ISCAS-SKLCS-14-12 **August, 2014**

中 国 科 学 院 软 件 研 究 所
计 算 机 科 学 国 家 重 点 实 验 室
技 术 报 告

Static Optimal Scheduling and Mapping of Synchronous Dataflow Graphs on a Heterogeneous Multiprocessor Platform with Model Checking

by

Xue-Yang Zhu, Rongjie Yan, Yu-Lei Gu
Guangquan Zhang

State key Laboratory of Computer Science
Institute of Software
Chinese Academy of Sciences
Beijing 100190. China

Static Optimal Scheduling and Mapping of Synchronous Dataflow Graphs on a Heterogeneous Multiprocessor Platform with Model Checking

Xue-Yang Zhu and Rongjie Yan
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing, China
{zxy,yrj,zj}@ios.ac.cn

Yu-Lei Gu and Guangquan Zhang
School of Computer Science and Technology
Soochow University
Suzhou, China
{guyl@ios.ac.cn,gqzhang@suda.edu.cn}

***Abstract*—Synchronous dataflow graphs (SDFGs) are widely used to model digital signal processing (DSP) and streaming media applications. Such applications are usually operated on embedded systems, which require high performance and low energy consumption. In this paper, we present exact methods for static optimal scheduling and mapping of SDFGs on a heterogenous multiprocessor platform. The optimization criteria we consider are throughput and energy consumption. By defining a (priced) timed automata (TA) semantics of system models, which includes an SDFG and a multiprocessor platform, we transform a system model to a (priced) TA network for real-time model checking tool UPPAAL (CORA); meanwhile, the optimization goals are formalized as logical formulas of the tools. Thanks to the exhaustive exploration nature of model checking and the facility of the tools, we can get throughput-optimal schedules that have a best energy consumption, and energy consumption-optimal schedules that have a best throughput. Although for model checking, state explosion is inevitable when the models scale up, our experiments, carried out on an MPEG-4 decoder algorithm and a computation example with various parameters, show that our methods can deal with relatively large scale models within reasonable execution times. Our method can also reveal impacts of different parameters on optimization goals.**

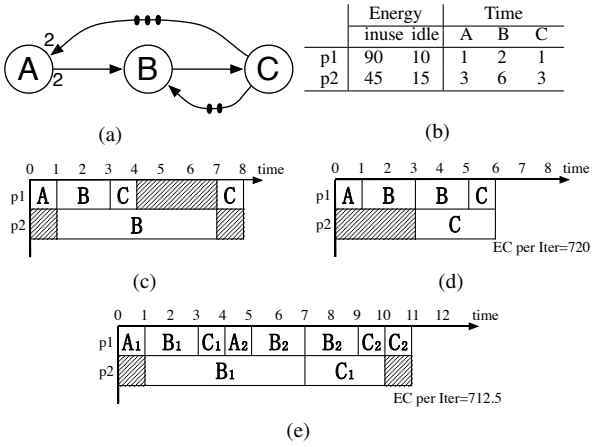


Fig. 1. The system model $\mathcal{M}_1$ and its schedules. (a) The SDFG G_1; (b) the execution platform P_1 and the execution times of actors G_1 on different processors; (c) an ASAP periodic schedule of G_1 with IP=8; (d) a periodic schedule of G_1 with IP=6; (e) an unfolding schedule of G_1 with IP=$\frac{11}{2}$. The sample rates in the SDFG are omitted when they are 1; black dots represent initial tokens on the edges.

I. Introduction

Synchronous dataflow graphs (SDFGs) [1] are widely used to represent DSP and streaming media applications, which are usually operated under real-time and resource constraints. Each node (also called actor) in an SDFG represents a computation and each edge models a FIFO channel; the sample rates of actors may differ. An example SDFG, G_1, is shown in Fig. 1(a). There are many practical applications modeled with SDFGs, such as a spectrum analyzer [2], a satellite receiver [3] and an MPEG-4 decoder[4]. In this paper, we are concerned with constructing efficient static (compile-time) schedules of SDFGs on a heterogeneous multiprocessor platform.

A *static schedule* arranges the actors of an SDFG to be executed repeatedly, also called a *periodic schedule*. Execution of all the actors for the required number of times is referred to as an *iteration*. An iteration of an SDFG may include more than one execution, also called *firing*, of an actor. Different actors may fires a different number. Actor B in G_1, for example, fires twice in an iteration, while A fires once. The average computation time per iteration is called *iteration period* (IP). The IP is the reciprocal of the *throughput*. We use IP and throughput alternatively in the remainder of the paper.

Static schedules of SDFGs can be non-overlapped or overlapped. If the scheduled computations in any two consecutive iterations do not overlap, it is a non-overlapped schedule; otherwise, it is an overlapped schedule. The timing models can be integral or fractional. In an integral model the scheduled computation can only start at integral number while in a fractional model it can start at any rational number. We focus on non-overlapped scheduling for the integral timing model in this paper.

For homogeneous multiprocessor scheduling of SDFGs, an *as soon as possible* (ASAP) execution can be used to find the schedules with minimal IP (throughput-optimal). An ASAP method on SDFGs is similar to the critical path methods on its equivalent *Homogenous synchronous dataflow graphs* (HSDFGs), a special type of SDFGs, of which all sample rates of actors are one.

For heterogeneous multiprocessor scheduling, however, an

ASAP schedule is not necessarily throughput-optimal. The ASAP schedule shown in Fig. 1(c), for example, arranges executions of actors of G_1 on a platform including two heterogeneous processors as shown in Fig. 1(b). It has an IP larger than the IP of another schedule shown in Fig. 1(d), which is not ASAP. The latter is actually throughput-optimal when we consider one iteration as a cycle. It can be further optimized by unfolding scheduling.

Scheduling f iterations as one *cycle* may lead to more options for parallel execution if $f > 1$ and therefore may reduce the IP and the energy consumption of a schedule. This is *unfolding* scheduling and f is called *unfolding factor*. Unfolding is an important optimization technique for data flow models. It has been widely studied [5], [6]. See Fig. 1(e) for example. The IP of a periodic schedule of G_1 with unfolding factor 2 is $\frac{11}{2}$, smaller than that of the schedule shown in Fig. 1(d). The improvement of energy consumption per iteration (IEC) is similar.

In this paper, we present a method to schedule and map the firings of actors of an SDFG on a heterogeneous multiprocessor platform for a given unfolding factor. The schedules are either throughput-optimal with a best energy consumption or energy consumption-optimal with a best throughput.

By defining a timed automata (TA) [7] semantics of SDFGs, we transform an SDFG to a TA network for real-time model checking tool UPPAAL [8]. Formalizing the ending of an unfolding periodic schedule as a temporal logical formula, we obtain a throughput-optimal schedule by the tool. The energy consumption is then computed according to the schedule. Adding energy consumption as a constraint, we can find a fastest schedule under the constraint. By restricting the constraint we get a throughput-optimal schedule with a best energy consumption.

Similarly, by defining a priced timed automata (PTA) [9] semantics of SDFGs, we transform an SDFG to a PTA network of tool UPPAAL CORA. An energy consumption-optimal schedule is found by the tool. However, the throughput of the schedule may be not the best under the optimal energy consumption. Using UPPAAL on the SDFG again with the optimal energy consumption as a constraint, we get an energy consumption-optimal schedule with a best throughput.

Other kinds of constraints, e.g. auto-concurrency constraints and buffer size constraints, are also considered and integrated into our method framework. To the best of our knowledge, this is the first exact method to schedule SDFGs on heterogenous multiprocessor platforms according to optimization criteria combining throughput and energy consumption, and meanwhile the scheduling takes into account the combination of various constraints.

To evaluate our method, we have implemented it in the tool iDFOS and carried out experiments on an MPEG-4 decoder algorithm and a computation example with various parameters. We show the outputs of our method and the execution time and memory used of the procedure for throughput-optimal schedules. Although for model checking, state explosion is inevitable when the models scale up, our experimental results show that our methods can deal with relatively large scale models within reasonable execution times.

The remainder of this paper is organized as follows. The input models and the problems addressed are formulated in Section II. Our main contributions are illustrated in Sections III, IV and V. Section VI provides an experimental evaluation. Related work is introduced in Section VII. Section VIII concludes.

II. Model Description and Problem Fomulation

An *execution platform* P is a set of heterogeneous processors. A computation will require different amounts of running time if it is executed on different processors. The energy consumption per unit time for each processor p is defined by $uEC(p)$, indicating the energy consumption when p is used for some tasks, and $iEC(p)$, indicating the energy consumption per unit time when p is idle.

A *synchronous dataflow graph* (SDFG) is a finite directed graph $G = \langle V, E \rangle$. V is the set of actors, modeling the functional elements of the system; E is the set of directed edges, modeling interconnections between functional elements. Each edge e is weighted with three properties, $d(e)$, $prd(e)$ and $cns(e)$. Property $d(e)$ is the number of initial tokens on e, $prd(e)$ is the number of tokens produced onto e by each firing of the source of e, and $cns(e)$ is the number of tokens consumed from e by each firing of the sink actor of e. These numbers are also called the *delay*, *production rate* and *consumption rate*, respectively.

The source actor and sink actor of e are denoted by $src(e)$ and $snk(e)$, respectively. The set of incoming edges to actor α is denoted by $InE(\alpha)$, and the set of outgoing edges from α by $OutE(\alpha)$. If $prd(e) = cns(e) = 1$ for each $e \in E$, G is a *homogeneous SDFG* (HSDFG).

An actor may require different amounts of execution time if running on different processors. If execution platform P is considered, each actor α is weighted with computation times $t(\alpha, p)$, for all $p \in P$. Normally, $t(\alpha, p)$ is a nonnegative integer. For technical reason, we allow $t(\alpha, p)$ to be 0 or -1. The former is used to model the execution time of some dummy actors; the latter is used when α is not allowed to run on p.

An SDFG G is *sample rate consistent* [1] if and only if there exists a positive integer vector $q(V)$ satisfying *balance equations*, $q(src(e)) \times prd(e) = q(snk(e)) \times cns(e)$ for all $e \in E$. The smallest q is called the *repetition vector*. We use q to represent the repetition vector directly. For example, a balance equation can be constructed for each edge of G_1 in Fig. 1 (a). By solving these equations, we have G_1's repetition vector $q = [1, 2, 2]$. Only sample rate consistent and deadlock-free SDFGs are meaningful in practice. We consider only such SDFGs, which can be verified efficiently [1].

An *iteration* is a firing sequence in which each actor α occurs exactly $q(\alpha)$ times. A consistent SDFG can always be translated to an equivalent HSDFG [10]. Each actor α in the SDFG has $q(\alpha)$ copies in its equivalent HSDFG. The HSDFG of G_1 is shown in Fig. 2.

Definition 1 (System model)**.** A *system model* includes an SDFG G and its execution platform P, denoted by $\mathcal{M} = (G, P)$.

A *static schedule* arranges computations of an algorithm to be executed repeatedly. An *unfolding schedule* of system model $\mathcal{M} = (G, P)$ is a static schedule arranging f consecutive

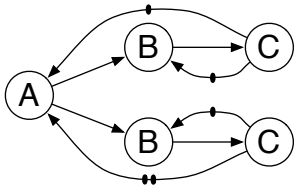


Fig. 2. The equivalent HSDFG of G_1.

iterations of G running on P. The number f is called *unfolding factor* and the f iterations form a *cycle*. An unfolding schedule with unfolding factor f is an f*-schedule*.

Definition 2 (f-schedule)**.** An f*-schedule* of system model $\mathcal{M} = (G, P)$ is a function $S : V \times \mathbb{N} \to \mathbb{N} \times P$, where $\mathbb{N}$ is the set of non-negative integers, defining the time arrangement and the processor allocation of firings of actors in G. Schedule S with a *cycle period* (CP) T is defined as follows. For the i^{th} firing of actor α, denoted by (α, i), $i \in [1, \infty)$:

1) $S(\alpha, i).st$ is the start time of (α, i), when there are sufficient tokens on each $e \in InE(\alpha)$ for a firing of α;
2) $S(\alpha, i).pa$ is the processor assigned to (α, i), which is available at the moment $S(\alpha, i).st$;
3) $S(\alpha, i + f \cdot q(\alpha)).st = S(\alpha, i).st + T$;
4) $S(\alpha, i + f \cdot q(\alpha)).pa = S(\alpha, i).pa$

Such a schedule can be represented by the first f iterations and period T. It is the part of the schedule defined by $S(\alpha, i)$ with $1 \le i \le f \cdot q(\alpha)$ for all α. From now on, we only consider the finite part of f-schedules.

The *iteration period* (IP) of S is the average computation time of an iteration, that is, $IP = \frac{CP}{f} = \frac{T}{f}$.

The energy consumption of f-schedule S can be computed as follows. For conciseness, we omit parameters S and f when it is clear in context. Denote the set of all firings assigned on processor p by $AonP(p)$.

$$AonP(p) \equiv_{def} \{(\alpha, i) | S(\alpha, i).pa = p \wedge i \in [1, f \cdot q(\alpha)] \wedge \alpha \in V\}.$$

The total time p occupied in S is

$$occT(p) = \sum_{(\alpha,i) \in AonP(p)} t((\alpha, i), p), \quad (1)$$

where $t((\alpha, i), p) = t(\alpha, p)$.

Then the energy consumption of S is

$$EC = \sum_{p \in P} occT(p) \cdot uEC(p) + [CP(S) - occT(p)] \cdot iEC(p). \quad (2)$$

The *iteration energy consumption* (IEC) of S is the average energy consumption of an iteration, that is, $IEC = \frac{EC}{f}$.

Given system model $\mathcal{M} = (G, P)$ and unfolding factor f, suppose the set of all f-schedules of $\mathcal{M}$ is $\mathbf{S}$, the problems we address are:

1) how to find a f-schedule S_{optP} such that

$$IP(S_{optP}) = \min\{IP(S) | S \in \mathbf{S}\}, \text{ and}$$
$$IEC(S_{optP}) = \min\{IEC(S) | S \in \mathbf{S} \wedge IP(S) = IP(S_{optIP})\}$$

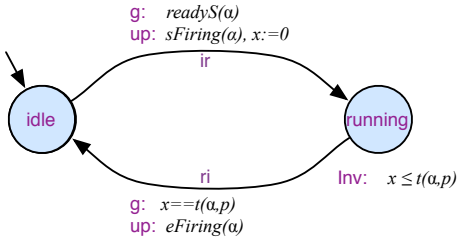


Fig. 3. The timed automaton ta_p.

2) how to find a f-schedule S_{optE} such that

$$IEC(S_{optE}) = \min\{IEC(S) | S \in \mathbf{S}\}, \text{ and}$$
$$IP(S_{optE}) = \min\{IP(S) | S \in \mathbf{S} \wedge IEC(S) = IEC(S_{optE})\}$$

III. A Timed Automata Semantics of System Models

A. Introduction to Timed Automata

In this section we recap the concepts of syntax and semantics of timed automata (TA) [7] and its extension with cost [9]. Let X be a set of clocks, $\mathcal{V}$ be a set of bounded integer variables. An integer expression over $\mathcal{V}$ is of the form $v, k, \varepsilon_1 + \varepsilon_2$ or $\varepsilon_1 - \varepsilon_2$, where $v \in \mathcal{V}$, k is an integer constant, and ε_1 and ε_2 are integer expressions. We use $C(X, \mathcal{V})$ to denote the set of constraints in the form of $\phi ::= x \sim k \,|\varepsilon_1 \sim \varepsilon_2 \,|\phi_1 \wedge \phi_2$, where $x \in X$, and $\sim \in \{<, \le, =, \ge, >\}$. Let $U(X, \mathcal{V})$ be the set of updates in the form of $\eta ::= x := 0 \,|v := \varepsilon \,|\eta_1; \eta_2 \,|fun$, where $x \in X$, $v \in \mathcal{V}$, ε is an integer expression, η_1 and η_2 are updates, and *fun* is a function with a set of updates.

A timed automaton is a tuple $(L, X, \mathcal{V}, \mathcal{E}, Inv, l_0)$, where L is a set of locations, $\mathcal{E} \subseteq L \times C(X, \mathcal{V}) \times U(X, \mathcal{V}) \times L$ is a set of edges, $Inv : L \to C(X, \mathcal{V})$ assigns invariants to locations, and l_0 is the initial location. A network of n timed automata (NTA) is a tuple of timed automata $A_1 \| \cdots \| A_n$ over X, $\mathcal{V}$.

A clock valuation γ for a set X is a mapping from X to $\mathbb{R}^+$, where $\mathbb{R}^+$ is the set of non-negative real numbers. A variable valuation u is a function from $\mathcal{V}$ to $\mathbb{Z}$, where $\mathbb{Z}$ is the set of integers. A pair of valuation (γ, u) satisfies a constraint ϕ over X and $\mathcal{V}$, denoted by $(\gamma, u) \models \phi$, if and only if ϕ evaluates to *true* with the values γ and u. Let $\gamma_0(x) = 0$ for all $x \in X$. For $\delta \in \mathbb{R}^+$, $\gamma + \delta$ denotes the clock valuation that maps every clock x to the value $\gamma(x) + \delta$. For an update $\eta(Y, \mathcal{V}')$ over a pair of valuation (γ, u), where $Y \subseteq X$ and $\mathcal{V}' \subseteq \mathcal{V}$, $(\gamma, u)[\eta(Y, \mathcal{V})]$ denotes the clock valuation that maps all clocks in Y to zero and agrees with γ for all clocks in $X \setminus Y$, and the variable valuation that maps all integer variables in $\mathcal{V} \setminus \mathcal{V}'$ agree with u.

For example, Fig. 3 shows a time automaton for processor p. There are two locations to mark idle or running status. When the guard $readyS(\alpha)$ is satisfied, the transition from location *idle* to *running* is enabled. Once the transition is triggered, updates on clock $x := 0$ and other integer variables in $sFiring(\alpha)$ are executed. The invariant $x \le t(\alpha, p)$ of location *running* restricts the allowed maximal delay.

The semantics of timed automata is defined as follows.

Definition 3 (Semantics of timed automata)**.** The semantics of a timed automaton $A = (L, X, \mathcal{V}, \mathcal{E}, Inv, l_0)$ is a timed transition system $\mathcal{T} = \langle \mathcal{S}, s_0, \to \rangle$ where $\mathcal{S} \subseteq L \times \mathbb{R}^+ \times \mathbb{Z}$ is the set of

states, $s_0 = (l_0, \gamma_0, u_0)$ is the initial state and $\rightarrow$ is the transition relation such that

- delay transition. $(l, \gamma, u) \xrightarrow{\delta} (l, \gamma + \delta, u)$ if $\forall \delta' : 0 \leq \delta' \leq \delta \Rightarrow (\gamma + \delta', u) \models Inv(l)$ where $\delta \in \mathbb{R}^+$, and
- discrete transition. $(l, \gamma, u) \rightarrow (l', \gamma', u')$ if there exists $e = (l, g, \eta(Y, \mathcal{V}'), l') \in \mathcal{E}$ such that $(\gamma, u) \models g$, $(\gamma', u') = (\gamma, u)[\eta(Y, \mathcal{V}')]$, and $(\gamma', u') \models Inv(l')$.

The trace of a timed automaton is a finite or infinite sequence $(l_0, \gamma_0, u_0) \rightarrow (l_1, \gamma_1, u_1) \rightarrow \ldots$, where $\rightarrow$ is either a delay transition or a discrete transition. For a network of timed automata, the discrete transitions are executed interleavingly.

Priced timed automata [9] is an extension of timed automata to allow the accumulation of costs during behaviour. The extension from timed automata is $A_c = (L, X, \mathcal{V}, \mathcal{E}, Inv, l_0, \mathcal{P})$, where $\mathcal{P} : L \cup \mathcal{E} \rightarrow \mathbb{N}$ assigns cost rates and costs to locations and edges, respectively. The semantics of priced timed automata is similar to the version without price, except that the cost in a delay transition is in direct proportion to the time elapsed, and the cost in a discrete transition is the cost of the edge.

For a network of timed automata, we use vectors of locations and the cost rate of a vector of locations is the sum of cost rates in the locations of the vector. For a finite trace of a priced timed automaton, the cost is the sum of the cost for all discrete and delay transitions.

B. A TA Semantics of System Models

Before introducing the TA semantics of system models, we first formalize the behavior of SDFGs and introduce some notations to simplify the later illustrations.

The behavior of an SDFG consists of a sequence of *firings*. We use updates $sFiring(\alpha)$ and $eFiring(\alpha)$ to encode the start and the end of a firing of α, and use $readyS(\alpha)$ to describe the enabling condition of $sFiring(\alpha)$. Additionally, we introduce sets of variables $tn(E)$ and $numF(V)$, to record the current number of tokens on edges in E and the firing times of actors in V, respectively. Testing and updating the value of $numF(V)$ are not really a part of the behavior of SDFGs, which are used to facilitate the construction of a f-schedule.

The guard $readyS(\alpha)$ tests if there are sufficient tokens on the incoming edges of actor α to enable a firing. If the firing number of α reaches $f \cdot q(\alpha)$, no new firing of α is allowed.

$$readyS(\alpha) \equiv_{def} \quad \forall e \in InE(\alpha) : tn(e) \geq cns(e) \\ \wedge numF(\alpha) < f \cdot q(\alpha).$$

When a firing of α starts, it reduces the number of tokens of its incoming edges according to the consumption rates.

$$sFiring(\alpha) \equiv_{def} \quad \forall e \in InE(\alpha) : tn'(e) = tn(e) - cns(e) \\ \wedge numF'(\alpha) = numF(\alpha) + 1,$$

where $tn'(e)$ refers to the value of $tn(e)$ in the new state, and the same for $numF'(\alpha)$. For conciseness, we omit the elements of states if their values remain unchanged.

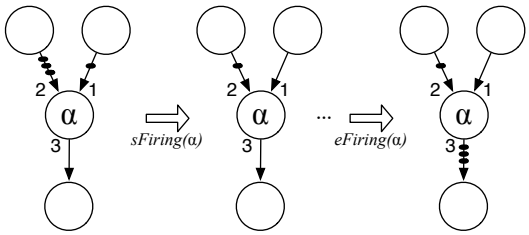


Fig. 4. The effect of *sFiring* and *eFiring*.

If a firing of α runs on processor p, it will finish after $t(\alpha, p)$ units of time. And update $eFiring(\alpha)$ increases tokens of α 's outgoing edges according to the production rates.

$$eFiring(\alpha) \equiv_{def} \quad \forall e \in OutE(\alpha) : tn'(e) = tn(e) + prd(e)$$

The effects of *sFiring* and *eFiring* are demonstrated in Fig. 4.

For a given system model $\mathcal{M} = (G, P)$, a processor is either idle or running and an actor is enabled or not. When an actor is enabled, it may start a firing. While it is firing, it is firing on some processor p and p enters a running state.

Then the behavior of α running on p can be modeled in a timed automaton $ta_p(\alpha)$; and the behavior of processor p can be modeled by $ta_p(\alpha)$ with non-deterministically selecting actor α from V.

Definition 4 (TA of the behavior of processors)**.** A TA of the behavior of processor p is $ta_p = \exists \alpha \in V : ta_p(\alpha)$, and $ta_p(\alpha) = (L, X, \mathcal{V}, \mathcal{E}, Inv, l_0)$, where $L = \{idle, running\}$, $X = \{x\}$, $\mathcal{V} = \{tn(E), numF(V)\}$, $l_0 = idle$, $Inv = \{running : x \leq t(a, p)\}$, and $\mathcal{E} = \{ir, ri\}$, where $ir = (idle, readyS, sFiring(\alpha); x := 0, running)$, and $ri = (running, x == t(\alpha, p), eFiring(\alpha), idle)$.

The locations of ta_p indicate the status of processor p. That is, $ta_p.idle$ means p is idle and therefore is available for a firing of actors to run, and $ta_p.running$ means p is occupied by some firing. The graphical representation of ta_p is shown in Fig. 3.

Actors of SDFG G can fire in parallel only if they are ready and there are available processors. The behavior of processors also implicates the behavior of the SDFG.

Subsequently, the behavior of system model $\mathcal{M}$ can be described in a network of timed automata $nta_{\mathcal{M}}$, which has $|P|$ concurrent processes and a global clock, where $|P|$ is the number of processors in P. The global clock is used to measure the whole execution time of the system.

Definition 5 (NTA of the behavior of system models)**.** The behavior of system model $\mathcal{M} = (G, P)$ is a NTA $nta_{\mathcal{M}} = \|_{p \in P} ta_p$ with a global clock $glbClk$.

The above-mentioned semantics are the standard timed automata description, which can be translated into the input of UPPAAL straightforwardly. Quantification $\exists \alpha \in V$ can be implemented by the 'Selections' feature of UPPAAL.

Above defined ta_p and $nta_{\mathcal{M}}$ implicatively include f as a parameter. We omit it in the notations for conciseness.

IV. Static Optimal Scheduling and Mapping

A. Traces and Schedules

A f-schedule of $\mathcal{M}$ can be found by a trace of $nta_{\mathcal{M}}$.

Guard $numF(\alpha) < f \cdot q(\alpha)$ in $readyS(\alpha)$ forces $nta_{\mathcal{M}}$ being deadlocked after the firings of f-iterations of G are finished. Therefore a trace of $nta_{\mathcal{M}}$ is infinite with finite discrete transitions, i.e., $p.sf(\alpha)$ and $p.ef(\alpha)$, where $p.sf(\alpha)$ is the transition caused by update $sFiring(\alpha)$ of edge ir of ta_p, and $p.ef(\alpha)$ is caused by $eFiring(\alpha)$ of edge ri. Hence we consider only the finite part of a trace that includes all finite discrete transitions. Denote the set of transitions of trace σ as $\mathcal{E}_\sigma$ and the state caused by $p.sf(\alpha)$ as $s_{p-\alpha}$.

Theorem 6. In a trace σ of $nta_{\mathcal{M}}$, for each actor α:

1) $\nexists s_{p-\alpha}$ such that $s_{p-\alpha}.numF(\alpha) > f \cdot q(\alpha)$;
2) $\forall i \in [1, f \cdot q(\alpha)]$, there is a unique $s_{p-\alpha}$ such that $s_{p-\alpha}.numF(\alpha) = i$;
3) when $p.sf(\alpha)$ occurs, there are sufficient tokens on each $e \in InE(\alpha)$ for one firing of α and processor p is available.

Proof: 1) is guaranteed by $readyS(\alpha)$; 2) is guaranteed by $sFiring(\alpha)$; according to the definition of ta_p, only when $ta_p.idle$ and $readyS(\alpha)$ are satisfied, p may select α to fire and therefore 3) is guaranteed. ■

Algorithm 1 presents the procedure of finding a f-schedule from a trace. Its correctness is ensured by Theorem 6.

Algorithm 1 Sch($\mathcal{M}$,σ)

Require: A trace σ of $nta_{\mathcal{M}}$
Ensure: An f-schedule of $\mathcal{M}$, S

```
for all e ∈ E_σ do
    if ∃α ∈ V : e == p.sf(α) then
        S(α, s_{p−α}.numF(α)).st = s_{p−α}.glbClk
        S(α, s_{p−α}.numF(α)).pa = p
    end if
end for
return S
```

See system model $\mathcal{M}_1$ shown in Fig. 1 for example. The schedule in Fig. 1(c) is a 1-schedule. It can be found in a trace of $nta_{\mathcal{M}_1}$, part of which is shown in Fig. 5.

B. Throughput-Optimal Solution

We denote the f-schedule derived by trace σ as S_σ. The cycle period of S_σ is the time when the last firing terminates, that is:

$$CP(S_\sigma) = \max\{s_{p-\alpha}.glbClk + t(\alpha, p) | s_{p-\alpha} \in \mathcal{S}_\sigma\}.$$

Suppose the set of traces of $nta_{\mathcal{M}}$ is Σ, the optimal IP of f-schedules of $\mathcal{M}$ is

$$optIP(\mathcal{M}) = \min\left\{\frac{CP(S_\sigma)}{f} \middle| \sigma \in \Sigma\right\}$$

A *throughput-optimal* f-schedule is a schedule with its IP equal to $optIP$.

For a given model $\mathcal{M}$ and an unfolding factor f, $nta_{\mathcal{M}}$ will be deadlocked after the firings of f-iterations of G terminate.

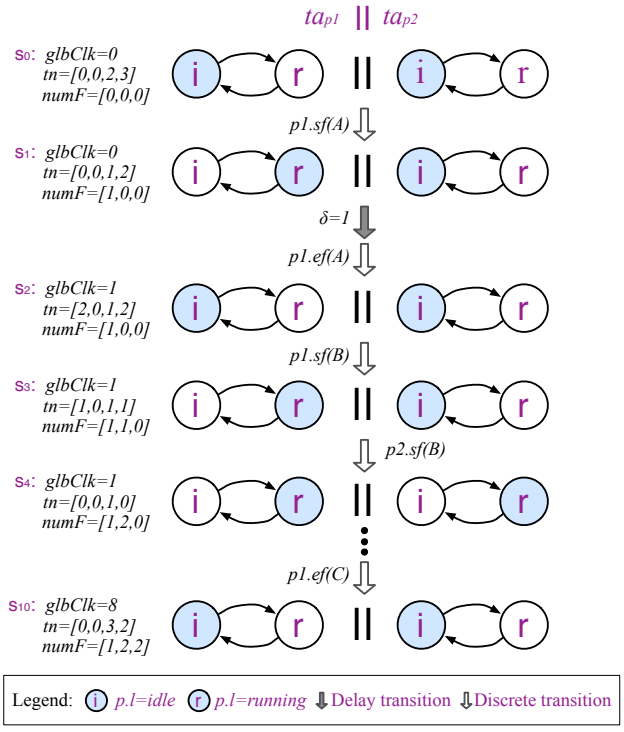


Fig. 5. A part of a trace of system model $\mathcal{M}_1$ shown in Fig. 1, where cycles in blue show the current location.

This property can be formalized by a CTL (Computation Tree Logic) formula **EF** *deadlock*. CTL formula **EF**ϕ is true when ϕ is eventually true at some states of some traces of $nta_{\mathcal{M}}$, denoted by $nta_{\mathcal{M}} \models$ **EF**ϕ.

In the following discussion, we always apply UPPAAL to return a fastest trace, implemented by function $trace(nta_{\mathcal{M}}, \phi)$, where $\mathcal{M}$ is the given model and ϕ is the property.

From the trace returned by $trace(nta_{\mathcal{M}},$ **EF** $deadlock)$, we obtain a throughput-optimal f-schedule of $\mathcal{M}$, denoted by S_{optIP}, i.e.,

$$S_{optIP} = Sch(\mathcal{M}, trace(nta_{\mathcal{M}}, \textbf{EF}\ deadlock)).$$

The energy consumption of the schedule, $EC(S_{optIP})$, can be computed according to Eqn. (2).

To find an f-schedule with $optIP$ and a best energy consumption, we need to add a constraint on energy consumption. Therefore, we add an update $occT(p) = occT(p) + t(\alpha, p)$ to edge ri in ta_p, and the subsequent model is $nta_{\mathcal{M}'}$. Then according to Eqn. (2), the property that the energy consumption at time $glbClk$ is no more than a given ec is defined as

$$con(ec) \equiv_{def} glbClk \leq \frac{ec - \sum_{p\in P} occT(p) \cdot [uEC(p) - iEC(p)]}{\sum_{p\in P} iEC(p)}$$

With $con(ec)$ as the additional constraint, we decrease ec gradually to check whether we can reach a smaller energy consumption with $optIP$. The details on computing an f-schedule with $optIP$ and a best energy consumption are explained in Algorithm 2.

Algorithm 2 optPSch($\mathcal{M}$)

Require: $\mathcal{M}$
Ensure: An f-schedule S_{optP} of $\mathcal{M}$
1: $S_{optIP} = Sch(\mathcal{M}, trace(nta_{\mathcal{M}}, \mathbf{EF}\ deadlock))$
2: $ec = EC(S_{optIP})$
3: $S_{optP} = S_{optIP}$
4: **repeat**
5: $\phi = \mathbf{EF}\ deadlock \wedge con(ec - 1)$
6: $S_{IP} = Sch(\mathcal{M}, trace(nta_{\mathcal{M}'}, \phi))$
7: **if** $IP == optIP$ **then**
8: $ec = EC(S_{IP})$
9: $S_{optP} = S_{IP}$
10: **end if**
11: **until** $IP > optIP$
12: **return** S_{optP}

C. Energy-Optimal Solution

Decreasing ec in Algorithm 2 until ϕ is not satisfied, we can obtain an f-schedule with an optimal energy consumption and a best throughput. We can answer our second problem formulated in Section II by this way. The experiments we performed reveal that this method is inefficient, however. A more efficient way is to integrate the use of priced timed automata.

By adding cost $iEC(p)$ and $uEC(p)$ to locations *idle* and *running* of ta_p, respectively, we obtain a priced timed automaton pta_p for processor p. Consequently, we use $npta_{\mathcal{M}} = \|_{p \in P} pta_p$ with a global clock $glbClk$ to describe system model $\mathcal{M}$. With this formalization, by applying UPPAAL CORA to check $npta_{\mathcal{M}} \models \mathbf{EF}\ deadlock$, we can obtain an energy consumption-optimal f-schedule of $\mathcal{M}$ with $optEC$, denoted by S_{optEC}. Taking $con(optEC)$ as the additional constraint, we can apply UPPAAL to check $nta_{\mathcal{M}} \models \mathbf{EF}\ deadlock \wedge con(optEC)$, and obtain an f-schedule S_{optE} with an optimal energy consumption and a best throughput.

V. Dealing with More Constraints

In this section, we discuss the integration of various kinds of constraints into our solutions. We first introduce the general framework of our method, then discuss the details of the two kinds of constraints, auto-concurrency constraints and buffer size constraints. Other kinds of constraints that may be integrated are also discussed.

The effects of constraints on the behavior of an SDFG are summarized in Table I. The first column lists the corresponding names of *readyS*, *sFiring* and *eFiring* for constraint *con*. The second column includes guard and updates we defined before. The 3-5 columns give the extra guard and updates for different constraints, auto-concurrency (ac), buffer size (bs) and both of them, respectively. Combining any of them with the second column forms the corresponding $readyS_{con}$, $sFiring_{con}$ and $eFiring_{con}$. For example, the enable condition of starting firing for an auto-concurrency constraint is represented as:

$$readyS_{ac} \equiv_{def} readyS \wedge hasF.$$

Replacing *readyS*, *sFiring* and *eFiring* in ta_p and pta_p defined in Section III with $readyS_{con}$, $sFiring_{con}$ and $eFiring_{con}$, respectively, we get NTA and NPTA of a system model with constraint *con*. The ways to find f-schedules S_{optP} and S_{optE} are the same as the system without these constraints.

TABLE I. Constrained Behavior of actor α

Constrained Behavior of α	NO Con.	Constraints (con)		
		auto-conc. (ac)	buffer size (bs)	both
$readyS_{con}$	$readyS$	$hasF$	$sufB$	$hasF \wedge sufB$
$sFiring_{con}$	$sFiring$	$addF$	$claB$	$addF \wedge claB$
$eFiring_{con}$	$eFiring$	$delF$	$relB$	$delF \wedge relB$

A. Auto-concurrency constraints

Suppose the number of auto-concurrent actors is limited to $conN$, which is equivalent to add a self-loop edge with $conN$ initial tokens to each actor. We use a set $conC(V)$ to control the number of concurrent firings of each actor $\alpha \in V$. The extra condition for *readyS*, updates for *sFiring* and *eFiring* are formulated as $hasF(\alpha)$, $addF(\alpha)$ and $delF(\alpha)$, respectively.

$$\begin{aligned} hasF(\alpha) &\equiv_{def} conC(\alpha) \leq conN \\ addF(\alpha) &\equiv_{def} conC'(\alpha) = conC(\alpha) + 1 \\ delF(\alpha) &\equiv_{def} conC'(\alpha) = conC(\alpha) - 1 \end{aligned}$$

Non auto-concurrency is a special case, which can be specified by $conN = 1$. Our method can also be used in a generalized case in which there is a constraint for each actor. For the generalized case, a set $conN(V)$ is used and above $conN$ are replace by $conN(\alpha)$.

B. Buffer size constraints

In practice, the storage space of a system must be bounded. The storage used by edges may be shared or separate. Here, we consider a separate buffer for each edge.

In line with [11], we choose a relatively conservative storage abstraction to leave more room for implementation. That is, when an actor starts firing, it claims the space of the tokens it will produce, and it releases the space of the tokens it consumes only when the firing ends. A set $tnb(E)$ is added to capture the buffer space used by each $e \in E$.

Suppose a schedule is constrained by a set $B(E)$, which limits the buffer usage of each edge, an enabled firing can not start when there is no sufficient space on its outgoing edges. The extra condition for *readyS* is formulated as $sufB(\alpha)$. When an actor starts a firing, it claims the required space on its outgoing edges. The update is formulated as $claB(\alpha)$. Only when a firing ends, it releases the space of its incoming edges. The update is formulated as $relB(\alpha)$.

$$\begin{aligned} sufB(\alpha) &\equiv_{def} \forall e \in OutE(\alpha) : prd(e) \leq B(e) - tnb(e) \\ claB(\alpha) &\equiv_{def} \forall e \in OutE(\alpha) : tnb'(e) = tnb(e) + prd(e) \\ relB(\alpha) &\equiv_{def} \forall e \in InE(\alpha) : tnb'(e) = tnb(e) - cns(e) \end{aligned}$$

A separate storage with other abstract is even easier to be integrated. For example, suppose an actor releases the space of its incoming edges when it starts a firing and claims and

occupies the space of its outgoing edges only when it ends a firing, we do not need the extra set $tnb(E)$ and updates $claB$ and $relB$. In $sufB(\alpha)$, $tnb(e)$ is simply replaced by $tn(e)$.

A shared memory usage can be easily integrated in the framework by modifying $sufB(\alpha)$ as

$$\forall e \in OutE(\alpha) : prd(e) \leq sM - \sum_{e \in E} tnb(e),$$

where sM is the bound of the shared memory. As our method explores exhaustively the state space, the optimal solutions are guaranteed.

C. Constraints on processors

Adding extra condition $t(\alpha, p) \geq 0$ to $readyS(\alpha)$, we can model the situation that some actors are not allowed to be allocated on some processors. The constraint that actor α is not allowed to run on processor p can be represented by $t(\alpha, p) = -1$.

The constraint that a processor has a higher priority than another can be modeled by the 'Priorities' feature of UPPAAL.

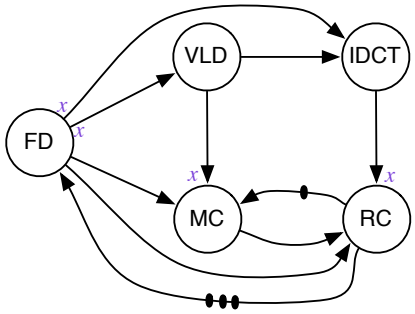
VI. Experimental Evaluation

We have implemented the translation from a system model to the input models of UPPAAL and UPPAAL CORA with different constraints, and the procedure to extract f-schedules from the returned traces. The experiments were performed on SDFGs of two practical dataflow applications with various parameters, running on a 2.90GHz CPU with 24M Cache and 384GB RAM. The applications include an MPEG-4 decoder [4] and a computation example [12]. The goal of the first case study is to evaluate the results and performance of our method under different parameters, including the sum of the repetition vector of an SDFG, the unfolding factor, the number of processors, and the buffer size constraints. The unfolding factor considered in the first case is relatively small. The second case study is mainly used to measure the impact of the unfolding factor.

A. MPEG-4 Decoder

The MPEG-4 decoder supports various kinds of frames. It is modeled as an Scenario-aware dataflow (SADF) model in [4]. Each scenario in an SADF model is actually an SDFG. We consider three scenarios, P30, P70 and P99.

The system models of the MPEG-4 decoder are shown in Fig. 6. The parameterized SDFG is shown in Fig. 6 (a), the value of x corresponding to Px. The repetition vector of each Px and the sums of the vectors (sumRV) are shown in Fig. 6 (b). Note that sumRVx is also the number of actors of the equivalent HSDFG of Px, and it is an important factor affecting the performance of almost all algorithms on SDFGs. The buffer size constraints are also a significant impact factor. We consider three cases: a model with a low buffer size bound, a high bound and no bound. The low bound is computed according to the method described in [13] to guarantee deadlock-free of an SDFG. The high bound is a minimal buffer size requirement to guarantee throughput-optimal of an SDFG when it is scheduled in an infinite number of homogeneous processors [11]. The sum of buffer size bounds of all edges of Px are shown in



(a)

frame	x	Repetition Vector					sumRV	Buffer Bound	
		FD	VLD	IDCT	MC	RC		Low	High
P30	30	1	30	30	1	1	63	128	149
P70	70	1	70	70	1	1	143	288	309
P99	99	1	99	99	1	1	201	404	425

(b)

proType	Energy (W)		Time (ps)				
	uCE	iCE	FD	VLD	IDCT	MC	RC
PT1	90	10	0	1	1	9	15
PT2	30	20	0	3	2	18	25

(c)

Fig. 6. System models of the MPEG-4 decoder. (a) Its SDFG; (b) the repetition vector of each Px, the sums of the vectors, and the considered bound of buffer size (c) the types of processor in the execution platform and the execution times of actors on different processors.

the last two columns of Fig. 6 (b). The information of its execution platform is shown in Fig. 6 (c). There are two types of processors, PT1 and PT2.

We show the experimental results for the MPEG-4 decoder in Table II, in which the parameters are shown in the first two rows and the first two columns. The others are the results. Px considered here are non auto-concurrent. The first column is the unfolding factor f. We consider 1-schedule and 2-schedule of the model. The second column is the number of processors $\#P$. We consider 2 processors, including one PT1 processor and one PT2 processor, and 4 processors, including two PT1 processors and two PT2 processors. The other 9 columns are the results for SDFG Px under a low buffer size bound, a high buffer size bound and no bound, respectively. The results include six parts. The first part is the optimal iteration period of Px, $optIP$; and the second part is the best energy consumption under $optIP$. The third part is the optimal energy consumption per iteration of Px, $optIEC$; and the fourth part is the best IP under $optIEC$. The fifth and sixth parts show the execution times and memory consumptions of the procedure finding $optIP$ of Px. If not marked specially, the unit of execution time is in second (s) and the unit of memory is in megabyte (MB).

When a low buffer size bound is used, the increasing of unfolding factor and number of processors have no improvement on the four values we have evaluated. Therefore, small unfolding factor and fewer processors are good enough

TABLE II. EXPERIMENTAL RESULTS FOR MPEG-4 DECODER

info / f #P		Low Bound P30	Low Bound P70	Low Bound P99	High Bound P30	High Bound P70	High Bound P99	No Bound P30	No Bound P70	No Bound P99
Optimal Iteration Period(optIP)										
1	2	83	163	221	82	162	220	63	117	158
	4	83	163	221	54	94	123	54	94	123
2	2	83	163	221	74	154	212	58	112	152.5
	4	83	163	221	48	88	117	43.5	82	111
Best Energy Consumption per Iteration under optIP										
1	2	9.2	18.0	24.3	7.4	13.8	18.4	7.2	13.9	18.8
	4	11.6	N	N	N	N	N	N	N	N
2	2	9.2	18.0	24.3	7.0	13.4	18.0	6.9	N	N *
	4	N	N	N	N	N	N	N	N	N
Optimal Energy Consumption per Iteration (optIEC)										
1	2	7.4	15.0	20.5	6.6	13.0	17.6	6.6	13.0	17.6
	4	11.3	22.5	30.6	9.5	18.3	24.7	8.3	16.1	21.7
2	2	7.4	15.0	20.5	6.5	12.9	17.6	6.5	12.9	17.6
	4	11.3	22.5	30.6	8.6	17.4	23.8	N	N	N
Best Iteration Period under optIEC										
1	2	131	251	338	102	182	240	74	148	206
	4	93	N	N	64	N	N	N	N	N
2	2	131	251	338	89.5	169.5	227.5	63.5	N	N
	4	N	N	N	N	N	N	N	N	N
Execution Time of Optimal IP (s)										
1	2	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.6	1.2
	4	0.1	0.1	0.2	0.2	0.5	0.6	2.1	12.6	21.3
2	2	0.0	0.0	0.1	0.1	0.1	0.2	2.2	14.7	22.6
	4	0.3	0.8	0.9	2.8	3.9	4.3	4m	29m	50m
Memory Consumed of Optimal IP (MB)										
1	2	4.7	4.8	4.9	4.8	5.0	5.2	7.9	13.3	21.9
	4	5.6	6.8	7.7	7.1	10.6	13.7	29.4	134.5	261.3
2	2	4.9	5.2	5.5	5.6	6.3	7.0	35.8	187.7	391.1
	4	11.9	18.8	26.8	34.3	54.0	70.8	1.5GB	9GB	19GB

* N: not finished after 3 hours or running out of memory.

for an optimal schedule of Px. Using a high bound provides more room for the improvement of iteration period and energy consumption, however, at the same time the performance of our method becomes worse. When without buffer size bound, the iteration period and energy consumption may be improved the most, but the performance becomes the worst.

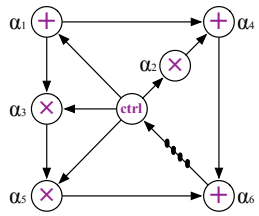
An implicated impact factor of this example is the number of the possible concurrent firings of the SDFG itself, i.e., x.

B. Computation Example

The system model of the computation example is shown in Fig. 7. It is described in a task graph in [12]. Actor *ctrl* connecting with original source and sink actors is added to limit the total latency. We have computed the results of unfolding factor from 1 to 10, and taken into account different combinations of values of three parameters : without buffer size bound and with a buffer bound; with and without auto-concurrency; 2 processors and 4 processors.

The experimental results are illustrated in Fig. 8. Some lines stop at the point that unfolding factor reaches 4 or 5, because the corresponding procedures for larger unfolding factors run out of memory. The throughput and energy consumption of schedules are improved by increasing unfolding factor; the degree of improvement decreasing accordingly. The buffer size bound and auto-concurrency constraints have larger impact on the cases with 4 processors than that with 2 processors.

In the two case studies , we present the unfinished points of the results to show the boundaries of the performance of our methods.



proType	Energy (W) uCE	Energy (W) iCE	Time (ps) +	Time (ps) ×	Time (ps) ctrl
PT1	90	10	2	3	0
PT2	30	20	5	7	0

Fig. 7. System model of the computation example.

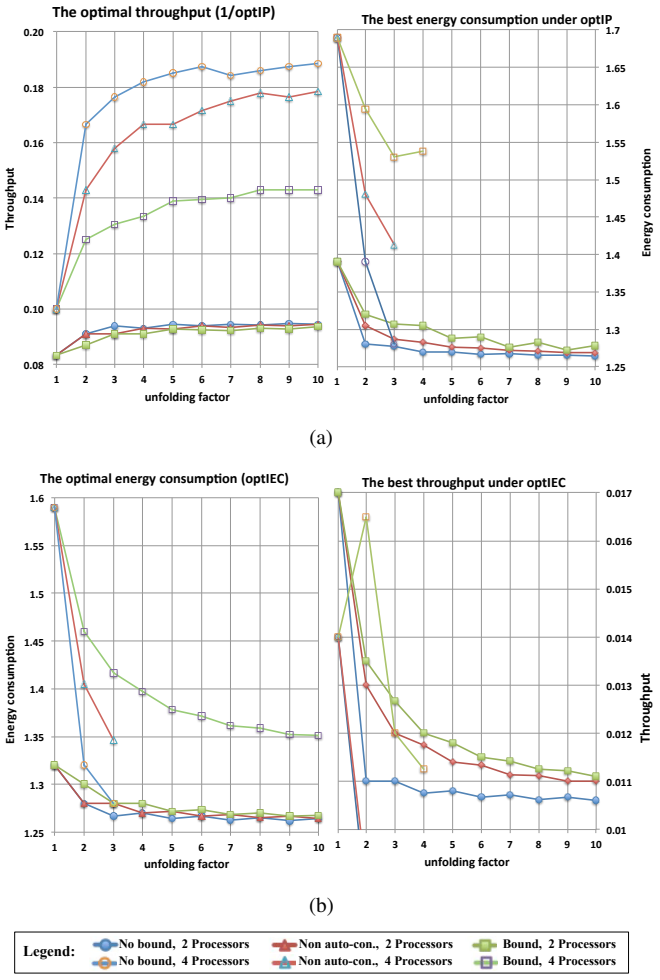


Fig. 8. The experimental results of the computation example. (a) the optimal throughput and the best energy consumption under the optimal throughput; (b) the optimal energy consumption and the best throughput under the optimal energy consumption.

VII. Related Work

Scheduling and mapping problem is generally NP-hard [14]. Researchers working on this problem usually target the solutions for particular application domain and specific optimization goals. And there are more heuristic methods than exact ones.

Scheduling SDFGs according to different optimization goals have been studied extensively [1], [15], [16], [17], [18], on both single processor platforms and mutilprocessor platforms. Many have been done on HSDFGs, while much less on the general SDFGs.

Theoretically, it is always possible to convert an SDFG to its equivalent HSDFG [10] and then use the available methods for HSDFGs. However, converting an SDFG to an HSDFG is very time-consuming when SDFGs scale up. The size of the HSDFG can be exponentially larger than the original SDFG in extreme cases [19].

Below we introduce the related work on static (compile-time) scheduling and mapping of SDFGs targeting on the throughput analysis and optimization.

The throughput (or IP) analysis of SDFGs on a single processor is trivial and there is no room for improvement. For a homogeneous multiprocessor platform, the number of processors can be infinite or finite.

The complexity of scheduling HSDFGs on an infinite number of processors is polynomial. The IP of HSDFGs can be computed with a critical path method [20]. It can be further improved by retiming [20] or unfolding [5]. Methods for analysis and improvement of the IP of SDFGs without converting them to the equivalent HSDFG are presented in [21], [22], [23]. There are many works on scheduling HSDFGs under IP constraints or optimal IP, such as [17], [5]. Methods directly work on SDFGs are presented in [24], [6], [25].

When a finite number of processors is considered, the problem becomes NP-complete. Heuristic methods of scheduling and mapping a weighted directed acyclic graph (DAG), or a task graph, for minimizing the completion time of the program are comprehensively compared in [26], [27]. By removing edges with initial tokens of a HSDFG, we can obtain the corresponding DAG. Then the methods for DAG are feasible for an equivalent HSDFG of an SDFG. A heuristic method can schedule SDFGs to achieve an optimal IP under the constraint on the number of processors [6], in which the mapping of SDFGs can be added straightforwardly without being converted into HSDFGs.

With the maturing of tools based on formal methods, an exhaustive search on the state space of the problem with reasonable scale becomes possible. A SAT-based mapping and scheduling of HSDFGs for throughput optimization is presented in [28], which can handle at most 30 actors and 8 processors. The work in [29] schedules HSDFGs under throughput constraints based on Constraint Programming, which is capable of dealing with at most 25 actors and 8 processors.

Scheduling and mapping SDFGs on heterogeneous multiprocessor platforms are even more intractable. We have only found a few works considering particular architectures. For example, the works in [30] and [31] apply heuristic methods on a multicore architecture where a core has a limited size of scratchpad memory (SPM).

We consider in this paper exact solutions for the general SDFGs and the general multiprocessor platforms. The mentioned constraints on buffer size or auto-concurrency are optional.

Our work presented in this paper is inspired by the previous work using model checking techniques on SDFGs, although the problems addressed are different.

Using model checking to schedule SDFGs according to a particular optimization goal was first presented by Geilen et al. [32], which targets at buffer minimization problem on a single processor with model checker SPIN [33]. There are also similar works, such as [34] and [35], using NuSMV [36] and SPIN, respectively, to solve the same problem.

The work in [37] presents a method which models a system described by a task graph binding to a heterogeneous multi-core platform in timed automata, and verifies the real-time schedulability of the system by UPPAAL [8]. A task graph can be transformed from an HSDFG as we have explained before.

The closest work to our methods is [38], which uses UPPAAL to analyze whether a timing constraint may be satisfied by a system represented as a SDFG binding to a heterogeneous multi-core platform. The main differences between the method in [38] and our method can be summarized as the following:

1) the problems addressed are different. [38] analyzes whether a feasible schedule exists for a given timing constraint, while our method gives optimal schedules combining the optimization goals with optimal throughput and energy consumption;
2) the input models are different. In [38], actors of an SDFG are binding to some core and edges to memories, while in our method, no binding is considered. On the contrary, the mapping between actors and processors according to the optimization goals is the result of our method;
3) the transformation method is different. [38] transforms each actor to a TA and each processor to a NTA. In our method, we combine the behavior of actors on processors. Consequently, there is one TA for each processor and the model for actors does not exist. The transformed NTA of our method has fewer concurrent processes.

Our energy consumption-optimal method is inspired by [12], in which a computation task modeled in priced timed automata is adopted for optimal energy consumption analysis. We also use this example to demonstrate our method with more parameters.

VIII. Conclusion

In this paper, we have presented two exact methods: finding static schedules with an optimal throughput and a best energy consumption, and finding static schedules with an optimal energy consumption and a best throughput for SDFGs on heterogenous multiprocessor platforms. Our methods can deal

with various parameters including unfolding factors, auto-concurrency constraints, buffer size constraints and other constraints on processors.

We use a model checking-based technique, which guarantees exact results of our methods. We transform a system model, which includes an SDFG and a multiprocessor platform, to a (priced) TA network of UPPAAL (CORA) and formalize the optimization goals as CTL formulas. The optimal schedules can then be computed from the traces returned by UPPAAL (CORA). Our experimental results show that our methods can deal with relatively large scale models within reasonable execution times, and that how different parameters impact on the results of different optimization goals.

References

[1] E. Lee and D. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput*, vol. 36, no. 1, pp. 24–35, 1987.

[2] V. Zivojnovic, S. Ritz, and H. Meyr, "Optimizing DSP programs using the multirate retiming transformation," *Proc. EUSIPCO Signal Process. VII, Theories Applicat*, 1994.

[3] S. Ritz, M. Willems, and H. Meyr, "Scheduling for optimum data memory compaction in block diagram oriented software synthesis," in *Proc. of the 1995 Acoustics, Speech, and Signal Processing Conf.* IEEE, 1995, pp. 2651–2654.

[4] B. Theelen, J.-P. Katoen, and H. Wu, "Model checking of scenario-aware dataflow with CADP," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2012, pp. 653–658.

[5] K. Parhi and D. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *Computers, IEEE Transactions on*, vol. 40, no. 2, pp. 178–195, 1991.

[6] X.-Y. Zhu, M. Geilen, T. Basten, and S. Stuijk, "Static rate-optimal scheduling of multirate DSP algorithms via retiming and unfolding," in *Proc. of the 18th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012*. IEEE, 2012, pp. 109–118.

[7] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.

[8] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 134–152, 1997.

[9] G. Behrmann, K. G. Larsen, and J. I. Rasmussen, "Priced timed automata: Algorithms and applications," in *Formal Methods for Components and Objects*. Springer, 2005, pp. 162–182.

[10] S. Sriram and S. S. Bhattacharyya, *Embedded multiprocessors: scheduling and synchronization*. CRC Press, 2009.

[11] S. Stuijk, M. Geilen, and T. Basten, "Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs," *IEEE Trans. Comput*, vol. 57, no. 10, pp. 1331–1345, 2008.

[12] P. Bouyer, U. Fahrenberg, K. G. Larsen, and N. Markey, "Quantitative analysis of real-time systems using priced timed automata," *Communications of the ACM*, vol. 54, no. 9, pp. 78–87, 2011.

[13] M. Adé, R. Lauwereins, and J. Peperstraete, "Data memory minimisation for synchronous data flow graphs emulated on dsp-fpga targets," in *Proc. of the 34th annual DAC*, 1997, pp. 64–69.

[14] M. R. Garey and D. S. Johnson, "Computers and Intractability: a guide to NP-completeness," 1979.

[15] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software synthesis from dataflow graphs*. Springer, 1996, vol. 360.

[16] S. Sriram and S. Bhattacharyya, *Embedded multiprocessors: scheduling and synchronization*. CRC, 2000.

[17] L. Chao and E. Sha, "Scheduling data-flow graphs via retiming and unfolding," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 12, pp. 1259–1267, 1997.

[18] R. Govindarajan, G. R. Gao, and P. Desai, "Minimizing buffer requirements under rate-optimal schedule in regular dataflow networks," *The Journal of VLSI Signal Processing*, vol. 31, no. 3, pp. 207–229, 2002.

[19] V. Zivojnovic and R. Schoenen, "On retiming of multirate DSP algorithms," in *Proceedings of the Acoustics, Speech, and Signal Processing, 1996. on Conference Proceedings., 1996 IEEE International Conference-Volume 06*. IEEE Computer Society, 1996, pp. 3310–3313.

[20] C. Leiserson and J. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.

[21] N. Liveris, C. Lin, J. Wang, H. Zhou, and P. Banerjee, "Retiming for synchronous data flow graphs," in *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*. IEEE Computer Society, 2007, pp. 480–485.

[22] X.-Y. Zhu, "Retiming multi-rate DSP algorithms to meet real-time requirement," in *Proc. of the 13th Design, Automation and Test in Europe (DATE)*. IEEE, 2010, pp. 1785–1790.

[23] X.-Y. Zhu, T. Basten, M. Geilen, and S. Stuijk, "Efficient retiming of multirate DSP algorithms," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 6, pp. 831–844, 2012.

[24] R. Govindarajan and G. Gao, "Rate-optimal schedule for multi-rate DSP computations," *The Journal of VLSI Signal Processing*, vol. 9, no. 3, pp. 211–232, 1995.

[25] A. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. Bekooij, B. Theelen, and M. Mousavi, "Throughput analysis of synchronous data flow graphs," in *Sixth International Conference on Application of Concurrency to System Design, ACSD 2006.* IEEE, 2006, pp. 25–36.

[26] Y.-K. Kwok and I. Ahmad, "Benchmarking and comparison of the task graph scheduling algorithms," *Journal of Parallel and Distributed Computing*, vol. 59, no. 3, pp. 381–422, 1999.

[27] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 35, 2011.

[28] W. Liu, M. Yuan, X. He, Z. Gu, and X. Liu, "Efficient SAT-based mapping and scheduling of homogeneous synchronous dataflow graphs for throughput optimization," in *Real-Time Systems Symposium, 2008*. IEEE, 2008, pp. 492–504.

[29] A. Bonfietti, M. Lombardi, M. Milano, and L. Benini, "Throughput constraint for synchronous data flow graphs," *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 26–40, 2009.

[30] J. Choi, H. Oh, S. Kim, and S. Ha, "Executing synchronous dataflow graphs on a SPM-based multicore architecture," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 664–671.

[31] W. Che and K. S. Chatha, "Unrolling and retiming of stream applications onto embedded multicore processors," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 1272–1277.

[32] M. Geilen, T. Basten, and S. Stuijk, "Minimising buffer requirements of synchronous dataflow graphs with model checking," in *Proc. of the 42nd Annu. Design Automation Conf.* ACM, 2005.

[33] G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[34] Z. Gu, M. Yuan, N. Guan, M. Lv, X. He, Q. Deng, and G. Yu, "Static scheduling and software synthesis for dataflow graphs with symbolic model-checking," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE, 2007, pp. 353–364.

[35] P. H. Hartel, T. C. Ruys, and M. C. Geilen, "Scheduling optimisations for SPIN to minimise buffer requirements in synchronous data flow," in *Proceedings of the 2008 International Conference on Formal Methods in Computer-Aided Design*. IEEE Press, 2008, p. 21.

[36] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri, "NuSMV: a new symbolic model checker," *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 4, pp. 410–425, 2000.

[37] J. Madsen, M. R. Hansen, K. S. Knudsen, J. E. Nielsen, and A. W. Brekling, "System-level verification of multi-core embedded systems using timed-automata," in *Proceedings of the 17th World Congress International Federation of Automatic Control Seoul, Korea*, 2008, pp. 9302–9307.

[38] M. Fakih, K. Grüttner, M. Fränzle, and A. Rettberg, "Towards performance analysis of SDFGs mapped to shared-bus architectures using model-checking," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013, pp. 1167–1172.