Petri Net Based Cyclic Scheduling

Zdeněk Hanzálek
Trnka Laboratory for Automatic Control
Department of Control Engineering
Czech Technical University in Prague
Karlovo nám. 13, 121 35 Prague 2, Czech Republic
hanzalek@rtime.felk.cvut.cz

Abstract

Loop scheduling is particularly important when designing efficient compilers for parallel architectures. In this article a cyclic schedule of nonpreemptive tasks with precedence constraints and no communication delays on an unlimited number of identical processors will be proposed. In addition an attempt is made to minimize the number of processors used without releasing the time-optimality condition. In order to better understand the nature of the problem the terms data parallelism and structural parallelism are clarified first.

1. Data parallelism

In order to clarify some terminology we will introduce the following hypothesis.
Hypothesis: there is no other parallelism than data parallelism and structural parallelism.

We will talk about data parallelism in the two following cases:
i) if the problem can be modelled in such a way that the resulting PN consists of several identical disconnected subgraphs
ii) if there is a P-invariant or 'degenerated' P-invariant containing more than one token. Degenerated P-invariant in marked graphs corresponds to a path from a source place to a sink place (when adding a dummy transition and connecting this transition to the source place and the sink place we obtain a P-invariant consisting of degenerated P-invariant and the dummy transition).

In our opinion it is very misleading to talk about 'pipe-line parallelism', because pipe-line is not a source of the parallelism, but it is a scheduling strategy. This is why we will talk about pipe-line parallelization.

1.1. SIMD parallelization

We will talk about 'SIMD parallelization' when identical disconnected subgraphs, mentioned in i), are scheduled on separate processors.

1.2. Pipe-line parallelization

For example loops without dependence cycles can be executed in a pipe-line manner, meaning that each instruction is scheduled on a separate processor (this technique was adopted for example in vector computers or inside ALUs).

In a certain level of abstraction we can see the SIMD parallelization and the pipe-line parallelization as two orthogonal approaches, both gaining from data parallelism.

1.3. Token's view of scheduling problem

A generalization of data parallelism leads to a schedule allowing the data to be processed as soon as they are able to do so. The fact that the valid data are represented by the tokens leads to the following reasoning.

Definition 1: *We say that 'token x holds its own processor P_x' iff there is no firing of any transition by any token $y \neq x$ scheduled on processor P_x.*

Theorem 1: *Let a schedule be such that for all markings M of a PN model each token holds its own processor, then the schedule is time-optimal.*

Proof:
If all tokens for all possible markings hold their own processors, then the situation, when any of the tasks is ready to be processed (the token is waiting in front of an enabled transition) and it is not processed, can never appear. Such schedule is time optimal.

Remark: Please notice that this approach is very similar to a dynamic scheduling policy: execute a task as soon as possible. There are two main studies devoted to the dynamic schedule behaviour of cyclic problems. The first, developed by Chrétienne [3], which uses graph theory arguments and longest-path computations and the second, developed by Cohen et al. [1], which uses the (max,+) algebra approach.

2. Structural parallelism

Positive linear invariants are of our interest when analyzing structural net properties [5, 6].

Therefore we can find many algorithms, some of them dating from the last century [8, 9], using different terminologies. So non-negative left annullers of a net's flow matrix are called positive P-invariants, P-semiflows or direction of a positive cone. In a similar way the 'set of minimal support P-invariants'[11] is called 'set of extremal directions of a positive cone' or simply 'generator of P-semiflows'. A recent article [2] by Colom & Silva highlights the connection between convex geometry and Petri Nets and presents two algorithms using heuristics for selecting the columns to annul. Performance evaluation of several algorithms can be found in [16] by Treves.

This paragraph introduces some new notions used to analyze structural properties of cyclic problems. Structural properties are independent on marking, so marked graphs could be replaced by ordinary directed graphs.

Definition 2: *Let* Π *be a square symmetrical matrix called a parallel matrix representing structural parallelism. An element* Π_{ij} *is equal to 1 iff there is no minimal support invariant passing through transition* T_i *and transition* T_j*. Otherwise it is equal to 0.*

An algorithm constructing the parallel matrix Π could be schematically written in the following way:

for i=1...number of transitions
  for j=1...number of transitions
    if (there is no minimal support invariant passing through transitions T_i and T_j)
      then $\Pi_{ij} = 1$
      else $\Pi_{ij} = 0$
    endif
  endfor
endfor

The parallel matrix Π is symmetrical so it can be represented by an undirected graph $G(T, E)$. There is an edge e between vertices T_i and T_j (corresponding to the transitions of the underlying PN model) if and only if the transitions T_i and T_j could be fired concurrently. Please notice that Π represents just a structural parallelism, so it does not hold any information about data parallelism.

A similar approach, called concurrency theory, was suggested by C.A. Petri. Concurrency theory is an axiomatic theory of binary relations of concurrency and causality. For a selection of recent results see [12].

3. Cyclic scheduling

Loop scheduling is particularly important when designing efficient compilers for parallel architectures. Up to now, iterative scheduling problems have been studied from several points of view, depending on the target application. Some theoretical studies have recently been devoted to these problems, in which basic results are often proved independently using different formalism. We hope that this article might contribute to the synthesis of this class of problems.

3.1. Additional terminology

We call a directed graph $G(V, E)$ a tree if its underlying graph is a tree in the undirected sense. We call a vertex v a root of a directed graph G if there are directed paths from v to every other vertex in G. A directed graph is called a directed tree if it is a tree and it contains a root (root has no input edge). We call a vertex with no output edge in a directed tree an endpoint.

There are simple relations between the spanning trees (set of trees covering the graph) and cycles in undirected graphs. To describe these relations we will introduce some terminology. Let $G(V, E)$ be a connected graph and let Tr be a spanning tree of G. An edge of G not lying in Tr is called a chord of Tr. Each chord of Tr determines a cycle in G, called fundamental cycle; namely, the cycle produced by adding the chord to Tr. Any cycle in G can be represented as a linear combination of fundamental cycles.

It is evident that a set of fundamental cycles forms the cycle subspace in a similar way like a basis of P-invariants when the Petri Net under assumption is a marked graph.

In the following paragraphs we will adopt a scheduling policy making use first of the structural parallelism first and then using the data parallelism.

3.2. Structural approach to scheduling

This paragraph introduces a structural approach to scheduling problems, so no tokens are assumed to be in the Petri Net model. The algorithms presented in this paragraph were inspired by P-invariants and the fact that the set of minimal support invariants is unique.

Definition 3: *A timed marked graph is a pair* $< N, \Theta >$ *such that:*
N *is a marked graph*

Θ *is a time associated with transitions*
$\Theta : T \to R^+$

Remark: in the case of marked graphs a representation of the scheduled algorithm by a T-timed marked graph (where a processing time is associated to a transition) is equivalent to a representation by a P-timed marked graph (the processing time is associated to output places). For a detailed discussion see paragraph 2.5.2.6. in [1].

Definition 4: *Let Λ be a schedule matrix of size [number of transitions, number of processors]. An element Λ_{ij} is equal to 1 iff a transition T_i is allocated to processor j and equal to 0 otherwise.*

Rule 1: *Let X be a matrix specifying the set of minimal support P-invariants of a given marked graph. An algorithm $\mathcal{A}_1$ for structural scheduling (no gain of data parallelism is assumed) without communication on an unbounded number of processors could be schematically written in the following way:*

Input: Θ, Pre^T, X
Output: Λ

while there exists a zero line in the matrix Λ
  j:=index of the maximum entry
    in $(\Theta \times Pre^T \times X)$;
  concatenate a column $[Pre^T \times X]_{:j}$ to Λ;
  for i=1..number of places
    if $X_{ij} = 1$
      then zero the line $X_{i:}$
    endif;
  endfor;
endwhile;

Remark 1: Notice that matrix $[Pre^T \times X]$ is a set of minimal support P-invariants expressed in the terms of transitions (each column is composed of transitions present in given P-invariant). Then $(\Theta \times Pre^T \times X)$ is a vector with entries corresponding to the total execution time of a given P-invariant.

Remark 2: The schedule Λ is time-optimal because (time of parallel algorithm execution) = (time of a minimal support P-invariant with the longest execution time).

Example 1: *consider an example of algorithm modeled by the marked graph given in Fig. 1 with Θ=[1 3 1 1 4 2 1]:*

Step 1: $\Theta \times Pre^T \times X = [10\ 8\ 8\ 6]$ therefore P-invariant X_1 is scheduled on processor 1

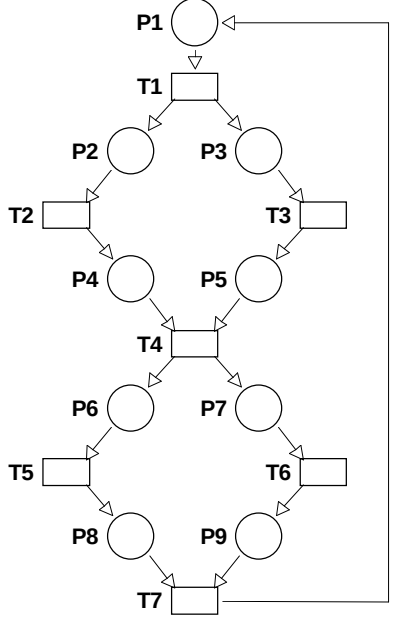

Figure 1: A simple instance for structural scheduling

$$X = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7 \\ P_8 \\ P_9 \end{array} \begin{array}{c} \begin{array}{cccc} X_1 & X_2 & X_3 & X_4 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{array} \qquad \Lambda = \begin{bmatrix} T_1 & 1 \\ T_2 & 1 \\ T_3 & 0 \\ T_4 & 1 \\ T_5 & 1 \\ T_6 & 0 \\ T_7 & 1 \end{bmatrix}$$

Step 2: $\Theta \times Pre^T \times X = [0\ 2\ 1\ 3]$ therefore P-invariant X_4 is scheduled on processor 2

$$X = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7 \\ P_8 \\ P_9 \end{array} \begin{array}{c} \begin{array}{cccc} X_1 & X_2 & X_3 & X_4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{array} \qquad \Lambda = \begin{bmatrix} T_1 & 1 & 0 \\ T_2 & 1 & 0 \\ T_3 & 0 & 1 \\ T_4 & 1 & 0 \\ T_5 & 1 & 0 \\ T_6 & 0 & 1 \\ T_7 & 1 & 0 \end{bmatrix}$$

In the following study we will show that less than $rank(X)$ processors will be needed by the algorithm $\mathcal{A}_1$.

Theorem 2: *Let $G(V, E)$ be a graph consisting of k connected components. Then G contains $m-n+k$ linearly independent cycles, where $m = |E|$ and $n = |V|$.*

Proof:
The proof is given in paragraph 14.1 in [7]. Notice that each of the k components is supposed to be connected (there exists a semipath from each vertex to each vertex) but not strongly connected (there exists a directed path from each vertex to each vertex).

Theorem 3: *Let $G(V,E)$ be a strongly connected directed graph specifying precedence constraints of instance $\mathcal{I}$. Algorithm $\mathcal{A}_1$ allocates p processors to schedule the instance $\mathcal{I}$, where $p \leq m-n+1$.*
Proof:
1) from Theorem 2 $\Rightarrow$ $dim(semicycles\ of\ G) = m-n+1$
2) each cycle is a semicycle
3) $dim(minimal\ support\ invariants\ of\ G) = dim(cycles\ of\ G)$
4) from 1),2) and 3) $\Rightarrow$ $dim(minimal\ support\ invariants\ of\ G) \leq m-n+1$
5) each invariant chosen in the k-th iteration of $\mathcal{A}_1$ is linearly independent of invariants chosen in the $(k-1)$ preceding iterations (this is due to the fact that the invariant chosen in the k-th iteration has a nonzero element in $(\Theta \times X)$, so it contains at least one element of Θ that was not zeroed in the $(k-1)$ preceding iterations) $\Rightarrow$ by mathematical induction we prove: **the chosen invariants are linearly independent**
6) from 4) $\Rightarrow$ $p = (number\ of\ processors) = (number\ of\ chosen\ invariantss) \leq dim(minimalsupportinvariants\ of\ G)$
7) from 4) and 6) $\Rightarrow p \leq m-n+1$

As stated in [15], the matrix X can have a non-polynomial number of columns (corresponding to minimal support invariants), as a consequence the algorithm $\mathcal{A}_1$ is not polynomial. So it will be more interesting to find an algorithm which operates on a positive basis of P-invariants.

Theorem 4: *Let $G(V,E)$ be a strongly connected directed graph. It is possible to find in polynomial time a basis B such that:*
i) B is nonnegative
ii) each cycle of G can be represented as a linear combination of fundamental cycles represented by columns of B $(\sum_{i=1}^{g} \lambda_i b^i)$ where $\lambda_i \in \mathcal{Z}$.

Proof:
1) G is strongly connected $\Rightarrow$ A Depth First Search algorithm (see basic textbooks on graph theory [7, 14]) starting in an arbitrary vertex $r \in V$ finds a directed spanning tree $\mathcal{T}$ covering the graph G
2) the vertex r, called root, has no input edge laying in $\mathcal{T}$
3) there is a directed path from r to every other vertex
4) from 2) and (G is strongly connected) $\Rightarrow$ there is at least one edge $x \in E$ which is an input edge to r and which is a chord (not laying in $\mathcal{T}$)
5) from 3) and 4) $\Rightarrow$ there is a cycle b^k consisting of edge x and the path from r to input vertex of x
6) when reducing (joining) all edges and all vertices of cycle b^k into one vertex r, we obtain a new graph G, which is still strongly connected and a new tree $\mathcal{T}$ which is still a directed spanning tree of G
7) when repeating 2)3)4)5)6) we obtain as many cycles b^k as many there are chords
8) each cycle b^k is linearly independent of cycles $b^1, ..., b^{k-1}$ because it contains a chord x which is not present in cycles $b^1, ..., b^{k-1}$
9) from the tree properties $\Rightarrow$ there are $m-(n-1)$ chords
10) from 8) and 9) $\Rightarrow$ all cycles $b^1, ..., b^{m-n+1}$ are linearly independent, so they are fundamental cycles found in polynomial time = (Depth First Search) + (m-n+1)

Remark: An algorithm finding directed spanning trees of general directed graphs (consisting of more strongly connected components) is given in paragraph 5.2.3. in [14].

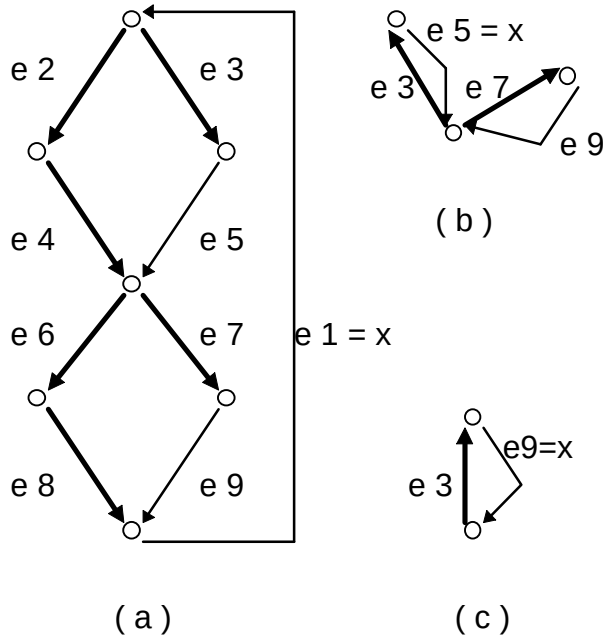

Figure 2: Underlying directed graph for Figure 1 and its reduction

Figure 2 illustrates the proof of Theorem 4 on a given instance reduced in three steps. The proof of the Theorem 4 is in fact an algorithm finding basis B:

$$B = \begin{bmatrix} & b^1 & b^2 & b^3 \\ e_1 & 1 & 1 & 1 \\ e_2 & 1 & 1 & 0 \\ e_3 & 0 & 0 & 1 \\ e_4 & 1 & 1 & 0 \\ e_5 & 0 & 0 & 1 \\ e_6 & 1 & 0 & 1 \\ e_7 & 0 & 1 & 0 \\ e_8 & 1 & 0 & 1 \\ e_9 & 0 & 1 & 0 \end{bmatrix}$$

Remark: Consequence for structural properties of Petri Nets: we can always find a basis consisting of positive P-invariants in fully connected marked graphs. If the marked graph under assumption is not fully connected it is desirable to find fully connected components first.

Theorem 5: *Let an algorithm $\mathcal{A}_2$ be created from the algorithm $\mathcal{A}_1$ in such a way that the set of minimal support invariants X is replaced by a basis B. Then the schedule obtained by algorithm $\mathcal{A}_2$ is time-optimal.*

Proof: Let us assign one token to each P-invariant corresponding to the column of the schedule matrix Λ, then for each marking each token holds at least one processor because the number of tokens inside P-invariant is constant and the schedule matrix Λ covers the whole net. The schedule is time-optimal as a consequence of Theorem 1.

Remark 1: An algorithm $\mathcal{A}_2$ is computed in polynomial time = (time to find B) + maximally (m-n+1) iterations of the algorithm $\mathcal{A}_2$.

Remark 2: The algorithm $\mathcal{A}_2$ does not find a schedule with a minimal number of processors. When scheduling the instance given in Figure 2(a) we obtain a schedule onto three processors:

$$\Lambda = \begin{bmatrix} t_1 & 1 & 0 & 0 \\ t_2 & 1 & 0 & 0 \\ t_3 & 0 & 0 & 1 \\ t_4 & 1 & 0 & 0 \\ t_5 & 1 & 0 & 0 \\ t_6 & 0 & 1 & 0 \\ t_7 & 1 & 0 & 0 \end{bmatrix}$$

It is evident that the number of processors is not minimized, because it is possible to find the time-optimal schedule on two processors as shown in Example 1.

3.3. Quasi-dynamic scheduling

Only structural parallelism was under consideration up to now. In this paragraph we make use of both forms of parallelism - data parallelism and structural parallelism.

The new term 'quasi-dynamic' is used to express the fact that the scheduling policy adopted in this paragraph assigns each task to a set of processors. This scheduling policy is still static because we are able to specify the processor on which the k-th iteration of a given task will be scheduled.

The quasi-dynamic scheduling policy is based on the observation given in Theorem 1 leading to algorithm $\mathcal{A}_3$.

Theorem 6: *Let an algorithm $\mathcal{A}_3$ be created from algorithm $\mathcal{A}_2$ in such a way that a column of the schedule matrix Λ is replicated as many times as there are tokens present in the corresponding positive P-invariant. Then the schedule obtained by algorithm $\mathcal{A}_3$ is time-optimal.*

Proof: similar to the proof of Theorem 5:
1) the schedule matrix Λ covers the whole net
2) the number of tokens inside a P-invariant is constant
3) from 1) and 2) $\Rightarrow$ each token in any marking holds at least one processor (the sufficient condition for Theorem 1 is satisfied)

To be exact the algorithm $\mathcal{A}_3$ could be written in the following way:

Input: Θ, Pre^T, B, M
Output: Λ

$oldB = B$
while there exists a zero line in the matrix Λ
    j := index of the maximum entry
        in $(\Theta \times Pre^T \times B)$;
    q = number of tokens in the P-invariant
        given by $oldB_{:j}$;
    concatenate q-times a column $Pre^T \times B_{:j}$
        to the schedule matrix Λ
    for i=1..number of places
        if $B_{ij} = 1$
            then zero the line $B_{i:}$
        endif;
    endfor;
endwhile;

4. Conclusion

The objective of this article was to bring original ideas to scheduling theory.

Some of the most distinctive features of the article are:

- It specifies the terms 'data parallelism' and 'structural parallelism' in order to understand where the parallelism comes from.
- An attempt is made to see a scheduling problem from the token side.
- Via two cyclic scheduling algorithms, gaining just from the structural parallelism, it leads to an original cyclic scheduling algorithm called 'quasi-dynamic scheduling'. The quasi-dynamic scheduling is time-optimal and achieves the same results like the so-called 'Periodic scheduling' (see [4]) and very similar results like cyclic scheduling based on (max,+) algebra (see [1]), but it uses its proper reasoning. This reasoning allows the extention of quasi-dynamic scheduling to general Petri Nets (not only marked graphs) if a positive P-invariant basis B will be found for general Petri Nets.

Acknowledgements

This research has been conducted at *Trnka Laboratory for Automatic Control* (supported by the Ministry of Education of the Czech Republic under VS97/034).

References

[1] F. Baccelli, G. Cohen, G.J. Olsder, J.P. Quadrat, *Synchronization and Linearity: an algebra for discrete event systems* John Wiley & Sons, (1992).

[2] J.M. Colom, M.Silva, "Convex Geometry and Semiflows in P/T Nets - A Comparative Study of Algorithms for Computation of Minimal P-semiflows", *Advances in Petri Nets 1990*, LNCS 483, Springer Verlag, (1990), 79-112.

[3] P. Chrétienne, "Transient and Limiting behavior of timed event graphs" *RAIRO-TSI*, 4, (1985) 127-192.

[4] P. Chrétienne, E.G. Cofman, J.K. Lenstra, Z. Liu, *Scheduling Theory and its Applications*, John Wiley & Sons, (1995).

[5] J.M.Couvreur, S. Haddad, "Towards a General and Powerfull Computation of Flowsfor Parametrized Coloured Nets", *9th European Workshop on Application and Theory of Petri Nets* Volume 2, Venice, (1988).

[6] J.M.Couvreur, S. Haddad, J.F. Peyre, "Computation of Generative Families of Semi-flows in Two Types of Colored Net", *12th International Conference on Application and Theory of Petri Nets* Aarthus, Denmark, (1991).

[7] J. Demel, *Graphs (in Czech)*, (SNTL Prague, 1989).

[8] J. Farkas, "Theorie der einfachen Ungleichungen", *Journal fur die reine und andgewandte mathematik*, 124 (1902) 1-27.

[9] J.B.J. Fourier, "Solution d'une question particulière du calcul des inégalités". *Oeuvres II*, Gauthier-Villars, Paris (1826!), 317-328.

[10] Z. Hanzalek, Parallel Algorithms for Distributed Control, *PhD. Thesis*, UPS Toulouse and CTU Prague (1997).

[11] F. Kruckeberg, M. Jaxy, "Mathematical Methods for Calculating Invariants in Petri Nets", *in: G. Rozenberg (ed): Advances in Petri Nets*, LNCS 266, Springer (1987) 104-131.

[12] O. Kummer, M.O. Stehr, "Petri's Axioms of Concurrency - A selection of Recent Results", *LNCS 1248*, Springer-Verlag (1997) 195-215.

[13] K. Lautenbach, H.A. Schmid, "Use of Petri Nets for Proving Correctness of Concurrent Process Systems", *IFIP 74*, North Holland Pub. Co., (1974) 187-191.

[14] J.A. McHugh, *Algorithmic Graph Theory*, (Prentice Hall, 1990).

[15] J. Martinez, M. Silva, "A Simple and Fast Algorithm to Obtain All Invariants of a Generalised Petri Net", *in: C. Girault, W. Reisig (eds): Application and Theory of Petri Nets*, Informatik Fachberichte No.52, Springer (1982), 301-310.

[16] N. Treves, "Comprehensive Study of Different Techniques for Semi-flows Computation in Place/Transition Nets", *Advances in Petri Nets 1989*, LNCS 424, Springer Verlag, (1989) 433-452.