `e` but not compatible with `c`.

For convenience, we define two general graph functions as following:

$$\Gamma(U) \ni (U, \Gamma(U)) \text{ is a complete undirected graph} \quad (1)$$

$$\Omega(H) = \text{the minimal set of cliques which can cover the graph } H \quad (2)$$

The system cost is determined by the maximum number of PE of each type occupied simultaneously. We define it formally as following:

Definition 4 Given an ETG G, a PE cost function α , and a schedule σ, the **system cost** is defined as

$$\begin{aligned} \mathrm{Cost}(\sigma) = \ & \textstyle\sum_{\forall p \in P} \{\alpha(p) \cdot \max_{\forall x \in \mathcal{R}} o(\Omega(C_p(x)))\} \\ \text{where} \quad & C_p(x) = (A_p(x), \Gamma(A_p(x) \cap F)) \\ & A_p(x) = \{t \mid \sigma(t) \le x < \sigma(t) + m_t, \ \forall t \ni t_p = p\} \\ & C = (V_T, F) \text{ is the compatible graph of } G \\ & P \text{ is the set of PEs used in } G \end{aligned} \quad (3)$$

Now, we can define the system-level timing-constrained scheduling problem as follow:

Definition 5 **System-level Timing-constrained Scheduling:** Given an ETG G, a PE cost function α , and a timing-constraint q, find a feasible schedule σ such that

$$\begin{aligned} & 0 \le \sigma(t) + m_t < q, \ \forall t \in V_t \\ .\text{and}. \quad & \mathrm{Cost}(\sigma) \ \text{ is minimal} \end{aligned} \quad (4)$$

3 The CLMF algorithm

In this section, we first present the mathematical model for our system-level timing-constrained scheduling algorithm in Section 3.1. Then, our scheduling algorithm is shown in Section 3.2.

3.1 Mathematical model

Definition 6 For a given ETG G in which start-time of some tasks is already determined, the corresponding **partial schedule** $\pi(t) = (\eta(t), \lambda(t))$ where

- both η and λ are schedules ;
- $\eta(t) = \lambda(t) =$ start-time of t , if t is scheduled ;
- $\eta(t)$ is earliest feasible start-time and $\lambda(t)$ is latest feasible start-time, otherwise.

When an ETG G in which no task has been scheduled yet is given, η is the ASAP schedule for G and λ is the ALAP schedule.

Definition 7 For a given partial schedule π, the corresponding **distribution** $\delta : P \times \mathcal{R} \to \mathcal{R}$ such that

$$\delta(p, x) = \sum_{\forall y \in \Omega(C_p(x))} \max_{t \in y} \frac{m_t}{\lambda(t) - \eta(t) + m_t} \quad (5)$$

$$\text{where} \quad C_p(x) = (A_p(x), \Gamma(A_p(x) \cap F) \quad (6)$$

$$A_p(x) = \{t \mid \eta(t) \le x < \lambda(t) + m_t, \forall t \ni t_p = p\} \quad (7)$$

Under the condition that no hint is provided, we assume the probability of scheduling a task t to any point between $\eta(t)$ and $\lambda(t)$ is equal. Thus the distribution function $\delta(p, x)$ is the expected resource requirement at time x. As a consequence, the **expected system cost** is

$$\mathrm{Ex\_Cost}(\pi) = \sum_{\forall p \in P} \{\alpha(p) \cdot \max_{\forall x \in \mathcal{R}} \delta(p, x)\} \quad (8)$$

Definition 8 For a given distribution δ, **distribution boundary** is a sorted list $B = \{x_1, x_2, \cdots, x_n\}$ such that

$$\eta(t), \eta(t) + m_t, \lambda(t), \lambda(t) + m_t \in B,\ \forall t \in V_T$$
$$\text{.and.} \qquad i \leq j \Leftrightarrow x_i \leq x_j \tag{9}$$

Theorem 1 Given a distribution δ and its distribution boundary B,

$$\forall b_i, b_{i+1} \in B,\ b_i \leq x < b_{i+1}, b_i \leq y < b_{i+1}$$
$$\Rightarrow \qquad \delta(p, x) = \delta(p, y) \tag{10}$$

proof : Given $t \in V_T$, $\ b_i \leq x < b_{i+1}, b_i \leq y < b_{i+1}$

$$\Rightarrow\ x, y < \eta(t)$$
$$\text{or } \eta(t) \leq x < \lambda(t) + m_t, \eta(t) \leq y < \lambda(t) + m_t$$
$$\text{or } \lambda(t) + m_t \leq x, y$$
$$\Rightarrow\ A_p(x) = A_p(y) \text{ where } A_p \text{ is defined as in Eq.(7)}$$
$$\Rightarrow\ \delta(p, x) = \delta(p, y)$$

Theorem 1 directly implies the following useful theorem:

Theorem 2 *The expected system cost* Ex_Cost *can be found over distribution boundary.*

Given a partial schedule π, let $\pi_{x,y} = (\eta_{x,y}, \lambda_{x,y})$ denote one of π's consequent partial schedule in which task x is then scheduled to start-time y. We have

$$\eta_{x,y}(t) = \begin{cases} \eta(t) + (y - \eta(x)) & \text{if } t \text{ is a successor of } x \\ \eta(t) & \text{otherwise} \end{cases} \tag{11}$$

$$\lambda_{x,y}(t) = \begin{cases} \lambda(t) - (\lambda(x) - y) & \text{if } t \text{ is a predecessor of } x \\ \lambda(t) & \text{otherwise} \end{cases} \tag{12}$$

Theorem 3 *For a given partial schedule π and a given un-scheduled task x,* $\max_{\eta(x) \leq z \leq \lambda(x)} \text{Ex\_Cost}(\pi_{x,z})$ *can be found over $\hat{B}$* , where

$$B \text{ is the distribution boundary}$$
$$d = \lambda(x) - \eta(x)$$
$$E_x = \{\eta(t) \mid t \text{ is a successor of } x\} \tag{13}$$
$$L_x = \{\lambda(t) \mid t \text{ is a predecessor. of } x\} \tag{14}$$
$$\dot{B}_y = \{b | b \in B, y \leq b \leq y + d\},\ \forall y \in E_x \tag{15}$$
$$\ddot{B}_y = \{b | b \in B, y - d \leq b \leq y\},\ \forall y \in L_x \tag{16}$$
$$\dot{B} = \{\eta(x) + b - y \mid \forall b \in \dot{B}_y, \forall y \in E_x\} \tag{17}$$
$$\ddot{B} = \{\lambda(x) - (y - b) \mid \forall b \in \ddot{B}_y, \forall y \in L_x\} \tag{18}$$
$$\hat{B} \text{ is the sorted list of } \dot{B} \cup \ddot{B} \tag{19}$$

proof:
Let $f(z) = \delta(p, q) \mid_{\pi_{x,z}}$
where δ is defined in Eq.(5)
Considering $\delta(p, q)$ where $q \leq z$
If task t is not a predecessor of task x

$$\frac{m_t}{\lambda(t)|_{\pi_{x,z}} - \eta(t)|_{\pi_{x,z}} + m_t} \tag{20}$$
$$= \frac{m_t}{\lambda(t) - \eta(t) + m_t} \tag{21}$$

All the terms in Eq.(21) are independent from z thus Eq.(21) is a constant.
If task t is a predecessor of task x

$$\frac{m_t}{\lambda(t)|_{\pi_{x,z}} - \eta(t)|_{\pi_{x,z}} + m_t} \tag{22}$$
$$= \frac{m_t}{\lambda(t) - (\lambda(x) - z) - \eta(t) + m_t} \tag{23}$$
$$= \frac{m_t}{c + z} \tag{24}$$

where

$$c = \lambda(t) - \eta(t) + m_t - \lambda(x) \tag{25}$$

is independent from z
When z is increasing, Eq.(24) is decreasing
On the other hand,
all the terms inside $\sum$ in Eq.(5) are not changed, since A_p defined in Eq.(7) is identical

Algorithm CLMF

begin
  Compute Compatible Graph of G
  $\pi = (\ \mathrm{ASAP}(G)\ ,\ \mathrm{ALAP}(G)\ )$
  while $\exists t \in V_T$, t is not scheduled **do**
    /* choose the best task */
    Find $t \in V_T$ such that $\frac{\lambda(t)-\eta(t)}{\alpha(t_p)}$ is minimum
    /* compute the best start-time */
    Compute distribution boundary B
    Compute $\hat{B}$ as defined in Eq.(19)
    find $z \in \hat{B}$ such that Ex_Cost($\pi_{t,z}$) is minimum
    $\pi := \pi_{t,z}$
  endwhile
end

Figure 2: Costly-Least-Mobility-First Scheduling Algorithm (CLMF)

for all $z \ni c_i \leq z \leq c_{i+1}$ where $c_i, c_{i+1} \in \hat{B}$

Thus,

$$f(z) = \delta(p,q)\ |_{\pi_{x,z},\ q \leq z} \tag{26}$$

is a decreasing function within domain $c_i \leq z \leq c_{i+1}$

Similarly,

$$f(z) = \delta(p,q)\ |_{\pi_{x,z},\ q \geq z} \tag{27}$$

is an increasing function within domain $c_i \leq z \leq c_{i+1}$

Therefore, $f(z)$ is monotonic between c_i and c_{i+1}

And, all the local max. and min. are located on $\hat{B}$

3.2 The algorithm

Since the scheduling problem is NP-hard, approximation algorithms which can find near-optimal solutions within predictable and acceptable time is favorite. We develop an algorithm that finds near-optimal solutions by constructing a sequence of partial schedule, where in each iteration an un-scheduled task is assigned to a start-time. The algorithm is shown in Figure 2.

The CLMF algorithm select the most critical task to be scheduled first by task mobility and PE cost. Since the selection of the start-time of the task with less mobility is more critical to the system cost, the task should be considered earlier. On the other hand, the cost of the PE which executes the operation can also play an important roll in choosing the candidate. Obviously costly PEs are more critical to the system cost than cheaper PEs. As a result, we use a mixed cost-mobility measure to decide which task is scheduled first.

Once the candidate which has highest priority is determined, the CLMF algorithm assigns the task to a start-time where the expected system cost Ex_Cost can be minimized. The statistical expected system cost of a partial schedule can be computed by Eq.(3). According to Theorem 2, the CLMF algorithm can find the Ex_Cost by computing Eq.(5) over the distribution boundary B defined in Definition 8. Whereas, with the help of Theorem 3, the CLMF algorithm can find the best start-time for the selected task t by evaluating all the Ex_Cost of scheduling t to each element in $\hat{B}$. As soon as the best start-time for the task t is found, the CLMF algorithm schedule t to the start-time, then the CLMF continues next iteration. Finally, all the tasks are scheduled one by one until the schedule is completed.

4 Experimental results

We use several system design examples to illustrate the efficiency and effectiveness of the CLMF algorithm. The results of CLMF algorithm are compared with human designs.

Figure 3 shows the ETG derived from an ATM encoder specification. The results obtained by both the CLMF algorithm and human designer are shown in Figure 4. We can see the CLMF algorithm obtains results as good as human designer in most cases on this small example.

Table 1 shows results of examples with different sizes. The cost ratio is the ratio of the cost of the result obtained by the CLMF algorithm against that of human

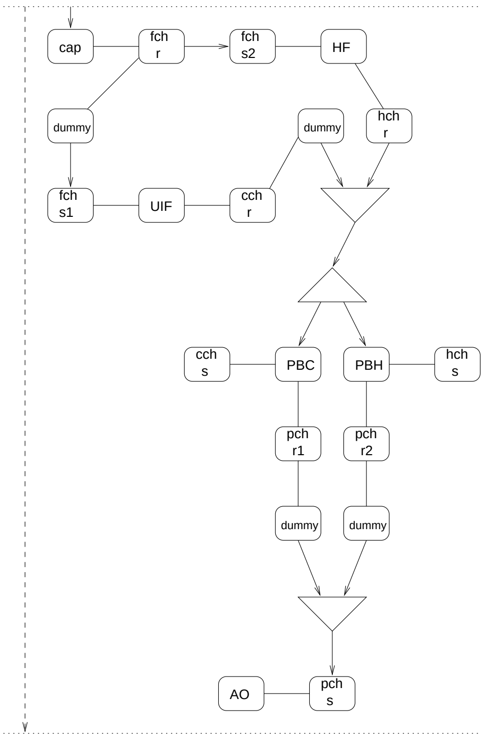

Figure 3: ETG of an ATM encoder specification

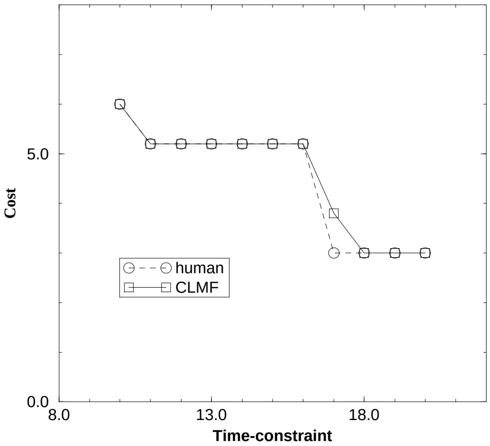

Figure 4: Results of human designer and the CLMF algorithm

	# node	# arc	CPU time	A/H cost ratio
atm2	54	93	0.05s	1.01
s1	138	313	0.67s	1.00
s2	287	704	3.54s	0.98
s3	583	1372	18.12s	n.a

Table 1: Results of four larger examples

designer. The human designer performs worse and becomes more erroneous when dealing with larger system specification. The ratio of the largest example is not available, since the example is too big to be comprehended by human designers.

5 Concluding remarks

We define the system-level timing-constrained scheduling problem in this paper. HLS scheduling algorithms can not be applied on system-level synthesis due to the following problems:

- The control-step is not available at system-level.
- Mixed concurrent and exclusive execution flows
- Synchronization among objects scheduled
- Execution time of objects scheduled may not be determined until run-time.

We present a data-structure to specify the input for system-level scheduling, and an algorithm which can perform system-level timing-constrained scheduling effectively and efficiently.

Static scheduling approach, which determines the execution sequence of tasks when the system is synthesized, is used in this paper. Static scheduling has two advantages:

- No OS overhead
- Better system WCET

The algorithm presented can obtain near-optimal solutions within acceptable and predictable CPU time.

The system-level scheduling methodology presented in this paper targeting on obtaining scheduling which can meet a given system response time requirement. Methodologies which consider features like system with multiple local timing-constraints, architecture which consists pipelines, and scheduling which can meet a given throughput requirement are still needed.

References

[1] D. Gajski (Editor), *Silicon Compilation.* Addison-Wesley, 1988.

[2] D. Gajski, N. Dutt, C. Wu, and Y. Lin, *High-Level Synthesis: Introduction to Chip and System Design.* Boston, Massachusetts: Kluwer Academic Publishers, 1991.

[3] J. Zhu, R. Dömer, and D. Gajski, "Essential issues in codesign." Chapter 1 in J. Staunstrup and W. Wolf, Editors, *Hardware/Software Co-Design: Principles and Practice*, Kluwer Academic Publishers, October 1997 (ISBN 0-7923-8013-4).

[4] J. Zhu, R. Dömer, and D. Gajski, "Syntax and sematics of the SpecC language," in *Proceedings of the Synthesis and System Integration of Mixed Technologies*, December 1997.

[5] D. Gajski, G. Aggarwal, E.-S. Chang, R. Dömer, T. Ishii, J. Kleinsmith, and J. Zhu, "Methodology for design of embedded systems." UC Irvine, Dept. of ICS, Technical Report 98-07,March 1998.

[6] T.-Y. Yen and W. Wolf, "Sensitivity-driven co-synthesis of distributed embedded systems," in *Proceedings of the International Symposium on System Synthesis*, 1995.