Improving the Performance of Periodic Real-time Processes: a Graph Theoretical Approach

Ton Boode
Hajo Broersma
Jan Broenink

Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, The Netherlands
August 26, 2013
Overview

- Periodic real-time processes represented by graphs
Overview

- Periodic real-time processes represented by graphs
- *Cartesian product* $H_i \square H_j$
Overview

- Periodic real-time processes represented by graphs
- Cartesian product $H_i \Box H_j$
- **Weak synchronised product** $H_i \Box H_j$
Overview

- Periodic real-time processes represented by graphs
- Cartesian product $H_i \boxtimes H_j$
- Weak synchronised product $H_i \boxslash H_j$
- Reduced weak synchronised product $H_i \boxdot H_j$
Overview

- Periodic real-time processes represented by graphs
- Cartesian product $H_i \square H_j$
- Weak synchronised product $H_i \Box H_j$
- Reduced weak synchronised product $H_i \bigotimes H_j$
- **Synchronised product** $H_i \boxdot H_j$
Overview

- Periodic real-time processes represented by graphs
- Cartesian product $H_i \Box H_j$
- Weak synchronised product $H_i \sqsubset H_j$
- Reduced weak synchronised product $H_i \Box H_j$
- Synchronised product $H_i \boxdot H_j$
- **Performance gain, necessary and sufficient conditions**
Overview

- Periodic real-time processes represented by graphs
- Cartesian product $H_i \sqcap H_j$
- Weak synchronised product $H_i \sqsupset H_j$
- Reduced weak synchronised product $H_i \sqsupseteq H_j$
- Synchronised product $H_i \boxdot H_j$
- Performance gain, necessary and sufficient conditions
  - **Future work**
Parallel processes represented by graphs

OBJECT DISTANCE =
read_distance_sensors
compute_object_distance
distance_meas
→ SKIP

ROBOT SPEED =
distance_meas
compute_robot_speed
robot_speed
→ SKIP

MOTOR SPEED =
robot_speed
compute_motor_speed
write_motor_speed_setpoint
→ SKIP

SEQUENCE CONTROL =
(OBJECT DISTANCE ∥ ROBOT SPEED ∥ MOTOR SPEED);
SEQUENCE CONTROL;
Parallel processes represented by graphs

\[ \text{SQ} = \text{MS} + \text{RS} + \text{OD} \]

- Asynchronous
- Synchronous
Parallel processes represented by graphs

\[ MS = (V(H_1), A(H_1), \{\lambda(a)|a \in A(H_1)\}) \]
\[ = (\{v_1, v_2, v_3, v_4\}, \{v_1 v_2, v_2 v_3, v_3 v_4\}, \{(v_1 v_2, rs), (v_2 v_3, cms), (v_3 v_4, wmss)\}) \]

\[ RS = (V(H_2), A(H_2), \{\lambda(a)|a \in A(H_2)\}) \]
\[ = (\{v_5, v_6, v_7, v_8\}, \{v_5 v_6, v_6 v_7, v_7 v_8\}, \{(v_5 v_6, dm), (v_6 v_7, crs), (v_7 v_8, rs)\}) \]

\[ OD = (V(H_3), A(H_3), \{\lambda(a)|a \in A(H_3)\}) \]
\[ = (\{v_9, v_{10}, v_{11}, v_{12}\}, \{v_9 v_{10}, v_{10} v_{11}, v_{11} v_{12}\}, \{(v_9 v_{10}, rds), (v_{10} v_{11}, cod), (v_{11} v_{12}, dm)\}) \]
Cartesian product
Weak synchronised product
Reduced weak synchronised product
Synchronised product intermediate stage

Intermediate stage
Synchronised product
Multi dimensional pathological example

\[ H_1 + H_2 + H_3 \]

\[ H_1 \square H_2 + H_3 \]

\[ (H_1 \square H_2) \square H_3 \]
Lemma 5

Let $H_i$ be an acyclic graph for $i = 1, 2, \ldots, k$, where $k \geq 2$. Then $\ell(\square H_i) = \ell(H_1) + \ell(H_2) + \ldots + \ell(H_k)$ if and only if every $H_i$ has at least one longest path without synchronising arcs.
Lemma 6

Let $H_i$ be an acyclic graph for $i = 1, 2, \ldots, k$, where $k \geq 2$. Then $\ell(\Box H_i) < \ell(\square H_i)$ if there exists $H_n, H_m$, $n \neq m, 1 \leq n, m \leq k$, such that each longest path in $H_n, H_m$, contains at least one same labelled synchronising arc.
Theorem 1

Let $H_i$ be an acyclic graph for $i = 1, 2, \ldots, k$, where $k \geq 2$. Then $\ell(\Box H_i) < \ell(\square H_i)$ if there exists $H_n, H_m$, $n \neq m$, $1 \leq n, m \leq k$, such that each longest path in $H_n$, contains at least one synchronising arc and there is at least one longest path with a same labelled synchronisation arc in $H_m$. 
Future work

- Algorithms for optimising the performance gain
Future work

- Algorithms for optimising the performance gain
- The number of longest paths in a graph is exponential
Future work

- Algorithms for optimising the performance gain
- The number of longest paths in a graph is exponential
- **Scheduling of the synchronised product with internal deadlines**
Future work

- Algorithms for optimising the performance gain
- The number of longest paths in a graph is exponential
- Scheduling of the synchronised product with internal deadlines
- Memory usage
Future work

- Algorithms for optimising the performance gain
- The number of longest paths in a graph is exponential
- Scheduling of the synchronised product with internal deadlines
- Memory usage
- Synchronised product, associativity and commutativity
Future work

- Algorithms for optimising the performance gain
- The number of longest paths in a graph is exponential
- Scheduling of the synchronised product with internal deadlines
- Memory usage
- Synchronised product, associativity and commutativity
- **Decomposition of a component into its prime factors**
Future work

- Algorithms for optimising the performance gain
- The number of longest paths in a graph is exponential
- Scheduling of the synchronised product with internal deadlines
- Memory usage
- Synchronised product, associativity and commutativity
- Decomposition of a component into its prime factors
- **Constraints for the prime factors of the synchronised product**
Future work

- Algorithms for optimising the performance gain
- The number of longest paths in a graph is exponential
- Scheduling of the synchronised product with internal deadlines
- Memory usage
- Synchronised product, associativity and commutativity
- Decomposition of a component into its prime factors
- Constraints for the prime factors of the synchronised product
- Algorithm that calculates prime factors.
Future work

- Algorithms for optimising the performance gain
- The number of longest paths in a graph is exponential
- Scheduling of the synchronised product with internal deadlines
- Memory usage
- Synchronised product, associativity and commutativity
- Decomposition of a component into its prime factors
- Constraints for the prime factors of the synchronised product
- Algorithm that calculates prime factors.
- **An example of the decomposition of a graph**
Decomposition of the original graph into its prime factors
Decomposition of the original graph into its prime factors

\[ H = H_1 + H_2 + H_3 \]
Decomposition of the original graph into its prime factors
Decomposition of the original graph into its prime factors
Decomposition of the original graph into its prime factors
Decomposition of the original graph into its prime factors

intermediate stage \( (H_1 \otimes H_2 \otimes H_2) \)
Decomposition of the original graph into its prime factors

$H \neq H_1 \square H_2 \square H_3$
Decomposition of a component into its prime factors
Memory usage versus performance using decomposition

![Graph showing memory usage over time with a peak at m and a drop at d.](image)
Thank you for listening!
Improving the Performance of Periodic Real-time Processes: a Graph Theoretical Approach

Ton Boode
Hajo Broersma
Jan Broenink

Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, The Netherlands
August 26, 2013