A linear programming approach to general dataflow process network verification and dimensioning

Renaud Sirdey Pascal Aubry
CEA, LIST
Embedded Real-Time System Lab
91191 Gif-sur-Yvette Cedex, France
renaud.sirdey@cea.fr p.aubry@cea.fr

In this paper, we present linear programming-based sufficient conditions, some of them polynomial-time, to establish the liveness and memory boundedness of general dataflow process networks. Furthermore, this approach can be used to obtain safe upper bounds on the size of the channel buffers of such a network.

1 Introduction

With the frequency version of Moore's law coming to an end, a new generation of massively multi-core microprocessors is emerging. This has triggered a regain of interest for the so-called dataflow programming models in which one expresses computation-intensive applications as networks of concurrent tasks interacting through (and only through) unidirectional FIFO channels.

Our main result is a linear programming model (Sect. 2) of the states of a *general* Dataflow Process Network (DPN), in the sense of Lee & Parks [1], which allows to obtain a polynomial-time sufficient condition for both liveness (in a sense which is defined in Sect. 3) and memory boundedness (Sect. 4). Furthermore, this approach can be turned into a safe buffer dimensioning method.

2 Modelling system states

2.1 Notations and general assumptions

Let T and F respectively denote the set of tasks and channels.

To each task $t \in T$, we associate a state-transition graph $G_t = (\{v_t^{(0)}\} \cup V_t, \{\tau_t^{(0)}\} \cup A_t)$ (parallel arcs and loops are allowed), where $v_t^{(0)}$ denotes the initial state of task t and where $\tau_t^{(0)}$ denotes the initial transition of that task, this transition being unique and unconditional. Also, given $t \in T$, $P_t \subseteq F$ (respectively $C_t \subseteq F$) denotes the set of channels in which t produces (respectively consumes) data. Note that we have $P_t \cap C_t = \emptyset$. Also note that for each $t, t' \in T^2$, $t \neq t'$, we have $P_t \cap P_{t'} = \emptyset$ as well as $C_t \cap C_{t'} = \emptyset$.

Let $t \in T$, $\tau \in A_t$ and $f \in P_t$ (respectively $f \in C_t$), $\mathsf{qp}_{\tau f}$ (respectively $\mathsf{qc}_{\tau f}$) denotes the amount of data produced (respectively consumed) in channel f by task t when transition τ is executed. An additional constraint is that, $\forall t \in T$, $\forall \tau \in A_t$,

$$\sum_{f \in P_t} \mathsf{qp}_{\tau f} + \sum_{f \in C_t} \mathsf{qp}_{\tau f} > 0, \quad (1)$$

thereby excluding the existence of transitions having no effect on any of the channels.

Given $f \in F$, p_f and c_f respectively denote the tasks which produce and consume data in channel f. Also, d_f denote the capacity of the buffer associated to channel f (depending on the problem at hand d_f can be either given or unknown, as we shall later see).

In the sequel, it is further assumed without loss of generality that the network graph, the directed graph having the tasks as vertices and the channels as arcs, is (simply) connected.

2.2 Variables and linear constraints

For all $\tau \in \bigcup_{t \in T}\{\tau_t^{(0)}\} \cup A_t$, we introduce a variable denoted by $n_\tau \in \mathbb{Z}^+$ which indicates the number of times transition τ has been executed. In order for the n_τ to represent admissible system states, a number of (linear) constraints must be satisfied.

Initialization constraints. Let $t \in T$, initial transition $\tau_t^{(0)}$ must have been executed once and only once, thus, $n_{\tau_t^{(0)}} = 1$.

Conservation constraints. Let $t \in T$ and $v \in V_t$. We then have the following constraint:

$$\sum_{\tau \in \omega^-(v)} n_\tau - 1 \leq \sum_{\tau \in \omega^+(v)} n_\tau \leq \sum_{\tau \in \omega^-(v)} n_\tau.$$

Such a constraint simply reflects the fact that for the n_τ's to represent an admissible system state, vertex v must have been left as many times it has been entered or as many times minus one (in which case it defines the current state of t). If $\sum_{\tau \in \omega^+(v)} n_\tau = \sum_{\tau \in \omega^-(v)} n_\tau - 1$, then the system state described by the n_τ's is such that task t is in state v.

For convenience, let $\gamma_v = \sum_{\tau \in \omega^-(v)} n_\tau - \sum_{\tau \in \omega^+(v)} n_\tau$ meaning that $\gamma_v = 0$ if task t is not in state v and 1 otherwise. Remark that state $v_t^{(0)}$ is duly excluded from the previous sums as this state is by definition left once and never entered.

Unicity constraints. Furthermore, for the system state described by the n_τ's to be admissible, each task must be in one and only one state. That is, for each $t \in T$, if $\sum_{v \in V_t} \gamma_v = 1$. Again, note that $v_t^{(0)}$ is duly excluded from the previous sum.

Consistency constraints. Let $f \in F$, let $\mathsf{qp}_f = \sum_{\tau \in A_{p_f}} n_\tau \mathsf{qp}_{\tau f}$ and $\mathsf{qc}_f = \sum_{\tau \in A_{c_f}} n_\tau \mathsf{qc}_{\tau f}$ respectively denote the amount of data so far produced and consumed on channel f (for convenience). For the n_τ's to describe a valid system state, we must have $\mathsf{qp}_f \geq \mathsf{qc}_f$.

Capacity constraints. Also, for each $f \in F$, we must have

$$\mathsf{qp}_f - \mathsf{qc}_f \leq d_f. \tag{2}$$

3 Modelling undesirable system properties

3.1 Strong and weak blockedness

In a given system state, a task $t \in T$ is *strongly blocked* when it is in a state v in which no outgoing transition can be executed. Consider the following sets of constaints, for each $\tau = (v, v') \in A_t$,

$$\begin{cases} \gamma_v \leq 0, & (3) \\ \mathsf{qp}_f - \mathsf{qc}_f \leq \mathsf{qc}_{\tau f} - 1, & \text{for each } f \in C_t, \quad (4) \\ \mathsf{qc}_f - \mathsf{qp}_f \leq \mathsf{qp}_{\tau f} - d_f - 1, & \text{for each } f \in P_t. \quad (5) \end{cases}$$

Then, strong blockedness means that for each $\tau = (v, v') \in A_t$ either constraint (3) or at least one the constraints of type (4) or at least one of the constraints of type (5) is satistifed for task t.

In a given system state, a task $t \in T$ is *weakly blocked* when it is in a state v in which not all outgoing transitions can be executed. Consider the following sets of constraints, for each $t \in T$ and for each $v \in V_t$,

$$\begin{cases} \gamma_v \leq 0, & (6) \\ \mathsf{qp}_f - \mathsf{qc}_f \leq \mathsf{qc}_{\tau f} - 1, & \text{for each } \tau = (v, v') \in A_t \text{ and for each } f \in C_t, \quad (7) \\ \mathsf{qc}_f - \mathsf{qp}_f \leq \mathsf{qp}_{\tau f} - d_f - 1, & \text{for each } \tau = (v, v') \in A_t \text{ and for each } f \in P_t. \quad (8) \end{cases}$$

Then, weak blockedness signifies that for each $t \in T$ and for each $v \in V_t$ either constraint (6) or at least one the constraints of type (7) or at least one of the constraints of type (8) is satistifed for task t.

The above strong blockedness property is suitable to model non deterministic tasks whereas the weak blockedness property allows modelling deterministic ones (enforcing the fact that, when the task is in a given state, all outgoing transitions are feasible so as to guarantee that the next transition the task *has* to do is possible).

3.2 Sufficient liveness conditions

Although the definitions of the strong and weak blockedness properties involves disjunctive constraints, these can be linearized using standard linear programming modelling techniques (see e.g., [2]).

Thus, given a DPN and a dimensioning $d \in \mathbb{Z}^{|F|}$ we have shown how to formulate an integer linear system of inequalities,

$$\{x \in \mathbb{Z}^n : Ax \leq b(d)\} \quad (9)$$

which *inconsistency*, i.e. $\{x : Ax \leq b(d), x \in \mathbb{Z}^n\} = \emptyset$, is *sufficient* to establish the liveness of the network. A fortiori, the inconsistency of the continuous relaxation of that system, $\{x : Ax \leq b(d), x \in \mathbb{R}^n\} = \emptyset$, provides a polynomial-time (weaker) sufficient condition to establish that property.

4 Memory boundedness of a DPN

4.1 Monotony with respect to dimensioning

As shown by Lee & Parks [1] the DPN formalism is equivalent to the well-known Kahn Process Networks (KPN) formalism. Thus, DPN also exhibit the determinism property exhibited by KPN whereby, for a given input, the data circulating on the channels does not depend on the execution trace. This very convenient property allows, still for a given input, to derive general network properties from properties exhibited by particular traces of execution.

The KPN formalism assumes blocking reads and non blocking writes, which may induce an infinite memory requirement on some channels. However, a KPN K subject to capacity constraints on its channels can straightforwardly be turned into another KPN $K(d)$ ($d \in \mathbb{Z}^{+n}$): all is needed is to emulate a blocking write with a blocking read e.g., to mirror each channel f by an opposite channel f' initially provided with d_f data and to require that each write, respectively read, operation on f be mirrored by a equivalent read, respectively write, operation on f'. Of course, the properties of KPN K with respect to the data circulating on the channels are not preserved by this transformation and, in particular, $K(d)$ may not be deadlock-free despite of the fact that K is.

In essence, the liveness property defined in the previous section ensures that, *for all possible inputs*, an infinite amount of data circulates on *at least one* channel. Assume that a KPN $K(d)$ is live, then a straightforward consequence of the determinism property of KPN is that any KPN $K(d')$ such that $d' \geq d$ (i.e., $\forall f \in F$, $d'_f \geq d_f$) is also live. Indeed, the liveness of $K(d)$ implies that, for any input α, any given trace of execution $\omega(\alpha)$ of $K(d)$ is such that an infinite amount of data circulates on at least one channel and since $\omega(\alpha)$ is also a valid trace of execution of $K(d')$, from the determinism property, all traces of execution of $K(d')$ on α are also such that an infinite amount of data circulates on at least one channel and, thus, $K(d')$ is live. It follows that, given a DPN, if we can find $d \in \mathbb{Z}^{|F|}$ such that either $\{x : Ax \leq b(d), x \in \mathbb{Z}^n\} = \emptyset$ or $\{x : Ax \leq b(d), x \in \mathbb{R}^n\} = \emptyset$, then for any $d' \in \mathbb{Z}^{|F|}$ such that $d' \geq d$, the DPN is live.

4.2 Verifying memory boundedness

Recall integer linear system (9), assume that $d_f = z$ ($\forall f \in F$) and consider the following Integer Linear Program

$$\begin{cases} z_{\mathrm{IP}} = \text{Maximize } z \\ A'y \leq b', \\ y \in \mathbb{Z}^{n+1}, \end{cases} \tag{10}$$

where vector y is the concatenation of vector x and scalar z. This program is straightforwardly derived from system (9) by replacing d_f by z in constraints (2) as well as constraints (5) or (8) (depending on which of the two applies) and by moving z to the LHS.

In essence, from the monotony property derived in Sect. 4.1, any solution to the above program provides *the largest value of z such that the network is not live* and, thus, for any dimensionning $d \in \mathbb{Z}^{|F|}$ such that $\forall f \in F$, $d_f \geq z_{\mathrm{IP}}$ the network is live. Three cases can then occurs. Case 1: the ILP has no solution, a degenerate case which can occur only for networks with no channels (since, due to Eq. (1), all transitions are effective). This latter case in hereafter ignored. Case 2: $z_{\mathrm{IP}} < \infty$, which is sufficient to establish that the network is live and memory bounded. Case 3: $z_{\mathrm{IP}} = \infty$ in which case we cannot conclude with respect to both liveness and memory boundedness. Furthermore, when $z_{\mathrm{IP}} < \infty$, letting $d_f = z_{\mathrm{IP}} + 1$ ($\forall f \in F$) gives a (presumably small) channel buffer dimensioning which guarantees liveness.

Again, it is possible to consider the continuous relaxation z_{LP} of program (10). In particular, when one wishes only to determine wether or not $z_{\mathrm{IP}} < \infty$ then it is necessary and sufficient to determine whether or not $z_{\mathrm{LP}} < \infty$ as $z_{\mathrm{IP}} < \infty \Leftrightarrow z_{\mathrm{LP}} < \infty$. Indeed, a well known fact in the theory of linear and integer programming [2] states that if the integer hull P_I of a rational polyhedron P (i.e., the convex hull of the integral vectors in P) is nonempty then $\max\{cx : x \in P\}$ is bounded *if and only if* $\max\{cx : x \in P_I\}$ is bounded and, provided that $0 \in \{y : A'y \leq b', y \in \mathbb{Z}^{n+1}\}$, this result applies to program (10) and its relaxation. It follows that $z_{\mathrm{LP}} < \infty$ provides a polynomial-time sufficient condition to establish both liveness and boundedness of a DPN which is equivalent to $z_{\mathrm{IP}} < \infty$.

5 Remarks on algorithmic aspects

Although z_{LP} can be computed in polynomial-time, it can be expected, when $z_{\mathrm{LP}} < \infty$, that the integrality gap, $z_{\mathrm{LP}} - z_{\mathrm{IP}}$, is often quite large. Thus, should one wishes to obtain a tight (if not the tightest) upper bound on the channel buffer dimensioning, there is a practically relevant need to either solve program (10) or at least to decrease the aforementioned integrality gap.

Regarding the resolution of program (10), it should be emphasized that the polyhedron $P_I = \{y : A'y \leq b', y \in \mathbb{Z}^{n+1}\}$ (recall that the integer hull of a polyhedron is also a polyhedron) is generally not a polytope (since the n_τ are not necessarily bounded). Therefore, in the general case, procedures which enumerate integer points inside the polyhedron, as those used in most off-the-shelf integer linear programming solvers, are doomed not to terminate.

Thus, in order to guarantee termination in the present context, an (exterior) cutting plane approach must be used. As an example, Gomory's cutting plane algorithm is guaranteed to terminate (though generally after a prohibitively long time). A more practically promising approach, consists in using specially tailored classes of inequalities derived from a polyhedral study of the geometric structure of P_I so as to derive a custom cutting plane algorithm.

References

[1] E. A. Lee & T. M. Parks (1995): *Dataflow process networks*, Proceedings of the IEEE 83(5), pp. 773-779.

[2] G. L. Nemhauser and L. A. Wolsey (1999): *Integer and combinatorial optimization*, *Wiley*.