ICS

TECHNICAL REPORT

A Mathematical-Statistics-Based System-Level Time-Constraint Scheduling Algorithm

En-Shou Chang and Daniel D. Gajski

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

Technical Report 99-36

August 8, 1999

Abstract

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

- The clock is no longer able to align the scheduled objects at system-level
- Combined concurrent and exclusive execution flows
- Synchronization precedences among the scheduled objects
- Execution time of the scheduled objects may not be determined until run-time.

In this paper, we present an extended task graph to capture the system-level scheduling problems. We define the system-level time-constraint scheduling problems based on the extended task graph, and a scheduling algorithm is presented. Static scheduling, which can have less OS overhead and better system WCET, is used. The algorithm can obtain good solutions within acceptable and predictable CPU time.

Information and Computer Science

University of California, Irvine

1 Introduction

Scheduling has been a primary research interest for decades, since it plays a major roll of balancing cost and performance. Usually, allocating more resources for a design can obtain higher performance; whereas allowing lower performance for the design can reduce resource requirement. The task of scheduling trades off resources and performance and delivers the best schedule. Basically, there are two categories of scheduling: (a) **time-constraint scheduling**, the given time-constraint can be met with less resources needed; (b) **resource-constraint scheduling**, the given resource-constraint can be met with higher performance.

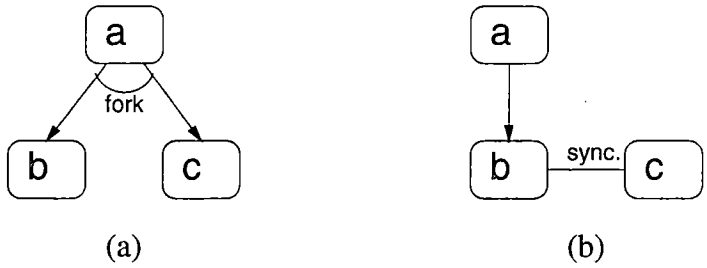
Time-constraint scheduling algorithms proposed by Paulin and Knight[12], Lee et.al.[2], Devadas and Newton[13] are widely used in High-Level Synthesis (HLS)[9]. However, these algorithms require the operations to be aligned by clock, where the clock is not available in system-level synthesis or too small to be applied to system-level synthesis. For example, a system may have processing elements (PEs) running on different clock rates, thus we can not have *a* clock available for the above algorithms. In addition, the objects scheduled at system-level are tasks, which can take hundreds or millions of clocks to execute. As a consequence, the above algorithms will take unbearable time to compute a schedule.

In this paper, we propose a novel time-constraint scheduling algorithm which works independently from clock. In addition, the algorithm also takes several important characteristics of system-level scheduling problems into account. These characteristics are list as following.

1. The task graph for system-level scheduling combines concurrent and exclusive execution flows. Contrary to control/data-flow graph (CDFG) used in HLS, which has concurrent execution flows only in DFG and exclusive execution flows only in CFG.

2. In addition to **sequentialities**, precedences in task graphs for system-level scheduling also include **synchronizations**. Figure 1 illustrates the idea of sequentiality and synchronization. Two tasks are said to be **concurrent** if they can be executed independently. Concurrent task are not required to be executed at the same time. On the other hand, two tasks may be required to be executed at exactly the same time, for example, two tasks which communicate by way of hand-shaking. We define the synchronization as a new type of execution precedence between two tasks, in which the tasks connected by the synchronization have to be executed at the same time.

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



(a) (b)

Figure 1: (a) Sequentiality and concurrent successors: task b and c *can* be executed concurrently once task a is complete, but b and c are not required to start at the same time. (b) Synchronization: task b and c can be executed after a is completed, and b and c have to start together exactly the same time. b and c are not required to start immediately after a is finished.

to cache effect and data-dependent execution, the execution time of a task may vary, though average or worst case execution time can be calculated before execution.

Basically, there are two approaches to do system-level scheduling, namely, dynamic scheduling and static scheduling. **Dynamic scheduling**[8, 10, 3, 4, 11] leaves the actual execution sequence of the tasks to be determined by the operating system (OS) at run-time, whereas the OS overheads, including the computation time consumed by the OS and the hardware cost of the OS, are paid to obtain higher utilization of the resources of the system. On the other hand, **static scheduling**[1, 5, 6] determines the execution sequence of tasks before run-time. The schedule is computed by design tools in advance, whereas no extra OS overheads need to be paid during run-time.

In addition to saving the OS hardware cost, static scheduling is also more favorite when any of the following conditions are present.

1. *Worst Case Execution Time (WCET) and average execution time of each task are closed.* The OS can impose 5% to 40% of additional execution time. In case the difference between the WCET and the average execution time is smaller than the OS overhead, static scheduling can obtain higher performance than dynamic scheduling in both worse case and average case.

2. *System WCET is more important than utilization.* The execution sequences of the tasks in worst case for both dynamic scheduling and static scheduling are the same. Since dynamic scheduling has to pay the OS overhead, static scheduling can obtain a shorter system WCET.

In this paper, we present a system-level scheduling model and a time-constraint scheduling algorithm. Static scheduling approach is used. The algorithm presented can obtain good schedules within acceptable and predictable CPU time. Comparison of our work with previous works on system-level static scheduling are made in Section 2. We define the scheduling model and the time-constraint scheduling problem in Section 3. Several mathematical statistic properties based on the scheduling model are developed in Section 4. Our scheduling algorithm, which make good use of the mathematical statistic properties, are presented in Section 5. Several system design examples are used to illustrate effectiveness of the algorithm in Section 6. Finally, a conclusion is drawn in Section 7.

2 Previous Works

Ha and Lee[5] propose a method to compute quasi-static scheduling of DFGs at compile-time to reduce the OS overhead. Two types of DFG nodes, namely, select and switch, are used to capture exclusive execution flow. These nodes are specially treated while computing scheduling. Similar to our work, data-dependent executions are profiled for making scheduling decisions. In their work, the number of PEs is given and a quasi-static scheduling with shorter DFG completion time is produced; whereas our work inputs a time-constraint and produces a relative static scheduling with less PEs required.

Similar to our work, Eles et.al.[6] define an extended task graph to accommodate required system-level scheduling information. Combined concurrent and exclusive execution flows are taken into account in their work, but synchronization precedences are not considered. All possible execution sequences of tasks are computed while the target system is synthesized. Contrary to our work in which the time-constraint is given, they assume the number of PEs (resource-constraint) is given.

Ziegenbein et.al.[1] attach additional information, which is called *process mode*, to each task. The process mode indicates the exclusive existence between the task and other tasks, as well as how the life-time of the task is related to process modes of other tasks. Compared to their approach, similar characteristics are explicitly expressed in the task graph we defined. Their work assume the amount of resources is given and try to compute a less exaggerated WCET while the target system is synthesized. Contrary to their work, we assume the time-constraint is given and try to compute a schedule satisfied the time-constraint and using less resources.

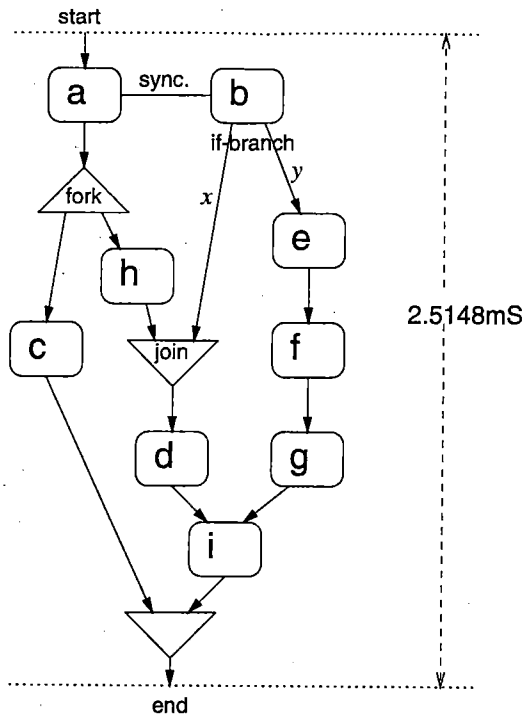


Figure 2: An example of ETG with a time-constraint

3 System-Level Time-Constraint Scheduling Problem

The input for system-level time-constraint scheduling is an acyclic **Extended Task Graph (ETG)** and a time-constraint. The ETG is a task graph representing precedences of the sub-behaviors which comprise the system. We call each sub-behavior a **task**. The ETG is defined as following.

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

- The set of nodes(vertexes) $V = V_T \cup V_F \cup V_J \cup \{ v_{start} , v_{end} \}$;
- V_T is a set of **tasks** ;
- V_F is a set of **forks** ;
- V_J is a set of **joins** ;
- v_{start} is the start node;

- v_{end} is the end node;
- The set of arcs $E = (E_{SEQ} \cup E_{SYNC}) \subset \{ (e_i, e_j) \mid e_i, e_j \in V \}$;
- $(e_i, e_j) \in V \Rightarrow \text{not}((e_j, e_i) \in V)$
- $E_{SEQ} \cap E_{SYNC} = \phi$
- E_{SEQ} is a set of **sequentialities**, which are directed arcs;
- 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, estimated WCET or average execution time can be used as m_t. Each arc represents a **precedence** between two nodes and the precedence can be either a sequentiality or a synchronization. At most one precedence is allowed between any two nodes. When multiple **sequentialities** source a **task**, only one of them can be true at run-time. When multiple sequentialities source a **fork**, all of them are true. The sink of a **join** can not be true until all sources of the join become true. Tasks connected by **synchronizations** have to start at the same time.

Definition 2 An ETG G is said to be **acyclic** if there is no directed loop in G. Undirected arcs can be traveled either way. A loop is not called **directed** if all the arcs in the loop are undirected.

Definition 3 For a given ETG $G = (V, E)$, a **schedule** $\sigma : V_T \rightarrow \mathcal{R}$ assigns to each tasks t a start-time, where $\mathcal{R}$ is the set of real numbers. A schedule is called **feasible** if no precedence is violated.

To take advantage of conditional branches to reduce the number of PEs required, we defined **task compatibility** as follow.

Definition 4 For a given ETG $G = (V, E)$, the associated **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 $(t_1, t_2) \in F$.

The compatible graph is an abstraction of exclusive existence of the tasks, including but not limited by conditional branches. Figure 2 shows an example of ETG. In the ETG, task d is compatible with a. Since d can not be executed before a is completed, d can never be executed at the same time when a is executed. Similarly, d is also compatible with e. Since only one branch, x or y, can be true, either e or d can be executed. As a consequence, e or d can never be executed at the same time. On the other hand, d is not compatible with c. Because once branch x is taken, both d and c will be executed. Since there are no paths from c to d or vice versa, they have chances to be executed at the same time.

The system cost is the total cost of each type of PE multiplied by the maximum number of the type of PE occupied simultaneously. For convenience, we define two notations Γ and Ω for graph as following.

$$\begin{aligned} \Gamma &: \text{power set of all the nodes} \rightarrow \text{power set of all the arcs} \\ \Gamma(U) &\ni (U, \Gamma(U)) \text{ is a complete undirected graph} \end{aligned} \tag{1}$$

$$\begin{aligned} \Omega &: \text{the set of all graphs} \rightarrow \text{power set of all graphs} \\ \Omega(H) &\equiv \text{the minimal set of cliques which can cover the graph } H \end{aligned} \tag{2}$$

We then define the system cost formally as following.

Definition 5 Given an ETG $G = (V, E)$ and a PE cost function $\alpha : P \rightarrow \mathcal{R}$, the **system cost**

$$\text{Cost} : \text{the set of schedules} \rightarrow \mathcal{R} \tag{3}$$

is defined as

$$\text{Cost}(\sigma) = \sum_{\forall p \in P} \{\alpha(p) \cdot \max_{\forall x \in \mathcal{R}} o(\Omega(C_p(x)))\} \tag{4}$$

where

$$\begin{aligned} &C_p(x) = (A_p(x), \Gamma(A_p(x)) \cap F) \\ &A_p(x) = \{t \mid \sigma(t) \leq x < \sigma(t) + m_t,\ \forall t \ni p_t = p\} \\ &C = (V_T, F) \text{ is the graph associated with } G \\ &P \text{ is the set of PEs used in } G \end{aligned}$$

In Definition 5, $A_p(x)$ are the set of tasks which are all associated with PE type p and are executed at time x. $\Gamma(A_p(x)) \cap F$ is the subset of all the arcs F of the compatible graph

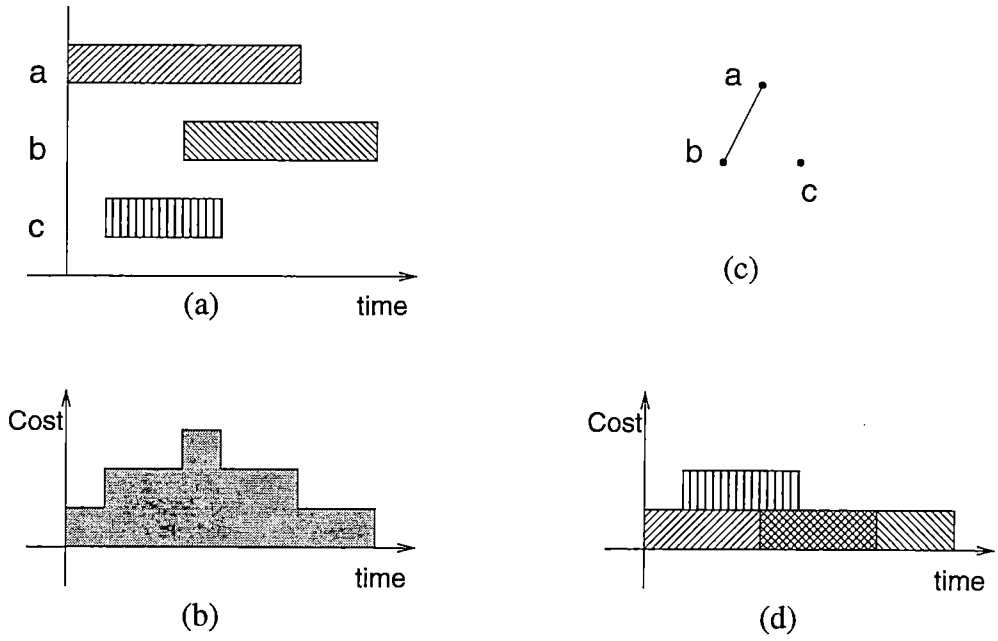


Figure 3: (a)A schedule of three tasks. (b) The total number of PEs required over time axis without considering task compatibility. (c) The compatible graph. (d) The total number of PEs required over time axis with considering compatibility.

C, where each arc in the subset has both vertexes in $A_p(x)$. Hence, $C_p(x)$ is the correlated sub-compatible-graph, and $o(\Omega(C_p(x)))$ is the total number of PE type p occupied at time x. Figure 3 shows a simplified example of the system cost. In the example, only one type of PE is used. Without considering task compatibility, the maximum number of PEs required is three. However, by definition we can schedule compatible tasks to the same PE. Thus, only two PEs are actually needed.

Based on the definition of the system cost, we can define the system-level time-constraint scheduling problem as following.

Definition 6 System-Level Time-Constraint Scheduling Problem: Given an ETG $G = (V, E)$, a PE cost function $\alpha : P \rightarrow \mathcal{R}$, and a time-constraint $q \in \mathcal{R}$, find a feasible schedule $\sigma : V_T \rightarrow \mathcal{R}$ such that

$$0 \leq \sigma(t) + m_t < q, \ \forall t \in V_T \ \text{ .and. } \ \text{Cost}(\sigma) \ \text{ is minimum} \tag{5}$$

4 Mathematical Statistic Model for System Cost

In this section, we present a mathematical statistic model to compute the expected system cost for partially scheduled ETGs. The model is then used in Section 5 for our system-level time-constraint scheduling algorithm to evaluate the quality of the intermediate schedules.

To build a statistic model for computing the expected system cost the intermediate schedules, we first need to formulate partially scheduled ETGs. We define it as following.

Definition 7 For a given ETG $G = (V, E)$ in which start-time of some tasks is already determined, the associated **partial schedule** $\pi : V_T \rightarrow \mathcal{R} \times \mathcal{R}$ can be defined as $\pi(t) = (\eta(t), \lambda(t))$ 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, if t is not scheduled.

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

Under the condition that no hint is provided, we assume the probability of scheduling an un-scheduled task t lies uniformly between $\eta(t)$ and $\lambda(t)$. Thus, we can define the PE distribution of a partial schedule as following.

Definition 8 For a given partial schedule $\pi : V_T \rightarrow \mathcal{R} \times \mathcal{R}$ which is associated with ETG $G = (V, E)$, the associated **PE distribution** $\delta : P \times \mathcal{R} \rightarrow \mathcal{R}$ is defined as

$$\delta(p, x) = \sum_{\forall (V_H, E_H) \in \Omega(C_p(x))} \max_{t \in V_H} \frac{m_t}{\lambda(t) - \eta(t) + m_t} \tag{6}$$

where

$$C_p(x) = (A_p(x), \Gamma(A_p(x)) \cap F) \tag{7}$$

$$A_p(x) = \{t \mid \eta(t) \leq x < \lambda(t) + m_t, \forall t \ni t_p = p\} \tag{8}$$

$C = (V_T, F)$ is the compatible graph associated with G

P is the set of PEs used in G

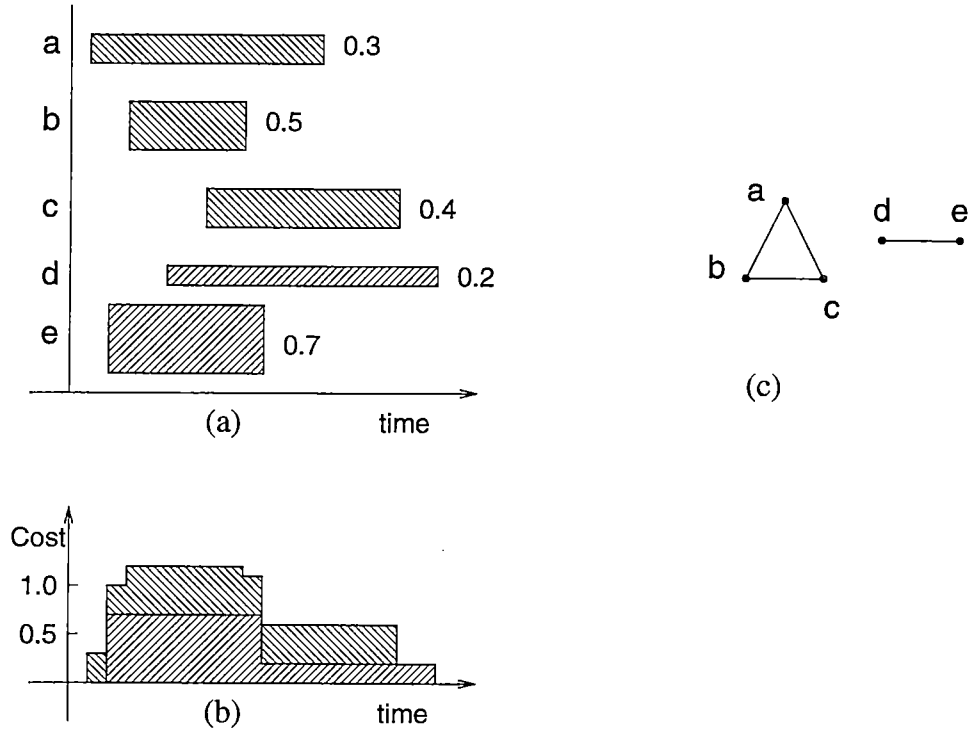


Figure 4: (a)The probabilities of five tasks. (b) The expected number of the PE over time axis. (c) The minimal set of cliques of the compatible graph.

In Definition 8, $A_p(x)$ are the set of tasks which are all associated with PE type p and can be executed at time x. $\Gamma(A_p(x)) \cap F$ is the subset of all the arcs F of the compatible graph C, where each arc in the subset has both vertexes in $A_p(x)$. Hence, $C_p(x)$ is the correlated sub-compatible-graph. Moreover, $\max_{t \in V_H} \frac{m_t}{\lambda(t)-\eta(t)+m_t}$ is the maximum value of the probability at x of each task in one clique, so the summation of all of these maximum values is the expected number of PE type p required at x. Figure 4 shows a simplified example of the PE distribution. In the example, only one type of PE is used. There are two cliques in the compatible graph. Since all the tasks connected in a clique can always be scheduled to the same PE, the maximum probability of these tasks can be the expected number of PE required by this group of tasks. The expect numbers of two task groups are accumulated as the expected number of this type of PE.

From Definition 8, we can multiple each of the expect numbers with the unit cost of the PE type, and accumulate them up to obtain the **expected system cost** Ex_Cost as following.

$$\text{Ex\_Cost} \ : \ \text{the set of partial schedules} \ \rightarrow \ \mathcal{R} \tag{9}$$

$$\text{Ex\_Cost}(\pi) \ = \ \textstyle\sum_{\forall p \in P}\{\alpha(p) \cdot \max_{\forall x \in \mathcal{R}} \delta(p,x)\} \tag{10}$$

Although the expected system cost is a discontinuous function over the *real number space*, yet we can compute it by just checking a set of points. We show how to easily compute the expected system cost as following.

Definition 9 For a given PE distribution $\delta : P \times \mathcal{R} \rightarrow \mathcal{R}$ associated with ETG $G = (V, E)$ and partial schedule $\pi : V_T \rightarrow \mathcal{R} \times \mathcal{R}$, the **distribution boundary** is a sorted list $B = \{x_1, x_2, \cdots, x_n\}$ such that

$$\eta(t), \eta(t) + m_t, \lambda(t), \lambda(t) + m_t \in B \ , \quad \forall t \in V_T \tag{11}$$

.and.

$$i \leq j \Leftrightarrow x_i \leq x_j \tag{12}$$

In Definition 9, Equation(11) defines the member of the distribution boundary, and Equation(12) defines the order of the member of the distribution boundary. Now we prove all points between two consequent elements of the distribution boundary are equal as following.

Theorem 1 Given a PE distribution $\delta : P \times \mathcal{R} \rightarrow \mathcal{R}$ associated with distribution boundary B, ETG $G = (V, E)$, and partial schedule $\pi : V_T \rightarrow \mathcal{R} \times \mathcal{R}$,

$$\forall b_i, b_{i+1} \in B \ , \ b_i \leq x < b_{i+1} \ , \ b_i \leq y < b_{i+1}$$

$$\Rightarrow \qquad \delta(p,x) = \delta(p,y) \tag{13}$$

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

$\Rightarrow x, y < \eta(t)$

or $\eta(t) \leq x < \lambda(t) + m_t$, $\eta(t) \leq y < \lambda(t) + m_t$

or $\lambda(t) + m_t \leq x, y$

$\Rightarrow A_p(x) = A_p(y)$ where A_p is defined as in Equation(8)

$\Rightarrow \delta(p, x) = \delta(p, y)$

q.e.d.

From Theorem 1, we can directly derive a more useful Theorem 2 as following.

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

Considering a partial schedule π and a task u which is going to be scheduled next, since there is no clock which can be used to align all the tasks in the system, we have unlimited start-time candidates for the task u. Thus, it is not possible to exhaustedly evaluate the expected system cost of each of the candidates to find the best start-time. In the rest of this section, we prove a theorem with which we can narrow them down to a finite number of candidates.

Given a partial schedule $\pi : V_T \rightarrow \mathcal{R} \times \mathcal{R}$, let $\pi_{u,y}(t) = (\eta_{u,y}(t), \lambda_{u,y}(t))$ denote one of π's **consequent partial schedules** where an un-schedule task u in π is scheduled to start-time y. Thus, we have

$$\eta_{u,y}(t) = \begin{cases} y & \text{if } t = u \\ \eta(t) + (y - \eta(u)) & \text{if } t \text{ is a successor of } u \\ \eta(t) & \text{otherwise} \end{cases} \tag{14}$$

$$\lambda_{u,y}(t) = \begin{cases} y & \text{if } t = u \\ \lambda(t) - (\lambda(u) - y) & \text{if } t \text{ is a predecessor of } u \\ \lambda(t) & \text{otherwise} \end{cases} \tag{15}$$

In case the task u is connected by synchronization arcs directly or indirectly with a set of tasks S, all the tasks in S have to start together with u. As a result, once u is scheduled to start at time y, all the tasks in S are also scheduled to start at time y at the same time.

Thus, Equation(14) and Equation(15) are extended as following.

$$\eta_{u,y}(t) = \begin{cases} y & \text{if } t \in S_u \\ \eta(t) + (y - \eta(u)) & \text{if } t \text{ is a successor of a task in } S_u \\ \eta(t) & \text{otherwise} \end{cases} \tag{16}$$

$$\lambda_{u,y}(t) = \begin{cases} y & \text{if } t \in S_u \\ \lambda(t) - (\lambda(u) - y) & \text{if } t \text{ is a predecessor of a task in } S_u \\ \lambda(t) & \text{otherwise} \end{cases} \tag{17}$$

where the **synchronized task set** S_u is defined as

$$S_u = S \cup \{u\} \tag{18}$$

From Equation(16), we can see

$$\begin{aligned} g &: \mathcal{R} \rightarrow \mathcal{R} \\ g(y) &= \eta_{u,y}(t) \end{aligned}$$

is a continuous and fixed-gradient function for any given t and u. Similarly, from Equation(17) we know

$$\begin{aligned} h &: \mathcal{R} \rightarrow \mathcal{R} \\ h(y) &= \lambda_{u,y}(t) \end{aligned}$$

is also a continuous and fixed-gradient function. With the above preparation, we now can prove the following theorem.

Theorem 3 For a given partial schedule $\pi : V_T \rightarrow \mathcal{R} \times \mathcal{R}$ and a given un-scheduled task u,

$$\min_{\eta(u) \leq z \leq \lambda(u)} \text{Ex\_Cost}(\pi_{u,z}) \tag{19}$$

can be found over a finite set of real numbers $\hat{B}$, where

$$\begin{aligned} &B \text{ is the distribution boundary} \\ &d = \lambda(u) - \eta(u) \\ &S_u \text{ is the synchronized task set of } u \\ &E_u = \{\eta(t) \mid t \text{ is a successor of a task in } S_u\} \end{aligned} \tag{20}$$

$$L_u = \{\lambda(t) \mid t \text{ is a predecessor of a task in } S_u\} \tag{21}$$

$$\dot{B}_y = \{b | b \in B, y \leq b \leq y + d\} \ , \ \forall y \in E_u \tag{22}$$

$$\ddot{B}_y = \{b | b \in B, y - d \leq b \leq y\} \ , \ \forall y \in L_u \tag{23}$$

$$\dot{B} = \{\eta(u) + b - y \mid \forall b \in \dot{B}_y, \forall y \in E_u\} \tag{24}$$

$$\ddot{B} = \{\lambda(u) - (y - b) \mid \forall b \in \ddot{B}_y, \forall y \in L_u\} \tag{25}$$

$$\hat{B} \text{ is the sorted list of } \dot{B} \cup \ddot{B} \tag{26}$$

proof:

Let $\pi_{u,z}$ denote one of π's consequent partial schedules where an un-scheduled task u in π is scheduled to start-time z. Let $\delta_{u,z}(p, x)$ denote the PE distribution associated with $\pi_{u,z}$. We define

$$f : \mathcal{R} \rightarrow \mathcal{R}$$

$$f(z) = \delta_{u,z}(p, x) \tag{27}$$

$$= \sum_{\forall (V_H, E_H) \in \Omega(C_p(x))} \max_{t \in V_H} \frac{m_t}{\lambda_{u,z}(t) - \eta_{u,z}(t) + m_t} \tag{28}$$

where $C_p(x)$ is defined as in Definition 8.

Now, we are going to prove that the function $f(z)$ is continuous and monotonic between any two consequent elements c_i and c_{i+1} of $\hat{B}$.

We first consider the case $x \leq z$. If task t is not a predecessor of a task in S_u, we have

$$\frac{m_t}{\lambda_{u,z}(t) - \eta_{u,z}(t) + m_t} \tag{29}$$

$$= \frac{m_t}{\lambda(t) - \eta(t) + m_t} \tag{30}$$

All the terms in Equation(30) are independent from z. Thus, Equation(30) is a constant for varied z. On the other hand, if task t is a predecessor of a task in S_u, we have

$$\frac{m_t}{\lambda_{u,z}(t) - \eta(t)_{u,z} + m_t} \tag{31}$$

$$= \frac{m_t}{\lambda(t) - (\lambda(u) - z) - \eta(t) + m_t} \tag{32}$$

$$= \frac{m_t}{c + z} \tag{33}$$

where

$$c \ = \ \lambda(t) - \eta(t) + m_t - \lambda(u) \tag{34}$$

is independent from z. When z is increasing, Equation(33) is decreasing. As a result, $f(z)$ is a continuous and decreasing function over the domain

$$\{ z \mid c_i \leq z \leq c_{i+1} \}$$

since each term inside the $\sum$ in Equation(28) is either a constant or a continuous and decreasing function, whereas $C_p(x)$ is identical for all z located between c_i and c_{i+1}.

Then we consider the case $x \geq z$. Similarly, we can derive that $f(z)$ is a continuous and increasing function over the domain

$$\{ z \mid c_i \leq z \leq c_{i+1} \}$$

Therefore, $f(z)$ is continuous and monotonic between any two consequent elements of $\hat{B}$.

As the consequence, we know all the local maximal and minimal values of $f(z)$ are all located on $\hat{B}$.

q.e.d.

5 The WLMF Algorithm

Since the scheduling problem is NP-hard[7], approximation algorithms which can find a good solution within predictable and acceptable time is favorite. Based on the properties obtained in Section 4, we develop a **weighted least-mobility first (WLMF)** algorithm that can find a near-optimal solution for system-level scheduling.

The WLMF algorithm is shown in Figure 5. The task compatible graph which is associated with the input ETG G is computed first. Once the compatible graph is obtained, the WLMF algorithm constructs a sequence of partial schedules, step by step toward the final schedule.

In each iteration an un-scheduled task is scheduled. The WLMF algorithm select the most critical task to schedule first. Intuitively, the mobility of the task and the cost of the PE which is associated with the task are two major factors which can determine the degree of urgency for scheduling. Other measures, for example, input/output degree of the task, can also be taken into account to determine the scheduling urgency. However, we compute the scheduling priority of the task only by its mobility and the PE cost associated with it in our implementation.

Once the candidate task t with the highest priority is selected, the WLMF algorithm assigns t to a start-time where the expected system cost Ex_Cost can be minimized. The

Algorithm WLMF

input

an acyclic ETG $G = (V_T \cup V_F \cup V_J \cup \{v_{start}, v_{end}\} , E_{SEQ} \cup E_{SYNC})$

a time-constraint

output

the scheduled ETG

begin

Compute the task compatible graph associated with G

$\pi = (\text{ASAP}(G) , \text{ALAP}(G))$

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 Equation(26)

find $z \in \hat{B}$ such that Ex_Cost($\pi_{t,z}$) is minimum

$\pi := \pi_{t,z}$

endwhile

end

Figure 5: Costly-Least-Mobility-First Scheduling Algorithm (WLMF)

statistical expected system cost of a partial schedule can be computed by Equation(10). According to Theorem 2, the WLMF algorithm can find the Ex_Cost by examining only the distribution boundaries B defined in Definition 9, instead of searching through the whole real number space. Whereas, with the help of Theorem 3, the WLMF 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 WLMF algorithm schedule t to the start-time, then the WLMF continues the next iteration. Finally, all the tasks are scheduled one by one until the schedule is completed.

In our system-level synthesis practice, tasks are assigned to PEs and synchronized by signals; contrary to practices of HLS, in which operations are synchronized by way of clock. In addition, a task can have variable execution time. The actual execution time may not be determined until the task is executed.

Average execution time or WCET can be used during scheduling, depends on the optimization objective. Figure 6 shows an example of WCET scheduling. In the example, we are trying to schedule an ETG as shown in Figure 6(b). The objective is to complete the ETG within the given time-constraint. The ETG is then scheduled according to WCET of each task. As the result, the ETG is scheduled as shown in Figure 6(c). Each box in Figure 6(c) indicates the schedule of a task. The gray shade in each box is the actual execution time, which is not known until the task is executed. Although the schedule is made according to WCET of each task and the actual execution time of each task is different from WCET, the time-constraint is still satisfied as well as the total order among all the tasks is still unchanged. Figure 6(d) shows the execution of the schedule. Each box in Figure 6(d) indicates the execution of a task.

In case the scheduling objective is average performance, we compute the schedule according to the average execution time of each task. As the result, the average case will be completed within the given time-constraint. Other execution time estimation can also be applied to the proposed technique, and result in the expected scheduling.

6 Experimental Results

The proposed system-level scheduling technique has been implemented as a part of the **System-Level Scheduler (SLS)**[14] of **SpecC System**[15, 16]. The SpecC System is an integrated synthesis environment, emphasized on IP-centric system design. It includes a C-based HDL and its compiler, simulation engines, GUI, exploration and refinement tools.

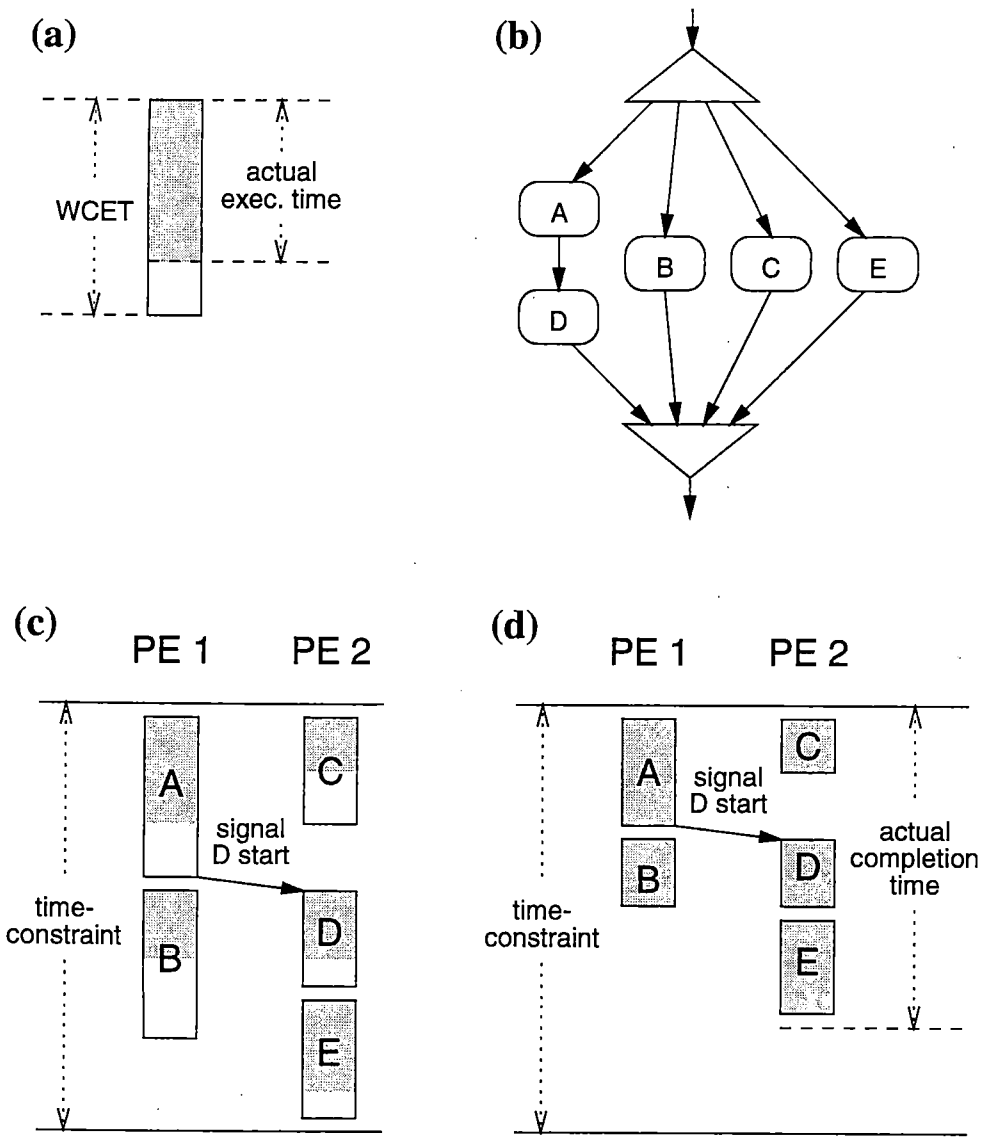


Figure 6: (a) The actual execution time and WCET of a task. (b) The ETG which is scheduled. (c) The schedule according to WCET. (d) Actual execution of the tasks.

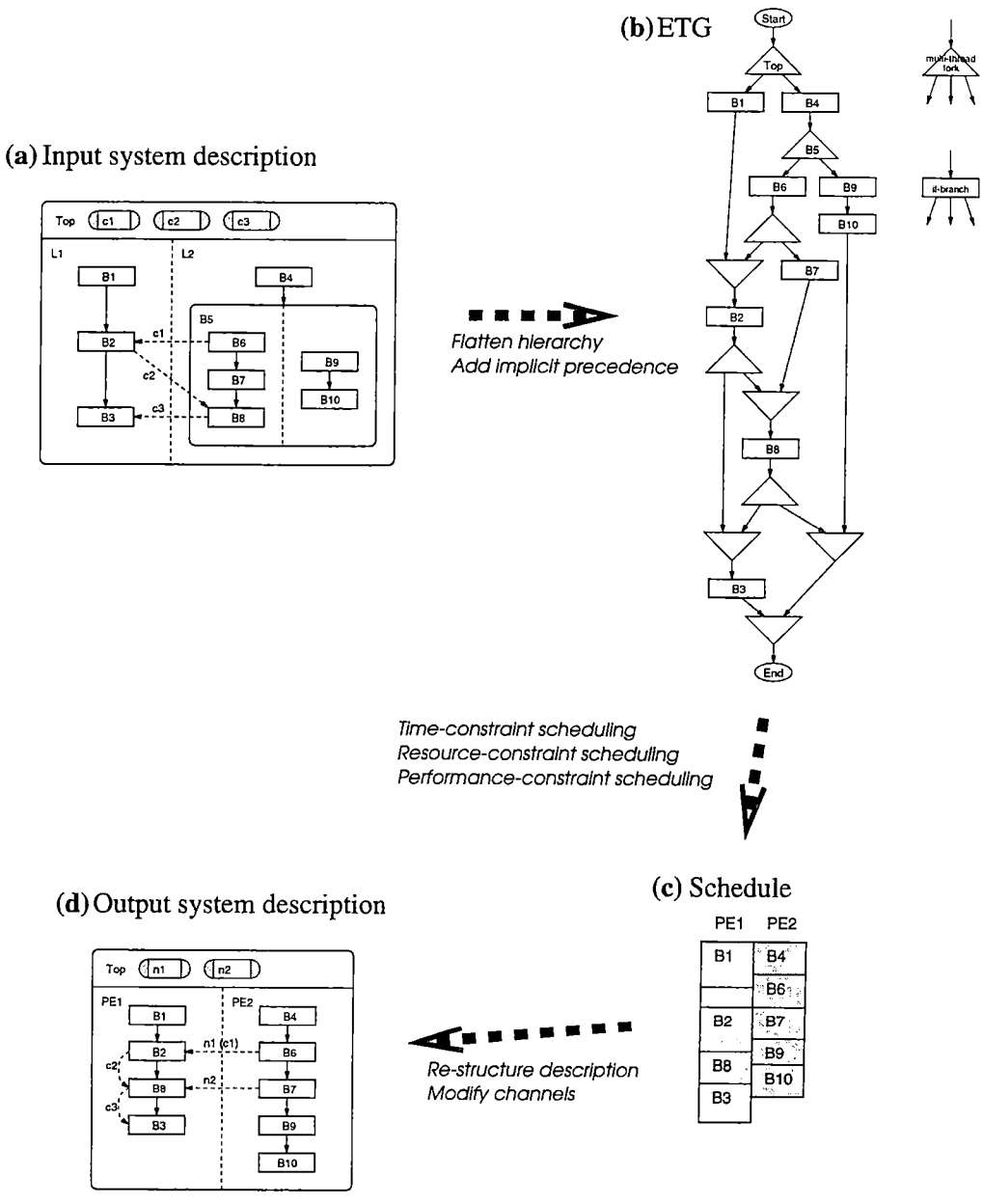


Figure 7: The SpecC system-level scheduling flow: (a) The input system description in SpecC; (b) The ETG; (c) The schedule obtained; (d) The refined SpecC description

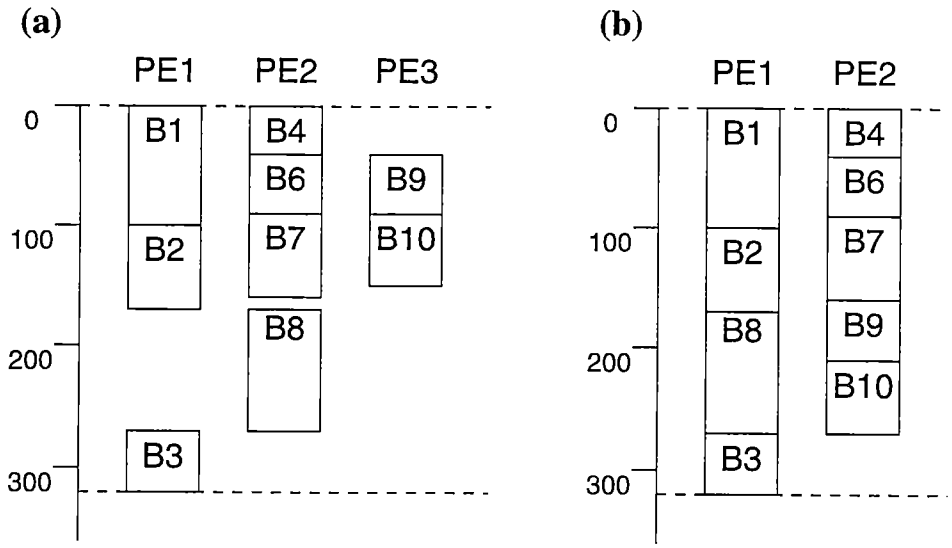


Figure 8: (a) Execution of the original description. (b) Execution of WLMF refined description. The time-constraint is set to 320, which is the same with the completion time of the original description.

	WCET		
	over all	case a	case b
find_az1	1240.51	n.a.	n.a.
find_az2	1240.51	n.a.	n.a.
az_lsp1	412.58	n.a.	n.a.
az_lsp2	412.58	n.a.	n.a.
vad_lp	571.32	81.50	571.32
int_lpc2	76.18	n.a.	n.a.
q_plsf_and _int_lpc	1313.91	1313.91	5.24
processor: 100 MHz Motorola DSP56600 time unit: micro second			

Table 1: Execution time of software implementation of tasks in the LP_analysis module

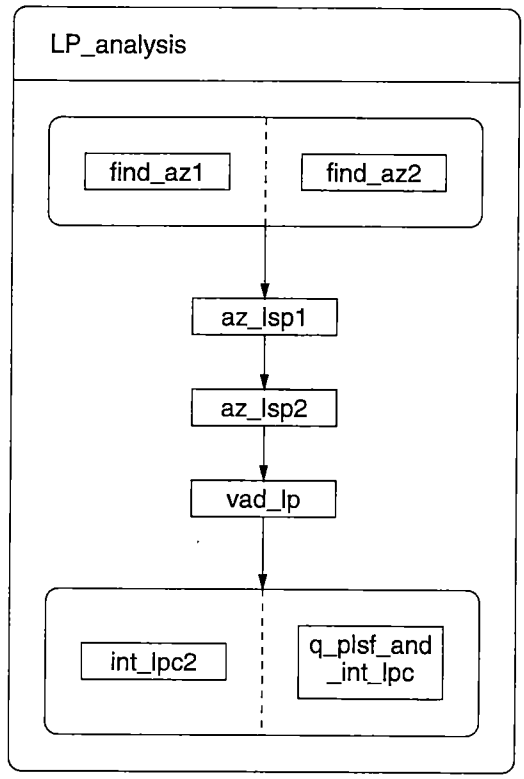


Figure 9: The original behavior structure of LP_analysis in SpecC

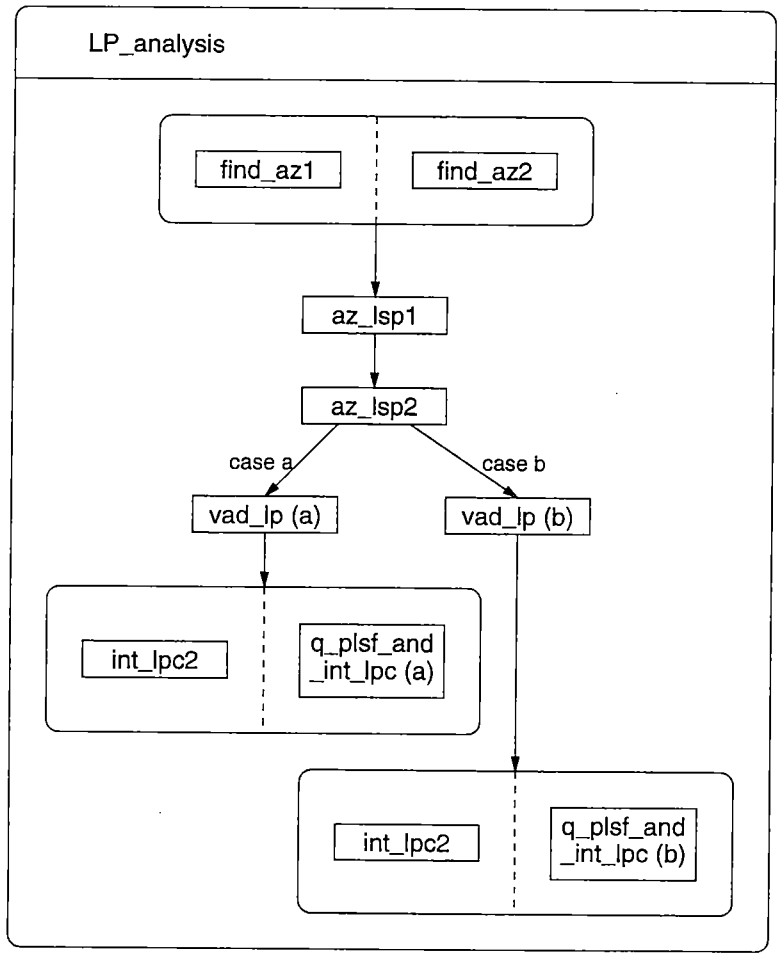


Figure 10: The new behavior structure of LP_analysis in SpecC

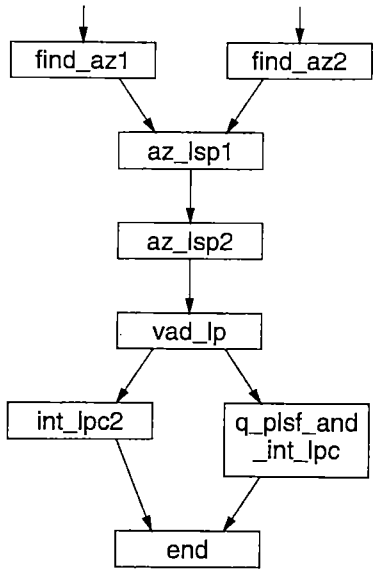


Figure 11: The DFG obtained from the original behavior structure of the LP_analysis

shows the behavior structure of LP_analysis in SpecC. The LP_analysis iterates to process input data frames. Each input data frame has to be processed within 5ms. The execution time of each task is shown in Table 1.

Figure 11 shows the DFG obtained from the behavior structure shown in Figure 9. With allocating 1 PE, the WCET of the DFG is 5.26759ms as shown in Figure 13(c), which exceeds the time-constraint given. In order to obtain WCET less than 5ms, two PEs are required for scheduling the DFG. As the result, a schedule with worst case DFG completion time 3.95090ms can be obtained as shown in Figure 13(b).

However, when we look closely into the task vad_lp, the WCET of a set of situations (case a) is 81.5μs; whereas the WCET of the rest situations (case b) is 571.32μs. On the other hand, when (case a) applies on q_plsf_and_int_lpc, the WCET is 1313.91μs; whereas the WCET of (case b) is 5.24μs. As a result, we can rewrite the behavior structure of LP_analysis as shown in Figure 10. The ETG can explicitly comprehend the concurrent and exclusive execution flow of the new LP_analysis as shown in Figure 12. With the help of the ETG, the WLMF can take advantage of concurrent and exclusive execution flow, and then obtains a better schedule as shown in Figure 13(a). In the schedule, only one PE is required to obtained the worst case ETG completion time 4.77777ms, which satisfied the time-constraint given.

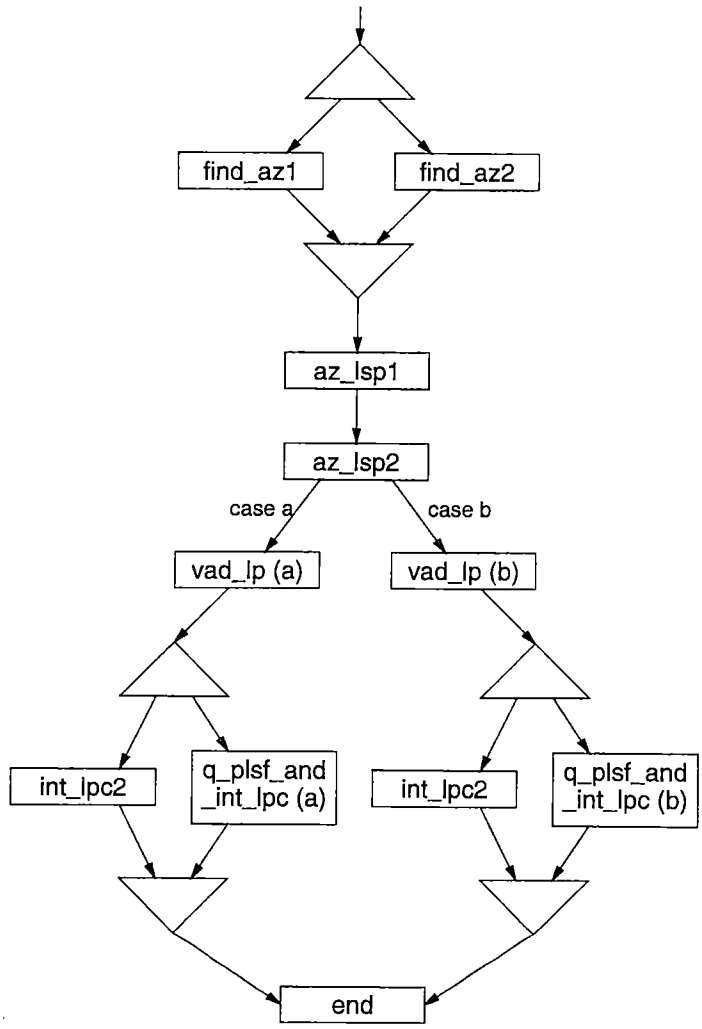


Figure 12: The ETG obtained from the new behavior structure of the LP_analysis

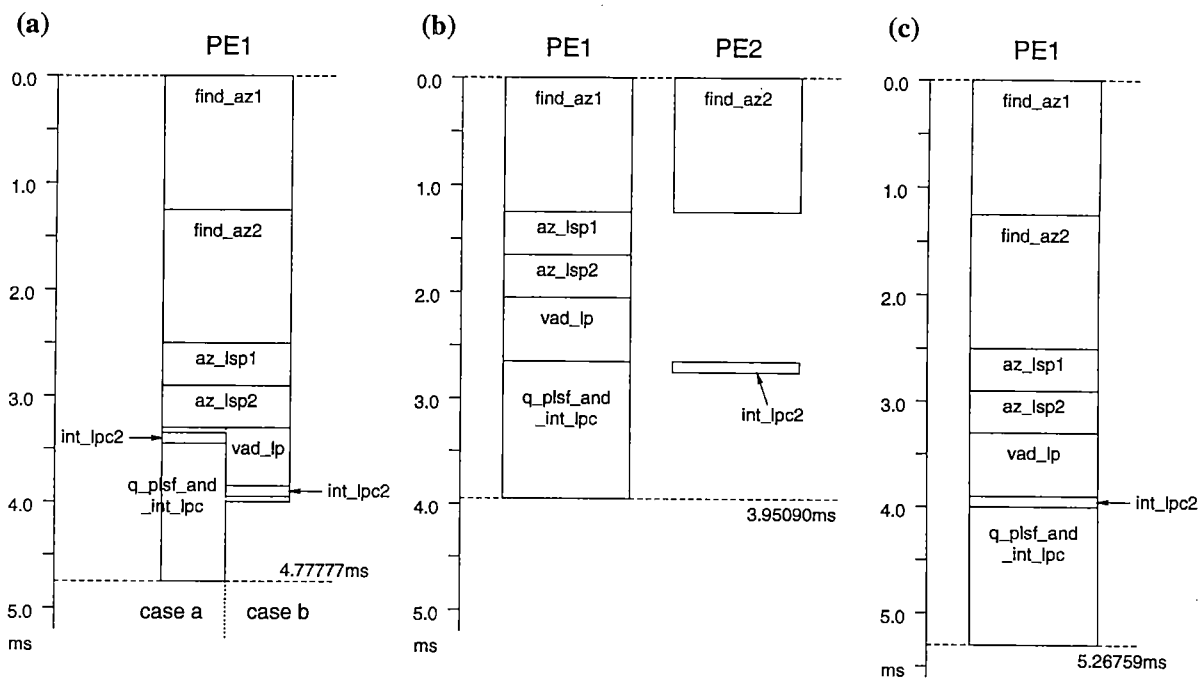


Figure 13: (a) The schedule of the ETG obtained from the new behavior structure of the LP_analysis. (b) The 2 PE schedule of the DFG obtained from the original behavior structure of the LP_analysis. (c) The 1 PE schedule of the DFG obtained from the original behavior structure of the LP_analysis.

7 Concluding Remarks

In the paper, we present an extended task graph ETG to capture the system-level scheduling problems. Based on the ETG, we define the system-level time-constraint scheduling problem. HLS scheduling algorithms can not be applied on system-level synthesis due to the following problems.

- The clock is no longer able to align the tasks at system-level;
- Combined concurrent and exclusive execution flows;
- Synchronization precedences among the tasks;
- Data-dependent execution time of the tasks.

We present the WLMF algorithm which can compute system-level time-constraint scheduling effectively and efficiently. The WLMF algorithm is based on mathematical statistics. Several statistical properties are developed to support the WLMF algorithm.

Static scheduling approach, which determines the execution sequence of the tasks while the target system is synthesized, is used in the paper. Static scheduling has following two advantages.

- Less OS overhead;
- Better system WCET.

The technique presented in the paper can obtain scheduling which meets *one* time-constraint. Methodologies which can obtain scheduling with multiple local time-constraints are still needed.

References

[1] D. Ziegenbein, K. Richter, R. Ernst, J. Teich, and L. Thiele, "Representation of process mode correlation for scheduling," in *Proceedings of the International Conference on Computer-Aided Design*, 1998.

[2] J. Lee, Y. Hsu, and Y. Lin, "A new integer linear programming formulation for the scheduling problem in datapath synthesis," in *Proceedings of the International Conference on Computer-Aided Design*, 1989.

[3] W. Chu and L.-T. Lan, "Task allocation and precedence relations for distributed real-time systems," *IEEE Transactions on Computers*, June 1987.

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

[5] S. Ha and E. Lee, "Compile-time scheduling of dynamic constructs in dataflow program graphs," *IEEE Transactions on Computer-Aided Design*, July 1997.

[6] P. Eles, K. Kuchcinski, Z. peng, A. Doboli, and P. Pop, "Scheduling of conditional process graphs for the synthesis of embedded systems," in *Proceedings of Conference on Design, Automation and Test in Europe*, 1998.

[7] J. Ullman, "NP-complete scheduling problem," *Journal of Comput. Syst. Sci.*, pp. 384–393, 1975.

[8] A. Burchard, Y. Oh, J. Liebeherr, and S. Son, "A linear-time online task assignment scheme for multiprocessor systems," in *11th IEEE Workshop Real-Time Operating Systems and Software*, 1994.

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

[10] L. Sha, R. Rajkumar, and S. Sathaye, "Generalized rate-monotonic scheduling theory: A framework for developing real-time systems," in *Proceedings of the IEEE*, January 1994.

[11] Y. Li and W. Wolf, "Hierarchical scheduling and allocation of multirate systems on heterogeneous multiprocessors," in *Proceedings of the International Conference on Computer-Aided Design*, 1998.

[12] P. Paulin and J. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," *IEEE Transactions on Computer-Aided Design*, June 1989.

[13] S. Devadas and A. Newton, "Algorithms for hardware allocation in data path synthesis," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 7, pp. 768–781, 1989.

[14] E.-S. Chang and D. Gajski, "SpecC system-level static scheduling." UC Irvine, Dept. of ICS, Technical Report 99-23,May 1999.

[15] D. Gajski, R. Dömer, and J. Zhu, "IP-centric methodology and design with the SpecC language," in *Contribution to NATO-ASI workshop on System Level Synthesis, Il Ciocco, Barga, Italy*, August 1998.

[16] D. Gajski, "IP-based design methodology," in *Proceedings of the Design Automation Conference*, 1999.

[17] 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.

[18] A. Gerstlauer, S. Zhao, and D. D. Gajski, "Design of a GSM vocoder using SpecC methodology." UC Irvine, Dept. of ICS, Technical Report 99-11,February 1999.