Constrained Global Scheduling of Streaming Applications on MPSoCs

Jun Zhu, Ingo Sander, Axel Jantsch
Royal Institute of Technology, Stockholm, Sweden
{junz, ingo, axel}@kth.se

Abstract— We present a global scheduling framework for synchronous data flow (SDF) streaming applications on MPSoCs, based on optimized computation and contention-free routing. The global scheduling of processors computing and communication transactions are formulated as constraint based problem, to avoid the scheduling overhead in TDMA-like heuristic schemes. A public domain constraint solver is exploited to solve the NP-complete scheduling efficiently, together with problem specific constraint modeling techniques. Experimental results show that the proposed framework can achieve a high predictable application throughput with minimized buffer cost. For instance, for applications in communication domain, higher throughput (up to 87%) has been observed with less buffer cost, compared to scenarios considering the heuristic scheduling overhead.

I. Introduction

Nowadays, multi-processor systems-on-chip (MPSoCs) are very popular computing platforms for modern embedded systems [1]. While they enable distributed processing, it is extremely challenging to design embedded streaming applications in communication and multimedia domains on MPSoCs, when there are non-functional constraints from both hardware and software modules, such as processor speed, buffer size, energy budget, and scheduling policy.

Synchronous data flow (SDF) model [2] has been widely used to design and analyze streaming applications on multi-processors [2, 3]. An illustrative SDF application is depicted in Fig. 1(a). The nodes denote computation *processes*, and the edges denote communication *channels*. Each time a process executes, it *consumes* (*produces*) a fixed number of tokens *from input* (*into output*) channels. These numbers are denoted as symbols at the each side of channels. For instance, process p_j consumes $m_{i,j}$ tokens from channel $ch_{i,j}$ and produces $n_{j,k}$ tokens into channel $ch_{j,k}$ on each invocation (firing).

Using one instance of the illustrative model with specified token rates, as illustrated in Fig. 1(b), we then allocate it onto a dual-processor architecture. Each process p_x has a worst case execution time (WCET) $t_{C,x}$, and each channel $ch_{x,y}$ is implemented as finite FIFO buffer with token storage $\gamma_{x,y}$. The hard real-time inter-processor communication latency is captured by an identity process p_δ with delay $t_{C,\delta}$. While processes are *enabled* when they have enough input data tokens and output buffer space, p_i and p_j can only be scheduled sequentially in one single processor μp_1. The scheduling problem on such kind of multi-processors with resource constraints has been known to be NP-complete [4]. Recently, heuristic algorithms have been proposed to provide predictable performance on MPSoCs [3, 5], yet as argued below, they may lead to sub-optimal solutions.

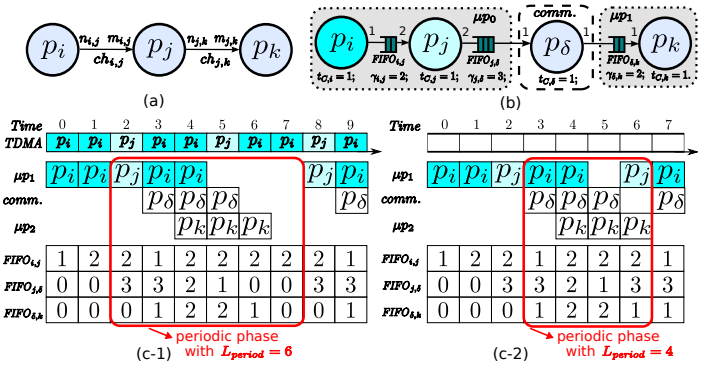

Fig. 1. Allocation and scheduling of an illustrative application. (a) an illustrative application; (b) application instance allocated onto buffer constrained dual-processor; (c-1) Scheduling with TDMA scheme on μp_1; (c-2) Throughput-optimized scheduling.

Motivation. Let p_i and p_j to be scheduled with a heuristic time-division multiple-access (TDMA) scheme [3], as marked horizontally on the timeline in Fig. 1(c-1). Each process is not allowed to fire only when it is *enabled*, but it should also be the process which gets the allocated TDMA time slots. Accordingly, a schedule is built below. The running processes on each processor and the FIFO usage are listed out vertically. At time tag 0, p_i starts the execution and requires space 1 for one output token on $FIFO_{i,j}$. At time tag 1, p_i finishes the previous firing, outputs 1 result token, and starts a new firing. As the scheduling evolves, p_δ and p_k execute once they are *enabled*, and p_i and p_j are scheduled in TDMA scheme. However, the TDMA assigned processes can be stalled, when they violate resource constraints (bounded buffer capacities or no enough input tokens). For instance, p_j is stalled ($\gamma_{j,\delta} = 3$) at time tag 5, and so is p_i ($\gamma_{i,j} = 2$) at time tag 6 and 7. From time tag 2 to 7, the schedule enters a periodic phase with length $L_{period} = 6$, in which the application throughput is guaranteed by process firing patterns. On the other hand, in Fig. 1(c-2), another optimized schedule exists without using TDMA scheme in μp_1. We observe a 50% throughput gain in this proposed optimized cyclic static schedule, i.e., $L_{period} = 4$, on the same platform.

Although TDMA-like or list scheduling heuristics can be used to design predictable distributed systems [3, 5], they have drawbacks in the following :

- They can not avoid the overhead in time slots allocation, which degrades application performance (Section VI);

- They are in lack of global optimization mechanisms on MPSoCs, where numerous processors or communication links are concurrently shared by different applications.

As the **contribution in this paper**, we propose a new scheduling framework on MPSoCs to build global schedules for both processors computing (process execution) and communication transactions (real-time traffic-flow). While buffer cost is minimized, a high predictable application throughput is guaranteed based on optimized computation and contention-free routing. The framework has been implemented on a public domain constraint solver Gecode [6].

This paper is structured as follows: the related work is introduced in Section II. Our MPSoC architecture platform is introduced in Section III. We present our constraint based scheduling framework in Section IV and the constraint programming techniques used in Section V. Section VI shows our experimental results. Finally, Section VII concludes the paper.

II. Related work

Many authors have explored the scheduling of SDF models on multi-processors at compile-time. Lee and Messerschmitt [2] first present general techniques to construct periodic admissible parallel schedules (PAPS) on limited number of multi-processors. Later, Govindarajan et al. [7] propose linear programming (LP) formulation to obtain maximal throughput and minimized buffer cost for SDF models without computation (number of processors) constraints. However, the scheduling problem turns to be more difficult on resource constrained platforms, when processor quantity, interconnection bandwidth, and buffer resources are bounded.

Eles et al. [5] first address the scheduling on distributed systems with communication protocols optimization. With optimized bus accessing, application throughput is maximized based on list scheduling (heuristic order) for task graph models. While task graphs can be viewed as special cases of acyclic SDF models with no overlap between different iterations of execution, buffer cost has not been addressed as resource constraints. In [3], Stuijk et al. propose a mapping and TDMA/list scheduling design flow for throughput constrained SDF applications on MPSoCs. Both paper are based on heuristic TDMA or list scheduling. We argue about the scheduling overhead and the lacking of global optimization on performance metrics, e.g., application throughput in Fig. 1(c-1).

Inspired by the success of (model-checking, SAT and constraint programming) techniques in solving NP-complete problems, Geilen et al. [8] first use model-checker to determine the minimal deadlock-free buffer cost to schedule SDF models (no computation constraints). Liu et al. [9] use SAT-solver to explore the mapping and scheduling of homogeneous SDF (HSDF) models[1] on multi-processors to *maximize* application throughput. However, a regular SDF model must be transformed (expanded) to a HSDF model to apply their techniques[2], which dramatically increases the problem size [10]. Constraint programming tools are used as well in the scheduling of task graphs on MPSoCs, without violating computation capacity and communication bandwidth [11, 12]. In our previous work [13], constraint based paradigm has been proposed for SDF models scheduling on hybrid CPU/FPGA platforms, in which communicate happens via dedicated (not shared) FIFO channels with ignored delay. Yet, all papers above do not consider the global optimization of distributed computing and communication transactions on MPSoCs. To the best of our knowledge, our work is the first to address this problem using combinatorial optimization techniques.

[1]HSDF models are special cases of SDF models, with all input/output token rates are 1.

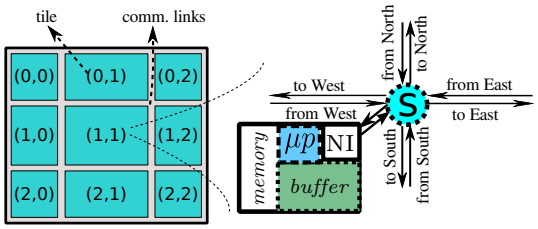

Fig. 2. A 3 × 3 regular mesh MPSoC architecture.

III. Architecture platform

The cornerstone of streaming applications with predictable performance is the architecture platform with deterministic real-time properties. In this paper, we consider the regular 2-D mesh tiled MPSoCs with communication happens via hard real-time networks-on-chip (NoCs) infrastructure [14, 15], as exemplified in Fig. 2. Nevertheless, our method is not limited to this particular architectural template.

Each tile $tile_n$ ($n \in \mathbb{N}_0$) consists of a processor (μp_n), an application memory, a token buffer $buffer_n$, and a network interface (NI). $tile_n$ is labeled as (x_n, y_n) in the mesh topology, where x_n and y_n correspond to the row and column indexing numbers respectively. Our work starts with a fixed allocation from processes to tiles. Hence, all application memory modules are pre-determined on the mesh. Instead, we focus on the analysis of routing, scheduling, and buffer properties on the dashed modules in Fig. 2, i.e., NoC switches, processors, and token buffers.

Tiles are connected to the communication network through switches (s), and communicate with each other via unidirectional communication links.

IV. Constraint based scheduling

The binding of streaming applications onto the target MPSoC architectures is a refinement process with resource limitations and real-time requirement. Here, we use constraint based formulation to model the application to architecture mapping, communication routing, flow control, and computation scheduling in this design flow.

[2]In [9], models contain up to 30 HSDF processes have been considered, but the equivalent HSDF H263 model in our experiments (Section VI) has 4754 processes.

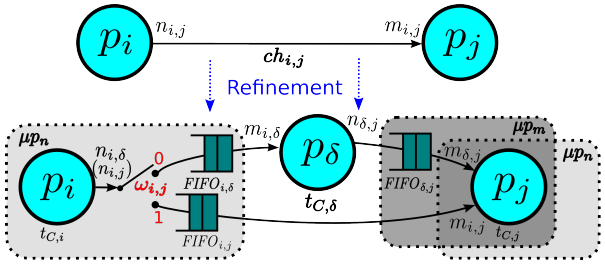

Fig. 3. A template of refined producer-consumer pair, in which $\omega_{i,j}$ denotes whether channel $ch_{i,j}$ is implemented as intra-processor communication.

We formalize our problem based on a general producer-consumer processes pair as illustrated in the upper part of Fig. 3. Each (acyclic or cyclic) SDF model can be analyzed as a composition of a set of concurrent producer-consumer pairs.

A. Mapping

A global computation and communication scheduling framework on MPSoCs needs to be aware of the mapping decisions from designers first. Such mapping decisions are modeled by two sets of decision variables: α and ω. A boolean variable $\alpha_{i,\mu p_n}$ denotes the presence of p_i on a processor μp_n. We assume that different instances of a process can only execute on one dedicated processor, which can be formalized in the following constraint.

Constraint 1 *(Single residence) Each process p_i needs one (and only one) specified processor for computation.*

$$\sum_{\mu p_n \in \boldsymbol{U}} \alpha_{i,\mu p_n} \equiv 1, \quad \forall p_i \in \boldsymbol{P} \tag{1}$$

in which $\boldsymbol{P}$ is the set of processes in application models, and $\boldsymbol{U}$ is the set of processors in the architecture platform.

Redundantly, a boolean variable $\omega_{i,j}$ denotes how a channel $ch_{i,j}$ is implemented, as illustrated in Fig. 3. When $\omega_{i,j} = 0$, p_i and p_j are mapped onto different processors (μp_n and μp_m), $ch_{i,j}$ is implemented as inter-processor communication with buffers $FIFO_{i,\delta}$ and $FIFO_{\delta,j}$, and the hard real-time communication is captured by process p_δ with bounded latency $t_{C,\delta}$ and no packet loss ($m_{i,\delta} = n_{\delta,j}$). Otherwise, $\omega_{i,j} = 1$, p_i and p_j are mapped onto the same tile, and the intra-tile communication $ch_{i,j}$ is implemented as $FIFO_{i,j}$ with ignored latency. Processors in the architecture platform are *homogeneous* in the sense that they are the same type and each p_j has the same WCET $t_{C,j}$ being mapped onto any processors. $\omega_{i,j}$ can be defined as the following.

Constraint 2 *(Correlated mapping decision) To be correlated with $\alpha_{i,\mu p_n}$ and $\alpha_{j,\mu p_m}$, $\omega_{i,j}$ denotes whether p_i and p_j are mapped onto the same processor.*

$$\omega_{i,j} = (x_n == x_m) \wedge (y_n == y_m),\ \alpha_{i,\mu p_n} = 1, \alpha_{j,\mu p_m} = 1. \tag{2}$$

In our previous work [13], event models based on cumulative functions were used to capture process working load and pressing capabilities. For instance, in the producer-consumer pair in the upper part of Fig. 3, an arrival function $R_{i,j}(t)$ is defined as the accumulated data tokens arrived in $ch_{i,j}$ until time tag t, a service function $C_{i,j}(t)$ is defined as the accumulated data tokens consumed by p_j until time tag t, and a demand function $D_{i,j}(t)$ captures the extra output buffer space requirement when p_i is executing as defined in the following.

$$D_{i,j}(t) = \begin{cases} R_{i,j}(t) + n_{i,j}, & \text{if } p_i \text{ is executing;} \\ R_{i,j}(t), & \text{if } p_i \text{ is stalling.} \end{cases} \tag{3}$$

Accordingly, the execution semantics of SDF applications can be formalized (refer to [13]).

In this paper, to be aware of the mapping decisions (the routing and flow control information later in Section IV. C as well), the process scheduling status derived from event models needs to be refined.

Constraint 3 *(Mapping & scheduling association) All the processes assigned to each processor can only execute (be scheduled) sequentially at any time. This mapping and scheduling association is formalized as:*

$$\sum_{p_i \in \boldsymbol{P}} \alpha_{i,\mu p_n} W_j(t) \in \{0,1\},\ \forall \mu p_n \in \boldsymbol{U}, t \in \mathbb{N}_0 \tag{4}$$

in which $W_j(t)$ denotes the 1-0 *(computing or stalling) status of each process p_j and is defined as:*

$$W_j(t) = max(L_j(t), L_j(t + \Delta_t)), \quad \forall \Delta_t \in [1, t_{C,j}] \tag{5}$$

$$L_j(t+1) = \frac{C_j(t+1) - C_j(t)}{m_{i,j}} \in \{0,1\}$$

with $L_j(t)$ as a helper function.

B. Template based Buffering Analysis

Here, based the refined producer-consumer template in Fig. 3, some buffer properties and constraints can be formulated from our event models and the mapping-aware decisions.

Property 1 *(Buffer usage) The buffer usages of $FIFO_{i,\delta}$, $FIFO_{\delta,j}$, and $FIFO_{i,j}$ in the template (Fig. 3) at time tag t are denoted as $B_{i,\delta}(t)$, $B_{\delta,j}(t)$, and $B_{i,j}(t)$ respectively, which are defined as:*

$$B_{i,\delta}(t) = D_{i,\delta}(t) - \neg\omega_{i,j}(C_{i,\delta}(t) - B^0_{i,\delta}) - \omega_{i,j}(C_{i,j}(t) - B^0_{i,j}) \tag{6}$$

$$B_{\delta,j}(t) = \neg\omega_{i,j} B'_{\delta,j}(t) = D_{\delta,j}(t) - \neg\omega_{i,j}(C_{\delta,j}(t) - B^0_{\delta,j}) \tag{7}$$

$$B_{i,j}(t) = \omega_{i,j} B_{i,\delta}(t) \tag{8}$$

in which $B^0_{i,\delta}(t)$, $B^0_{\delta,j}(t)$, and $B^0_{i,j}(t)$ are the initial data tokens (at time tag 0) in each buffer.

Furthermore, the asynchronous behaviors of the buffers, caused by data buffering or processing latency (computation and communication), are captured as the following.

Constraint 4 *(Asynchronous buffer) The incoming data tokens in buffers take at least the WCET of the consumer process to be consumed. For buffers $FIFO_{i,\delta}$, $FIFO_{\delta,j}$, and $FIFO_{i,j}$ in the template (Fig. 3), the asynchronous behaviors can be*

formalized as:

$$R_{i,\delta}(t) \geqslant \neg\omega_{i,j} C_{i,\delta}(t + t_{C,\delta}) + \omega_{i,j} C_{\delta,j}(t + t_{C,j}) - B^0_{i,\delta} \quad (9)$$

$$R_{\delta,j}(t) = \neg\omega_{i,j} R_{\delta,j}(t) \geqslant \neg\omega_{i,j} C_{\delta,j}(t + t_{C,j}) - B^0_{\delta,j} \quad (10)$$

$$R_{i,j}(t) = \omega_{i,j} R_{i,\delta}(t) \quad (11)$$

Depending on the buffer organization, there are two measures of buffer requirement when different FIFOs are assigned to the same tile (buffer) [8], i.e., the FIFO implementations of different communication channels are either partitioned disjointly or sharing a single space on a buffer. Accordingly, the buffer requirement of these two mechanism can be formalized in the following.

Property 2 *(Buffer cost – disjoint partition) When FIFOs are organized on each tile as disjoint buffer partitions, the buffer cost $\gamma_{Sum'}$ in the platform is simply the sum of individual FIFO sizes.*

$$\gamma_{Sum'} = \sum_{i,j,\delta} (\omega_{i,j}\gamma_{i,j} + \neg\omega_{i,j}(\gamma_{i,\delta} + \gamma_{\delta,j})),\ \forall p_i, p_j \in \boldsymbol{P} \quad (12)$$

$$where \quad \gamma_{i,j} \geqslant B_{i,j}(t), \gamma_{i,\delta} \geqslant B_{i,\delta}(t), \gamma_{\delta,j} \geqslant B_{\delta,j}(t),\ \forall t \in \mathbb{N}_0$$

in which $\gamma_{i,j}$, $\gamma_{i,\delta}$, and $\gamma_{\delta,j}$ denote the sizes of the disjoint FIFOs in the template (Fig. 3).

Property 3 *(Buffer cost – shared partition) When FIFOs are sharing space on each tile, the buffer cost $\gamma_{Sum''}$ in the platform is the sum of the shared buffer space on individual tiles.*

$$\gamma_{Sum''} = \sum_{n} \gamma_{tile_n} \quad (13)$$

$$where\ \gamma_{tile,n} \geqslant \sum_{i,j} (a_{i,\mu p_n} B_{i,\delta}(t) + a_{j,\mu p_n} B_{\delta,j}(t) + a_{i,\mu p_n}\omega_{i,j} B_{i,j}(t)),\ \forall p_i, p_j \in \boldsymbol{P}, \forall t \in \mathbb{N}_0$$

in which γ_{tile_n} denotes the size of the buffer as one shared partition on tile $tile_n$.

C. Routing and contention-free flow control

When processes in a producer-consumer pair are assigned to different processors, routing is needed for the inter-tile data communication. Here, we assume deterministic *X-Y* routing on NoC infrastructure. The bidirectional routing decisions are capture by two sets of triple-value (±1 and 0) variables β^r and β^c on the direction of *rows* and *columns* of physical links respectively, i.e., +1 (-1) represents routing a packet flow on the positive (negative) direction of a link on *X* or *Y* axis, and 0 represents no packet flow. β^r and β^c can be associated with the mapping variables α in the following.

Constraint 5 *(X-Y routing) Let $\beta^r_{ch_{i,j},l_k}$ represent the routing decision on a row link l_k between tiles (x_k, y_k) and $(x_k + 1, y_k)$, i.e., how channel $ch_{i,j}$ is assigned to l_k. The routing on the primary X axis is formalized as:*

$$\beta^r_{ch_{i,j},l_k} = \begin{cases} +1, & x_m > x_n, y_k = y_n, \forall x_k \in [x_n, x_m); \\ -1, & x_n > x_m, y_k = y_n, \forall x_k \in [x_m, x_n); \\ 0, & otherwise. \end{cases} \quad (14)$$

$$\forall p_i, p_j \in \boldsymbol{P}, \alpha_{i,\mu p_n} \equiv 1, \alpha_{j,\mu p_m} \equiv 1, m \neq n.$$

Similarly, the subsequent routing decisions $\beta^c_{ch_{i,j},l_k}$ on column links (Y axis) can be defined.

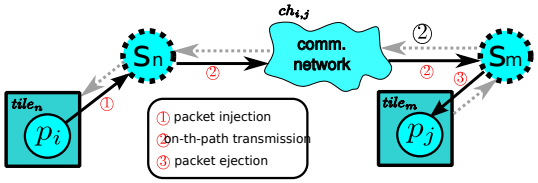

Fig. 4. Three phases to route a packet in inter-tile channel $ch_{i,j}$.

When a channel $ch_{i,j}$ is assigned to NoC , the communication time needed depends not only on the number of transferred data tokens but also the dynamic link bandwidth during the transaction. In hard real-time NoCs, contention-free routing at the traffic flow level is usually adopted, in which a packet (data token) reserves a circuit switching before the transmission finishes, e.g., in Æthereal [15]. For one application flow (channel), the routing of a packet from source tile to destination tile via NoC switches and links consists of three phases: packet *injection*, *on-the-path* transmission, and packet *ejection*, as illustrated in Fig. 4. While NoC resources (switches and links) can be shared by different application flows, packet *injection* (*ejection*) congests spatially when multiple producer (consumer) processes are mapped onto the same tile, so do *on-the-path* transmissions when different application flows are sharing the same links. However, different application flows can be contention-free by avoiding the competition on the same links at the same time (scheduled temporally), which is formalized as the following.

Constraint 6 *(Contention-free traffic flow scheduling) When communication channels are assigned to inter-tile physical links, the packet injection, ejection, and inter-tile traffic flow on rows of communication links are scheduled to avoid contention, as formalized in Eq. 15, 16, and 17 respectively.*

$$\sum_i \alpha_{i,\mu p_n} W_\delta(t) \in \{0, 1\}, \quad (15)$$

$$\sum_j \alpha_{j,\mu p_n} W_\delta(t) \in \{0, 1\}, \quad (16)$$

$$\sum_{i,j} \beta^r_{ch_{i,j},l_k} W_\delta(t) \in \{0, \pm 1\}, \sum_{i,j} |\beta^r_{ch_{i,j},l_k}| W_\delta(t) \in \{0, 1, 2\}, \quad (17)$$

$$\forall p_i, p_j \in \boldsymbol{P}, \mu p_n \in \boldsymbol{U}, t \in \mathbb{N}_0$$

with $W_\delta(t)$ (see Eq. 5) to denote the data transmission 1-0 *(working or idle) status on NoC modeled by p_δ. Similarly, the scheduling of traffic flow on columns of communication links can be formalized as in Eq. 17, which is omitted for clarity.*

D. Real-time constraints

Embedded streaming applications are usually required to run indefinitely with bounded buffers. Accordingly, such a balance equation holds for all the producer-consumer pairs in bounded SDF models [2]:

$$r_i \cdot n_{i,j} = r_j \cdot m_{i,j} \tag{18}$$

in which r_i and r_j denote how many times p_i and p_j fire respectively. A vector of the non-trivial minimal firing times for all processes defined by the set of balance equations is called *repetition vector*.

For streaming applications with real-time constraints, the efficient execution means the streaming services are delivered on-demand to the end-user, neither too fast nor too slow. Thus, a predetermined application throughput needs to be guaranteed, as formalized in the following.

Constraint 7 *(Application output throughput) After a transient phase τ_0 ($\tau_0 > 0$) with no stable output tokens, a specified output throughput ρ_{out} should be sustained at the application sink process p_j, which is guaranteed by a periodic phase (see fig 1) with length L_{period}.*

$$C_j(\tau_0 + k \cdot L_{period}) - C_j(\tau_0) = k \cdot J \cdot r_j \cdot m_{i,j}, \tag{19}$$
$$L_{period} = \lceil \frac{J \cdot r_k \cdot m_{i,j}}{\rho_{out}} \rceil, J \in \mathbb{N} \backslash \{\infty\}, k \in \mathbb{N}_0$$

in which k specifies the iteration number of periodic phases, and J (called unroll factor) denotes how many cycles processes fire as stated in the repetition vector in one periodic phase.

Redundantly, any process p_i has the following redundant constraint (derived from Eq. 18 and Eq. 19):

$$C_i(\tau_0 + k \cdot L_{period}) - C_i(\tau_0) = k \cdot J \cdot r_i \cdot n_{i,j} \tag{20}$$

Here, we omit the formulation of constraints on periodic phase checking (refer to [13]).

V. Constraint programming techniques

To apply the constraint programming approach, we encode both the SDF execution semantics [13] and the scheduling problem on MPSoCs with constraint solver Gecode [6], which is a library written in C++. Some modeling techniques have been conducted to improve the computation efficiency in solutions finding:

- **Redundant constraints.** See Eq. 7, 10, and 20.
- **Domain and constraints reduction.** A lower bound for any FIFO [8] can be computed as

$$n_{i,j}+m_{i,j}-gcd(n_{i,j},m_{i,j})+B^0_{i,j} \bmod gcd(n_{i,j},m_{i,j}) \tag{21}$$

  to prune infeasible (dead-lock) variable value domains. Furthermore, although $t \in \mathbb{N}_0$ holds for all timing related constraints, we only implement and evaluate them in time period $[0, \tau_0 + L_{period}]$ once periodic phase can happen.

TABLE I
Comparison of scenarios with varying OH. (3×3 platform).

specification				*SCP-OH*[a]		*SCP*	
app.	#[b]	*thru. req.*	*J*	$\gamma_{Sum'}(\gamma_{Sum''})$	*time*[c]	$\gamma_{Sum'}(\gamma_{Sum''})$	*time*[c]
Modem	2	3.125e-2	1	-[d]	352	98(45)	3.0e3
	2	1.667e-2	1	98(46)	5.0e3	92(41)	1.4e3
Wireless	2	2.5e-2	1	-	422	121(53)	4.7e4
	2	1.7e-2	1	123(49)	6.4e5	120(48)	1.2e3

[a] 50% OH in computation latency, plus 100% OH in communication latency.
[b] The number of application instances mapped onto platform.
[c] The solutions finding time (ms), branched and explored with $\gamma_{Sum'}$ and γ.
[d] The problem is infeasible.

- **Branching and exploration.** To construct the search tree, the branching variables are prioritized as follows: $\gamma_{Sum'}$ ($\gamma_{Sum''}$), γ_{tile_n}, and C. Empirically, the exploration starts with minimal values for all variables, which also guarantees that the first solution found has minimized buffer cost $\gamma_{Sum'}$ ($\gamma_{Sum''}$).

VI. Experimental results

We have performed some experiments on a few benchmarks to demonstrate how our Scheduling approach based on Constraint Programming (*SCP*) works with Gecode solver in practice. We first evaluate the effects of scheduling overhead (OH) which exists in TDMA-like scheduling, and then compare *SCP* with the heuristic PAPS [2] in buffer cost. All experiments are carried out on a HP xw4600 Linux workstation with a Quad-Core[3] 2.40GHZ processor and 4GB of RAM.

A. Evaluation of scheduling overhead

Usually, the predictable time slots allocation in TDMA or list scheduling is modeled by increasing the computation or communication latency with the postponed time. We slightly (compared with [3]) specify the OH to be 50% in computation and 100% in communication, i.e., the OH caused by TDMA time wheel allocation before the computation or communication can happen. Our *SCP* approach is then used to estimate the best scheduling quality of heuristics, denoted as *SCP-OH*. We have used two benchmarks from communication domains, i.e., a Modem application from [16], and a Wireless application from [17]. The architecture platform is a 3×3 mesh-based MPSoC.

Table I shows the experimental results. Each time two instances of the same applications are allocated onto the MPSoC empirically with specified throughput requirement, and the unroll factor J (Constraint 7) is fixed to 1. The search tree is branched and explored with buffer storage $\gamma_{Sum'}$ and γ_{tile_n}, then $\gamma_{Sum''}$ (Property 3) is calculated. We see our *SCP* can meet much higher (up to 87%) throughput requirement than *SCP-OH*, without much increase in buffer cost. Therefore, we argue about the performance degradation caused by the OH in heuristic scheduling on MPSoCs. Furthermore, buffer costs measured in $\gamma_{Sum''}$ show a great reduction ($55\sim65\%$)

[3] One core is actually utilized, since we do not explore multi-thread searching in this paper (see Section VII).

TABLE II
COMPARISON OF SCHEDULING METHODS (2×2 PLATFORM).

specification			*PAPS*			*SCP*		
app.	#	*thru. req.*	J	$\gamma_{Sum'}(\gamma_{Sum''})$	*time*	J	$\gamma_{Sum'}(\gamma_{Sum''})$	*time'(time")*[a]
Bipartite	1	1.101e-1	3	510(510)	120	3	36(31)	2.6e3(2.3e5)
	1	1.096e-1	2	352(352)	50	1	36(31)	1.8e3(1.4e5)
	1	1.082e-1	1	194(194)	20	1	36(28)	1.9e3(2.1e5)
Cd2dat	2	2.462e-1	-	-	-	1	68(34)	1.9e3(4.7e4)
	2	1.553e-1	2	2926(1504)	430	1	66(34)	1.8e3(3.3e5)
	2	1.550e-1	1	1472(762)	120	1	66(34)	1.9e3(3.3e5)
H263	2	2.103e-4	-	-	-	1	9512(9506)	9.7e3(2.3e5)
	2	2.102e-4	2	19016(19012)	2.0e5	1	9512(9506)	9.5e3(2.3e5)
	2	2.101e-4	1	9512(9508)	2.4e4	1	9512(9506)	9.2e3(2.1e5)

[a] The solutions finding time (ms) branched and explored with two buffer measurements.

on $\gamma_{Sum''}$. Although, to branch and explore with $\gamma_{Sum''}$ and γ_{tile_n} has the potency to further reduce $\gamma_{Sum''}$, it dramatically increase the problem complexity. For instance, using the second specification of Wireless, we find $\gamma_{Sum''}$ to be 30 (instead of 49 in Table I) in 3hrs8mins. But all other experiments fail to find a solution in 4 hrs. The memory usages for solutions finding are in $47\sim163$Mb. Interestingly, the infeasible cases on both applications take the least memory usage and time, which is opposed to the model-checking method in [8].

B. Comparison with PAPS

Here, we intend to evaluate another performance metrics (i.e., buffer cost), which has not been well constrained in heuristics for MPSoCs. Since *SCP-OH* employs our *SCP* method in solutions finding and other heuristic methods [3, 5] have different problem definitions and assumptions as we have, a PAPS implemented in C++ is used as the reference method, which has no buffer constraints. A fixed ideal delay (no-contention aware) in inter-tile communication is used in the reference PAPS, as communication protocols was not considered. We have used three benchmarks, i.e., a Bipartite model [16], a Cd2dat rate converter [16], and a H263 decoder [3]. The architecture platform is a 2×2 mesh-based MPSoC.

Table II shows the experimental results. In *SCP*, the search tree is branched and explored with two types of buffer measurements. The solutions finding time is reasonable for such a medium problem size. Although a NoC communication protocol is modeled, SCP shows a significant buffer saving (6% of PAPS in the best case), and can provide higher (up to 64%) throughput. Here, the highest memory usage for SCP in solutions finding is up to 219Mb, caused by a high peak depth 440 in its search tree.

VII. CONCLUSION

In this paper, we have presented a constraint based scheduling framework for SDF streaming application on MPSoCs. Based on constraint programming techniques, the global scheduling for both processors computing and communication transactions has been achieved. Compared with traditional heuristics scheduling, our method has higher predictable application throughput and less buffer cost.

We know that the complexity of scheduling on MPSoCs with resource constraints is exponential to the problem size. To deal with this complexity, we plan to combine heuristics with our constraint based techniques, such as using heuristics to explore the search tree. Recently, the latest Gecode 3.1.0 starts to support parallel search in multiple threads, but the searching speed-up depends heavily on whether the search tree can be distributed to each thread efficiently [6]. Our experiments on multiple threads ($2\sim4$) have not got significant speed-up yet. It might be necessary to reshape the search tree for parallel search, which remains to be our future work as well.

ACKNOWLEDGEMENTS

We thank Dr. Johnny Öberg for useful discussions to improve the techniques in this paper. This research has been partially supported by the SYSMODEL project, which is an European ARTEMIS supported Initiative.

REFERENCES

[1] W. Wolf, "The future of multiprocessor systems-on-chips," in *Proceedings of the 41st annual Conference on Design Automation (DAC '04)*. New York, NY, USA: ACM, 2004, pp. 681–685.

[2] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. C-36, no. 1, pp. 24–35, January 1987.

[3] S. Stuijk, T. Basten, M. C. W. Geilen, and H. Corporaal, "Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs," in *Proceedings of the 44th annual Conference on Design Automation (DAC '07)*. New York, NY, USA: ACM, 2007, pp. 777–782.

[4] M. R. Garey and D. S. Johnson, *Computers and intractability : a guide to the theory of NP-completeness*. W. H. Freeman, January 1979.

[5] P. Eles, Z. Peng, P. Pop, and A. Doboli, "Scheduling with bus access optimization for distributed embedded systems," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 8, no. 5, pp. 472–491, 2000.

[6] Gecode, "Generic Constraint Development Environment," 2009, http://www.gecode.org/.

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

[8] M. Geilen, T. Basten, and S. Stuijk, "Minimising buffer requirements of synchronous dataflow graphs with model checking," in *DAC '05: Proceedings of the 42nd annual conference on Design automation*. New York, NY, USA: ACM, 2005, pp. 819–824.

[9] 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 *Proceedings of the 28th IEEE international real-time systems symposium (RTSS '08)*. Barcelona, Spain: IEEE Computer Society, November 2008.

[10] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*. New York, NY, USA: Marcel Dekker, Inc., 2000.

[11] L. Benini, M. Lombardi, M. Milano, and M. Ruggiero, "A constraint programming approach for allocation and scheduling on the cell broadband engine," in *CP '08: Proceedings of the 14th international conference on Principles and Practice of Constraint Programming*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 21–35.

[12] P.-E. Hladik, H. Cambazard, A.-M. Déplanche, and N. Jussien, "Solving a real-time allocation problem with constraint programming," *J. Syst. Softw.*, vol. 81, no. 1, pp. 132–149, 2008.

[13] J. Zhu, I. Sander, and A. Jantsch, "Buffer minimization of real-time streaming applications scheduling on hybrid CPU/FPGA architectures," in *Proceedings of Design Automation and Test in Europe (DATE '09)*, Nice, France, April 2009, pp. 1506–1511.

[14] M. Bekooij, O. Moreira, P. Poplavko, B. Mesman, M. Pastrnak, and J. V. Meerbergen, "Predictable embedded multiprocessor system design," in *Proceedings of Workshop on Software and Compilers for Embedded Systems (SCOPES), LNCS 3199*. Springer, 2004.

[15] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: Concepts, architectures, and implementations," *IEEE Des. Test*, vol. 22, no. 5, pp. 414–421, 2005.

[16] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Synthesis of embedded software from synchronous dataflow specifications," *Journal of VLSI Signal Processing Systems*, vol. 21, no. 2, pp. 151–166, June 1999.

[17] O. Moreira, F. Valente, and M. Bekooij, "Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor," in *Proceedings of the 7th ACM & IEEE International conference on Embedded Software (EMSOFT '07)*. New York, NY, USA: ACM, 2007, pp. 57–66.