Timing Analysis of Concurrent Programs

Robert Mittermayr and Johann Blieberger

Institute of Computer-Aided Automation
Vienna University of Technology

WCET 2012, July 10, 2012
Outline

1 Overview
Outline

1. Overview

2. Kronecker Algebra
   - Kronecker Product
   - Kronecker Sum
Outline

1. Overview

2. Kronecker Algebra
   - Kronecker Product
   - Kronecker Sum

3. Concurrent Program Graphs (CPGs)
   - Properties of Concurrent Program Graphs
   - Implementation
Outline

1 Overview

2 Kronecker Algebra
   - Kronecker Product
   - Kronecker Sum

3 Concurrent Program Graphs (CPGs)
   - Properties of Concurrent Program Graphs
   - Implementation

4 WCET
   - Synchronizing Nodes
   - Dataflow Equations
Outline

1. Overview
2. Kronecker Algebra
   - Kronecker Product
   - Kronecker Sum
3. Concurrent Program Graphs (CPGs)
   - Properties of Concurrent Program Graphs
   - Implementation
4. WCET
   - Synchronizing Nodes
   - Dataflow Equations
5. Example
Overview

- We model concurrent programs by threads which use semaphores for synchronization
We model concurrent programs by threads which use semaphores for synchronization.

Threads and semaphores are represented by *Control Flow Graphs* (CFGs).
Overview

- We model concurrent programs by threads which use semaphores for synchronization.
- Threads and semaphores are represented by Control Flow Graphs (CFGs).
- We use Kronecker algebra to manipulate adjacency matrices and generate a whole system view.
Overview

- We model concurrent programs by threads which use semaphores for synchronization.
- Threads and semaphores are represented by *Control Flow Graphs* (CFGs).
- We use Kronecker algebra to manipulate adjacency matrices and generate a whole system view.
- Dataflow-based approach for generating WCET of concurrent program.
Kronecker algebra operates on the edges $\Rightarrow$ basic blocks on the edges.
From CFGs to Refined CFGs (RCFGs)

- Kronecker algebra operates on the edges $\Rightarrow$ basic blocks on the edges
- Basic blocks in CFGs may contain semaphore calls
From CFGs to Refined CFGs (RCFGs)

- Kronecker algebra operates on the edges \(\Rightarrow\) basic blocks on the edges
- Basic blocks in CFGs may contain semaphore calls
- In order to enable synchronization and generation of all possible interleavings we need to split such basic blocks
From CFGs to Refined CFGs (RCFGs)

- Kronecker algebra operates on the edges $\Rightarrow$ basic blocks on the edges
- Basic blocks in CFGs may contain semaphore calls
- In order to enable synchronization and generation of all possible interleavings we need to split such basic blocks
- Basic blocks are situated on the edges $\Rightarrow$ Edge Splitting
Kronecker algebra operates on the edges → basic blocks on the edges

Basic blocks in CFGs may contain semaphore calls

In order to enable synchronization and generation of all possible interleavings we need to split such basic blocks

Basic blocks are situated on the edges → Edge Splitting

Edge Splitting makes sure that semaphore calls are the only statement at an edge
From CFGs to Refined CFGs (RCFGs)

- Kronecker algebra operates on the edges $\Rightarrow$ basic blocks on the edges
- Basic blocks in CFGs may contain semaphore calls
- In order to enable synchronization and generation of all possible interleavings we need to split such basic blocks
- Basic blocks are situated on the edges $\Rightarrow$ Edge Splitting
- Edge Splitting makes sure that semaphore calls are the only statement at an edge
- Edge Splitting:
  - Input: CFG
  - Output: Refined CFG (RCFG)
Semaphores

- are a well-known vehicle for process synchronization
Semaphores

- are a well-known vehicle for process synchronization
- can be implemented efficiently
Semaphores

- are a well-known vehicle for process synchronization
- can be implemented efficiently
- are available in all operating systems
Semaphores

- are a well-known vehicle for process synchronization
- can be implemented efficiently
- are available in all operating systems
- two operations $p$ (blocking, aka lock) and $v$ (aka unlock)
Semaphores

- are a well-known vehicle for process synchronization
- can be implemented efficiently
- are available in all operating systems
- two operations $p$ (blocking, aka lock) and $v$ (aka unlock)

Matrix and CFG for the $i$th semaphore looks like this:

$$S(i) = \begin{pmatrix} 0 & p_i \\ v_i & 0 \end{pmatrix}$$
The so-called Kronecker algebra

- is sometimes being referred to as tensor algebra of matrices
The so-called Kronecker algebra

- is sometimes being referred to as tensor algebra of matrices
- operates on adjacency matrices
The so-called Kronecker algebra

- is sometimes being referred to as tensor algebra of matrices
- operates on adjacency matrices
- consists of Kronecker sum and Kronecker product
The so-called Kronecker algebra

- is sometimes being referred to as tensor algebra of matrices
- operates on adjacency matrices
- consists of Kronecker sum and Kronecker product

We use the

The so-called Kronecker algebra

- is sometimes being referred to as tensor algebra of matrices
- operates on adjacency matrices
- consists of Kronecker sum and Kronecker product

We use the

- Kronecker product or Zehfuss product [M. Davio, 1981] and [Zehfuss, 1858] for representing synchronization
Definition (Kronecker product)

Given a m-by-n matrix $A$ and an p-by-q matrix $B$, their Kronecker product denoted by $A \otimes B$ is an mp-by-nq block matrix defined by

$$A \otimes B = \begin{pmatrix} a_{1,1}B & \cdots & a_{1,n}B \\ \vdots & \ddots & \vdots \\ a_{m,1}B & \cdots & a_{m,n}B \end{pmatrix}.$$
### Example

Let \( A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \) and \( B = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix} \). The Kronecker product \( C = A \otimes B \) is given by

\[
C = \begin{pmatrix}
    a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,1}b_{1,3} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} & a_{1,2}b_{1,3} \\
    a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,1}b_{2,3} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} & a_{1,2}b_{2,3} \\
    a_{1,1}b_{3,1} & a_{1,1}b_{3,2} & a_{1,1}b_{3,3} & a_{1,2}b_{3,1} & a_{1,2}b_{3,2} & a_{1,2}b_{3,3} \\
    a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,1}b_{1,3} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} & a_{2,2}b_{1,3} \\
    a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,1}b_{2,3} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} & a_{2,2}b_{2,3} \\
    a_{2,1}b_{3,1} & a_{2,1}b_{3,2} & a_{2,1}b_{3,3} & a_{2,2}b_{3,1} & a_{2,2}b_{3,2} & a_{2,2}b_{3,3}
\end{pmatrix}.
\]
Let $A$, $B$, $C$ and $D$ matrices. Kronecker product is

- **Noncommutative:**
  In general $A \otimes B \neq B \otimes A$
Let $A$, $B$, $C$ and $D$ matrices. Kronecker product is

- **Noncommutative:**
  In general $A \otimes B \neq B \otimes A$

- **Distributivity over matrix addition:**
  \[(A + B) \otimes (C + D) = A \otimes C + B \otimes C + A \otimes D + B \otimes D\]
Let $A$, $B$, $C$ and $D$ matrices. Kronecker product is

- **Noncommutative:**
  In general $A \otimes B \neq B \otimes A$

- **Distributivity over matrix addition:**
  $$(A + B) \otimes (C + D) = A \otimes C + B \otimes C + A \otimes D + B \otimes D$$

- **Associativity:**
  $$A \otimes (B \otimes C) = (A \otimes B) \otimes C$$
Definition (Kronecker sum)

Given a m-by-m matrix $A$ and a n-by-n matrix $B$, their Kronecker sum denoted by $A \oplus B$ is a mn-by-mn matrix defined by

$$A \oplus B = A \otimes I_n + I_m \otimes B,$$  \hspace{1cm} \text{(2)}

where $I_m$ and $I_n$ denote the identity matrix\(^a\) of order\(^b\) $m$ and $n$, respectively.

\(^a\)The identity matrix $I_n$ is a n-by-n matrix with ones on the main diagonal and zeros elsewhere.

\(^b\)A k-by-k matrix is known as square matrix of order $k$. 

Example on matrix level 1/2

We use again $A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix}$ and $B = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}$.

$A \oplus B$ is given by $= A \otimes I_3 + I_2 \otimes B =$

\[
\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}
\]
Example on matrix level 2/2

\[
\begin{pmatrix}
a_{1,1} & 0 & 0 & a_{1,2} & 0 & 0 \\
0 & a_{1,1} & 0 & 0 & a_{1,2} & 0 \\
0 & 0 & a_{1,1} & 0 & 0 & a_{1,2} \\
a_{2,1} & 0 & 0 & a_{2,2} & 0 & 0 \\
0 & a_{2,1} & 0 & 0 & a_{2,2} & 0 \\
0 & 0 & a_{2,1} & 0 & 0 & a_{2,2} \\
\end{pmatrix}
+ 
\begin{pmatrix}
b_{1,1} & b_{1,2} & b_{1,3} & 0 & 0 & 0 \\
b_{2,1} & b_{2,2} & b_{2,3} & 0 & 0 & 0 \\
b_{3,1} & b_{3,2} & b_{3,3} & 0 & 0 & 0 \\
0 & 0 & 0 & b_{1,1} & b_{1,2} & b_{1,3} \\
0 & 0 & 0 & b_{2,1} & b_{2,2} & b_{2,3} \\
0 & 0 & 0 & b_{3,1} & b_{3,2} & b_{3,3} \\
\end{pmatrix}
= 
\begin{pmatrix}
a_{1,1} + b_{1,1} & b_{1,2} & b_{1,3} & a_{1,2} & 0 & 0 \\
b_{2,1} & a_{1,1} + b_{2,2} & b_{2,3} & 0 & a_{1,2} & 0 \\
b_{3,1} & b_{3,2} & a_{1,1} + b_{3,3} & 0 & 0 & a_{1,2} \\
a_{2,1} & 0 & 0 & a_{2,2} + b_{1,1} & b_{1,2} & b_{1,3} \\
0 & a_{2,1} & 0 & b_{2,1} & a_{2,2} + b_{2,2} & b_{2,3} \\
0 & 0 & a_{2,1} & b_{3,1} & b_{3,2} & a_{2,2} + b_{3,3} \\
\end{pmatrix}.
\]
Kronecker Sum 4/5 (Example)

Interleavings Example with RCFGs

(a) C

(b) D

(c) $C \oplus D$

In the following we list basic properties of the Kronecker sum of matrices $A$, $B$ and $C$.

- **Noncommutative using element-wise comparison:**
  
  In general $A \oplus B \neq B \oplus A$
In the following we list basic properties of the Kronecker sum of matrices $A$, $B$ and $C$.

- Noncommutative using element-wise comparison:
  In general $A \oplus B \neq B \oplus A$

- Commutative by structural isomorphism: It essentially commutes because the graphs of $A \oplus B$ and $B \oplus A$ are structurally isomorphic.
In the following we list basic properties of the Kronecker sum of matrices $A$, $B$ and $C$.

- Noncommutative using element-wise comparison:
  In general $A \oplus B \neq B \oplus A$

- Commutative by structural isomorphism: It essentially commutes because the graphs of $A \oplus B$ and $B \oplus A$ are structurally isomorphic.

- Associativity:
  The operation is also associative, as $(A \oplus B) \oplus C$ and $A \oplus (B \oplus C)$ are isomorphic.
Kronecker Algebra

The associativity properties of the operations $\otimes$ and $\oplus$ imply that the $n$-fold operations

$$
\bigotimes_{i=1}^{k} A_i \quad \text{and} \quad \bigoplus_{i=1}^{k} A_i
$$

are well defined.
Concurrent Program Graphs (CPGs)

We model interleavings and synchronization with Kronecker sum and product, respectively. The matrix $T = \bigoplus_{i=1}^{k} T(i)$ represents $k$ interleaved threads. The matrix $S = \bigoplus_{i=1}^{r} S(i)$ represents $r$ interleaved semaphores. $TS$ contains semaphore calls of $T$ only. $TV$ contains all other entries of $T$. Programs CPG $P = T_S \otimes S + T_V \otimes I_o(S)$, where

- First term: $s \cdot s = s$, otherwise $s \cdot t = 0$ iff $s \neq t$.
- Second term: $a \cdot 1 = a$ and $a \cdot 0 = 0$.  

Concurrent Program Graphs (CPGs)

- $T^{(i)}$ ... adj. matrix of thread $i$

We model interleavings and synchronization with Kronecker sum and product, respectively. The matrix $T = \bigoplus_{i=1}^{k} T^{(i)}$ represents $k$ interleaved threads. The matrix $S = \bigoplus_{i=1}^{r} S^{(i)}$ represents $r$ interleaved semaphores. The matrix $TS$ contains semaphore calls of $T$ only. $TV$ contains all other entries of $T$.

Programs $CPG = TS \otimes S + TV \otimes I_o(S)$, where

- First term: $s \cdot s = s$, otherwise $s \cdot t = 0$ iff $s \neq t$.
- Second term: $a \cdot 1 = a$ and $a \cdot 0 = 0$. 

R. Mittermayr and J. Blieberger (TU Wien) Timing Analysis of Concurrent Programs
Concurrent Program Graphs (CPGs)

- $T^{(i)}$ ... adj. matrix of thread $i$
- $S^{(i)}$ ... adj. matrix of semaphore $i$

We model interleavings and synchronization with Kronecker sum and product, respectively. The matrix $T = \bigoplus_{i=1}^{k} T^{(i)}$ represents $k$ interleaved threads. The matrix $S = \bigoplus_{i=1}^{r} S^{(i)}$ represents $r$ interleaved semaphores. $T \cdot S$ contains semaphore calls of $T$ only. $T \cdot V$ contains all other entries of $T$. Programs CPG $P = T \cdot S \otimes S + T \cdot V \otimes I$, where

First term:
$s \cdot s = s$, otherwise $s \cdot t = 0$ iff $s \neq t$.

Second term:
$a \cdot 1 = a$ and $a \cdot 0 = 0$. 

Concurrent Program Graphs (CPGs)

- $T(i)$ … adj. matrix of thread $i$
- $S(i)$ … adj. matrix of semaphore $i$
- We model interleavings and synchronization with Kronecker sum and product, respectively.

The matrix $T = \bigoplus_{i=1}^{k} T(i)$ represents $k$ interleaved threads.
The matrix $S = \bigoplus_{i=1}^{r} S(i)$ represents $r$ interleaved semaphores.
$T S$ contains semaphore calls of $T$ only.
$T V$ contains all other entries of $T$.

Programs $CPG = T S \otimes S + T V \otimes I$, where
First term: $s \cdot s = s$, otherwise $s \cdot t = 0$ iff $s \neq t$.
Second term: $a \cdot 1 = a$ and $a \cdot 0 = 0$. 
Concurrent Program Graphs (CPGs)

- $T^{(i)}$ . . . adj. matrix of thread $i$
- $S^{(i)}$ . . . adj. matrix of semaphore $i$
- We model interleavings and synchronization with Kronecker sum and product, respectively.

The matrix $T = \bigoplus_{i=1}^{k} T^{(i)}$ represents $k$ interleaved threads
Concurrent Program Graphs (CPGs)

- $T^{(i)} \ldots$ adj. matrix of thread $i$
- $S^{(i)} \ldots$ adj. matrix of semaphore $i$
- We model interleavings and synchronization with Kronecker sum and product, respectively.
- The matrix $T = \bigoplus_{i=1}^{k} T^{(i)}$ represents $k$ interleaved threads
- The matrix $S = \bigoplus_{i=1}^{r} S^{(i)}$ represents $r$ interleaved semaphores
Concurrent Program Graphs (CPGs)

- $T^{(i)}$ ... adj. matrix of thread $i$
- $S^{(i)}$ ... adj. matrix of semaphore $i$
- We model interleavings and synchronization with Kronecker sum and product, respectively.
- The matrix $T = \bigoplus_{i=1}^{k} T^{(i)}$ represents $k$ interleaved threads
- The matrix $S = \bigoplus_{i=1}^{r} S^{(i)}$ represents $r$ interleaved semaphores
- $T_S$ contains semaphore calls of $T$ only
Concurrent Program Graphs (CPGs)

- $T^{(i)}$ ... adj. matrix of thread $i$
- $S^{(i)}$ ... adj. matrix of semaphore $i$

We model interleavings and synchronization with Kronecker sum and product, respectively.

The matrix $T = \bigoplus_{i=1}^{k} T^{(i)}$ represents $k$ interleaved threads.

The matrix $S = \bigoplus_{i=1}^{r} S^{(i)}$ represents $r$ interleaved semaphores.

$T_S$ contains semaphore calls of $T$ only.

$T_V$ contains all other entries of $T$.
Concurrent Program Graphs (CPGs)

- $T^{(i)}$ ... adj. matrix of thread $i$
- $S^{(i)}$ ... adj. matrix of semaphore $i$
- We model interleavings and synchronization with Kronecker sum and product, respectively.

The matrix $T = \bigoplus_{i=1}^{k} T^{(i)}$ represents $k$ interleaved threads

The matrix $S = \bigoplus_{i=1}^{r} S^{(i)}$ represents $r$ interleaved semaphores

$T_S$ contains semaphore calls of $T$ only

$T_V$ contains all other entries of $T$

Programs CPG $P = T_S \otimes S + T_V \otimes I_o(S)$, where
Concurrent Program Graphs (CPGs)

- $T^{(i)}$ ... adj. matrix of thread $i$
- $S^{(i)}$ ... adj. matrix of semaphore $i$
- We model interleavings and synchronization with Kronecker sum and product, respectively.

The matrix $T = \bigoplus_{i=1}^{k} T^{(i)}$ represents $k$ interleaved threads.

The matrix $S = \bigoplus_{i=1}^{r} S^{(i)}$ represents $r$ interleaved semaphores.

$T_S$ contains semaphore calls of $T$ only.

$T_V$ contains all other entries of $T$.

Programs CPG $P = T_S \otimes S + T_V \otimes I_o(S)$, where

- First term: $s \cdot s = s$, otherwise $s \cdot t = 0$ iff $s \neq t$. 

Concurrent Program Graphs (CPGs)

- $T^{(i)}$ ... adj. matrix of thread $i$
- $S^{(i)}$ ... adj. matrix of semaphore $i$
- We model interleavings and synchronization with Kronecker sum and product, respectively.

The matrix $T = \bigoplus_{i=1}^{k} T^{(i)}$ represents $k$ interleaved threads.

The matrix $S = \bigoplus_{i=1}^{r} S^{(i)}$ represents $r$ interleaved semaphores.

$T_S$ contains semaphore calls of $T$ only.

$T_V$ contains all other entries of $T$.

Programs CPG $P = T_S \otimes S + T_V \otimes I_o(S)$, where

- First term: $s \cdot s = s$, otherwise $s \cdot t = 0$ iff $s \neq t$.
- Second term: $a \cdot 1 = a$ and $a \cdot 0 = 0$. 
Properties of CPGs

- $k \ldots$ number of threads
- $n \ldots$ number of nodes in each RCFG

$\Rightarrow$ Programs’ CPG

- has at most $n^k$ nodes
- has $2k \cdot n^k$ edges
- is a sparse graph as $|E| = O(|V|)$
Example

(d) $T_1$

(e) $T_2$
Example

(a) $T_1$

1

$T_1.p_1$

2

$T_1.p_2$

3

$T_1.a$

4

$T_1.v_2$

5

$T_1.v_1$

6

(b) $T_2$

1

$T_2.p_2$

2

$T_2.p_1$

3

$T_2.b$

4

$T_2.v_1$

5

$T_2.v_2$

6

(c) Resulting CPG

1

$T_2.p_2$ $T_1.p_1$

6

$T_2.p_1$ $T_1.p_1$ $T_2.p_2$ $T_1.p_2$

12

$T_2.b$ $T_1.a$

16

$T_2.v_1$ $T_1.v_2$

18

$T_2.v_2$ $T_1.p_1$ $T_2.p_2$ $T_1.v_1$

21

$T_1.p_1$ $T_2.v_2$

44

104

121

47

126

72

132

96

136

119

138

141
Some Unreachable Parts

Figure: Some Unreachable Parts of the Deadlock Example
Lazy Implementation

- Only small parts (23 nodes) of the graph ($T_1$-$T_2$-System with 144 nodes) are reachable from the entry node
Lazy Implementation

- Only small parts (23 nodes) of the graph (\( T_1 - T_2 \)-System with 144 nodes) are reachable from the entry node.
- We refer to the reachable part of a CPG as RCPG.
Lazy Implementation

- Only small parts (23 nodes) of the graph ($T_1$-$T_2$-System with 144 nodes) are reachable from the entry node
- We refer to the reachable part of a CPG as RCPG
- Always calculating all matrix entries would be a overkill
Lazy Implementation

- Only small parts (23 nodes) of the graph (\(T_1-T_2\)-System with 144 nodes) are reachable from the entry node.
- We refer to the reachable part of a CPG as RCPG.
- Always calculating all matrix entries would be an overkill.
- \(\Rightarrow\) Lazily calculate the matrix entries from the entry node on.
We apply a dataflow based approach introduced in [J. Blieberger, 2002]
We apply a dataflow based approach introduced in [J. Blieberger, 2002]

Dataflow equations are set up
We apply a dataflow based approach introduced in [J. Blieberger, 2002]
Dataflow equations are set up
Solved according to [Sreedhar, Gao, Lee, 1998]
We apply a dataflow based approach introduced in [J. Blieberger, 2002]

Dataflow equations are set up

Solved according to [Sreedhar, Gao, Lee, 1998]

Each RCPG node is assigned a dataflow variable

Each component of the vector reflects a processor and is used to calculate the WCET of the corresponding thread

We assume exactly one thread per CPU

Each thread executes its next statement if the thread is not blocked
We apply a dataflow based approach introduced in [J. Blieberger, 2002]

Dataflow equations are set up

Solved according to [Sreedhar, Gao, Lee, 1998]

Each RCPG node is assigned a dataflow variable

Each dataflow variable is represented by a vector
We apply a dataflow based approach introduced in [J. Blieberger, 2002].

Dataflow equations are set up.

Solved according to [Sreedhar, Gao, Lee, 1998].

Each RCPG node is assigned a dataflow variable.

Each dataflow variable is represented by a vector.

Each component of the vector reflects a processor and is used to calculate the WCET of the corresponding thread.
We apply a dataflow based approach introduced in [J. Blieberger, 2002]

Dataflow equations are set up

Solved according to [Sreedhar, Gao, Lee, 1998]

Each RCPG node is assigned a dataflow variable

Each dataflow variable is represented by a vector

Each component of the vector reflects a processor and is used to calculate the WCET of the corresponding thread

We assume exactly one thread per CPU
We apply a dataflow based approach introduced in [J. Blieberger, 2002].

Dataflow equations are set up.

Solved according to [Sreedhar, Gao, Lee, 1998].

Each RCPG node is assigned a dataflow variable.

Each dataflow variable is represented by a vector.

Each component of the vector reflects a processor and is used to calculate the WCET of the corresponding thread.

We assume exactly one thread per CPU.

Each thread executes its next statement if the thread is not blocked.
Let the vector $\mathbf{X} = (X_1, \ldots, X_\ell, \ldots, X_p)^\top$. We write $\mathbf{X}^{(\ell)} = X_\ell$ to denote the $\ell$th component of vector $\mathbf{X}$.

**Definition**

Let $\mathbf{X} = (X_1, \ldots, X_p)^\top$ and $\mathbf{Y} = (Y_1, \ldots, Y_p)^\top$. Then we define

$$\max(\mathbf{X}, \mathbf{Y}) := (\max(X_1, Y_1), \ldots, \max(X_p, Y_p))^\top.$$
A synchronizing node is a RCPG node $s$ such that

- there exists an edge $e_{in} = (i, s)$ with label $v_k$
A \textit{synchronizing node} is a RCPG node $s$ such that

- there exists an edge $e_{in} = (i, s)$ with label $v_k$ and
- there exists an edge $e_{out} = (s, j)$ with label $p_k$,

where $k$ denotes the same semaphore and $e_{in}$ and $e_{out}$ are mapped to different processors, i.e., $\mathcal{P}(e_{in}) \neq \mathcal{P}(e_{out})$. 
If $n$ is a non-synchronizing node, then

$$\bar{x}_n = \max_{k \in \text{Pred}(n)} \left( \bar{x}_k + t(k \rightarrow n) \right),$$

- $\text{Pred}(n)$ ... set of predecessor nodes of node $n$
- The $\ell$th component of vector $t(k \rightarrow n)$ is the time assigned to edge $k \rightarrow n$
- Edge $k \rightarrow n$ is mapped to processor $\ell$
- The other components of $t(k \rightarrow n)$ are zero.
Let $s$ be a synchronizing node. In addition, let $\pi_i$ and $\pi_j$ be the processors which the edges $i \rightarrow s$ and $s \rightarrow j$ are mapped to, i.e, $\pi_i = \Psi(i \rightarrow s)$ and $\pi_j = \Psi(s \rightarrow j)$. 

Then for $\ell \neq \pi_j$

$$X(s) = \max_{k \in \text{Pred}(s)} (X(k) + t(k \rightarrow s) \ell)$$

and

$$X(\pi_j) = \max(X(\pi_i) + t(i \rightarrow s) \pi_i, \max_{k: \Psi(k \rightarrow s) = \pi_j} (X(k) + t(k \rightarrow s) \pi_j))$$

where the first term considers the incoming v-edge and the second term takes into account all incoming edges of the blocking thread running on processor $\pi_j$. 
Let $s$ be a synchronizing node. In addition, let $\pi_i$ and $\pi_j$ be the processors which the edges $i \rightarrow s$ and $s \rightarrow j$ are mapped to, i.e, $\pi_i = \mathcal{P}(i \rightarrow s)$ and $\pi_j = \mathcal{P}(s \rightarrow j)$. Then for $\ell \neq \pi_j$

$$x^{(\ell)}_s = \max_{k \in \text{Pred}(s)} \left( x^{(\ell)}_k + t(k \rightarrow s)^{(\ell)} \right)$$
Let \( s \) be a synchronizing node. In addition, let \( \pi_i \) and \( \pi_j \) be the processors which the edges \( i \rightarrow s \) and \( s \rightarrow j \) are mapped to, i.e, \( \pi_i = \mathcal{B}(i \rightarrow s) \) and \( \pi_j = \mathcal{B}(s \rightarrow j) \). Then for \( \ell \neq \pi_j \)

\[
X_s^{(\ell)} = \max_{k \in \text{Pred}(s)} \left( X_k^{(\ell)} + t(k \rightarrow s)^{(\ell)} \right)
\]

and

\[
X_s^{(\pi_j)} = \max \left( X_i^{(\pi_i)} + t(i \rightarrow s)^{(\pi_i)}, \max_{k: \mathcal{B}(k \rightarrow s) = \pi_j} \left( X_k^{(\pi_j)} + t(k \rightarrow s)^{(\pi_j)} \right) \right)
\]

where the first term considers the incoming v-edge and the second term takes into account all incoming edges of the blocking thread running on processor \( \pi_j \).
Solving of the Dataflow Equations

- Dataflow equations can be solved by applying [Sreedhar, Gao, Lee, 1998]
Solving of the Dataflow Equations

- Dataflow equations can be solved by applying [Sreedhar, Gao, Lee, 1998]
- It relies on two operations:
  - inserting one equation into another
Solving of the Dataflow Equations

- Dataflow equations can be solved by applying [Sreedhar, Gao, Lee, 1998]
- It relies on two operations:
  - inserting one equation into another
  - solving recursions by so-called loop breaking
Solving of the Dataflow Equations

- Dataflow equations can be solved by applying [Sreedhar, Gao, Lee, 1998]
- It relies on two operations:
  - inserting one equation into another
  - solving recursions by so-called loop breaking
- The order of these operations is completely determined by the DJ graph introduced in [Sreedhar, Gao, Lee, 1998].
in contrast to [J. Blieberger, 2002] where CFGs are studied, RCPGs contain several copies of basic blocks in different places.
in contrast to [J. Blieberger, 2002] where CFGs are studied, RCPGs contain several copies of basic blocks in different places.

Thus, during loop breaking the number of loop iterations cannot be determined immediately.
Solving of the Dataflow Equations

- in contrast to [J. Blieberger, 2002] where CFGs are studied, RCPGs contain several copies of basic blocks in different places.
- Thus, during loop breaking the number of loop iterations cannot be determined immediately.
- We postpone the assigning of loop iterations and indicate this by "*".
Solving of the Dataflow Equations

- In contrast to [J. Blieberger, 2002] where CFGs are studied, RCPGs contain several copies of basic blocks in different places.
- Thus, during loop breaking the number of loop iterations cannot be determined immediately.
- We postpone the assigning of loop iterations and indicate this by "∗".
- After solving the equations we distribute the known number of loop iterations among all terms labeled by "∗" such that the timing values achieve their maxima.
Example

(a) RCFG of thread T1

(b) RCFG of thread T2
The dashed nodes 7 and 25 are the only synchronizing nodes.
Some Equations

\[ x_1 = \max \left( x_7 + \left( \frac{0}{d} \right), x_{25} + \left( \frac{b}{0} \right) \right) \]
Some Equations

\[ x_1 = \max \left( x_7 + (0), x_{25} + (b) \right) \]

\[ x_{25} = \left( \max \left( x_{18}^{(1)} + \nu, x_{31}^{(2)} + d \right) \right) \]
Some Equations

\[ x_1 = \max \left( x_7 + \left( \begin{array}{c} 0 \\ d \end{array} \right), x_{25} + \left( \begin{array}{c} b \\ 0 \end{array} \right) \right) \]

\[ x_{25} = \left( \begin{array}{c} \max \left( x_{18}^{(1)} + v, x_{31}^{(2)} + d \right) \\ x_{18}^{(1)} + v \end{array} \right) \]

\[ x_7 = \left( \begin{array}{c} \max \left( x_6^{(2)} + v, x_{31}^{(1)} + b \right) \\ x_6^{(2)} + v \end{array} \right) \]

\[ x_{18} = \max \left( x_{10} + \left( \begin{array}{c} a \\ 0 \end{array} \right), x_{24} + \left( \begin{array}{c} 0 \\ d \end{array} \right) \right) \]

\[ x_{24} = x_{16} + \left( \begin{array}{c} a \\ 0 \end{array} \right) \]

Example for insertions:

\[ 24 \rightarrow 18 : x_{18} = \max \left( x_{10} + \left( \begin{array}{c} a \\ 0 \end{array} \right), x_{16} + \left( \begin{array}{c} a \\ d \end{array} \right) \right) \]
WCET of our example:

\[
\text{WCET} = \max(\mathcal{X}_1^{(1)}, \mathcal{X}_1^{(2)}) = M_1^* + T_1^* + T_2^* + \alpha, \text{ where}
\]

- \( \alpha = p + a + v \),
- \( \gamma = p + c + v \),
- \( M_2 = \alpha + \gamma + T_2^* \), and
- \( M_1 = \max(T_1, M_2) \)
If $T_1$ and $T_2$ loop $r$ and $s$ times, respectively and $a = c = d = v = p = 1$ and $b = 10$, then we get (non-automatized) the WCET

$$\text{WCET} = \begin{cases} 
14 \left\lfloor \frac{s-1}{3} \right\rfloor + 13 \left( r - \left\lfloor \frac{s-1}{3} \right\rfloor \right) + 3 & \text{if } r > \left\lfloor \frac{s-1}{3} \right\rfloor, \\
14(r - 1) + 4(s - 3(r - 1)) + 3 & \text{if } r \leq \left\lfloor \frac{s-1}{3} \right\rfloor.
\end{cases}$$

**Figure:** A Simple Schedule
We established a framework for WCET analysis of concurrent systems

We construct a graph-based model out of CFGs using Kronecker algebra

Semaphores are used to model synchronization

Our graph representation (CPG) plays a similar role for concurrent systems as CFGs do for sequential programs

Open issues in distributing the "*"-terms of the resulting WCET formula
References


Thank you for your attention!

Questions?