Noname manuscript No.
(will be inserted by the editor)

Communication-Aware Heterogeneous Multiprocessor Mapping for Real-time Streaming Systems

Jing Lin · Andreas Gerstlauer · Brian L. Evans

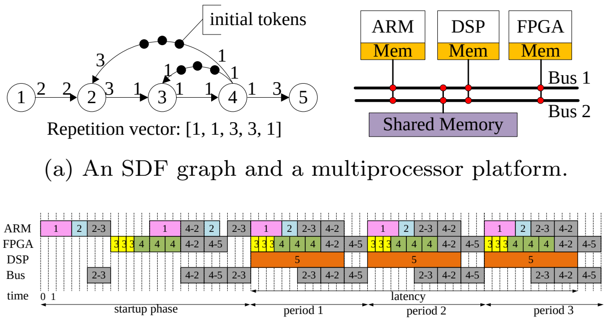
Received: date / Accepted: date

Abstract Real-time streaming signal processing systems typically desire high throughput and low latency. Many such systems can be modeled as synchronous data flow graphs. In this paper, we address the problem of multi-objective mapping of SDF graphs onto heterogeneous multiprocessor platforms, where we account for the overhead of bus-based inter-processor communication. The primary contributions include (1) an integer linear programming (ILP) model that globally optimizes throughput, latency and cost; (2) low-complexity two-stage heuristics based on a combination of an evolutionary algorithm with an ILP to generate either a single sub-optimal mapping solution or a Pareto front for design space optimization. In our simulations, the proposed heuristic shows up to 12x run-time efficiency compared to the global ILP while maintaining a 10^{-6} optimality gap in throughput.

Keywords Synchronous dataflow · Multiprocessor mapping · Inter-processor communication

1 Introduction

Real-time streaming signal processing applications are pushing embedded systems' capabilities of processing high-volume data streams with very low latency. Such stream-based systems are prevalent in a vast area of multimedia and communication applications. To achieve high performance, a stream processing system should have a highly-optimized execution path that maximizes throughput and minimizes latency in a balanced way.



(a) An SDF graph and a multiprocessor platform.

(b) Partitioning and scheduling output.

Fig. 1: An example of the SDF mapping problem.

With the current trend towards heteregoeneous multiprocessor systems-on-chips (MPSoCs), the mapping of applications onto such platforms is among the most critical tasks in the system design process. Many real-time streaming systems can be modeled by synchronous data flow (SDF) graphs [4]. Multiprocessor mapping of an SDF graph selects a number of processors from a library, binds each actor to a processor, allocates communication resources for inter-processor data exchange, and schedules actor executions and inter-processor communications (Fig. 1).

Traditionally, the synthesis of SDF models on multiprocessors has been focused on computational aspects, assuming negligible inter-processor communication (IPC) overhead. Earlier research efforts were devoted to SDF mapping on homogeneous multiprocessors [4,7]. More recent approaches have extended support to heterogeneous platforms [9]. In [15], the scheduling sub-problem

This research was supported by an equipment gift from Intel.

The work in this paper was presented in part at the 2011 IEEE International Conference on Acoustics, Speech and Signal Processing.

J. Lin, A. Gerstlauer, B. L. Evans
The University of Texas at Austin, USA
E-mail: linj@mail.utexas.edu, gerstl@ece.utexas.edu, bevans@ece.utexas.edu

was formulated in a constraint programming framework. The goal was to minimize memory requirements subject to throughput constraints. In [5], a graph-based approach was presented to globally tackle partitioning and scheduling for homogeneous SDF graphs, and several low complexity heuristic acceleration techniques were proposed. All these approaches focused on throughput and/or memory sizes as the main optimization goals, where no explicit guarantees were given on latency, which is among the most important real-time performance metrics.

As the number of processors on a single chip continue to increase, the communication architecture plays an increasingly important role in evaluating performance of the overall system. Neglecting communication overhead in system-level synthesis might lead to unacceptable deviation of the real performance from the predicted one, and to the violation of real-time constraints. To mitigate these effects, strategies have been developed to allocate and schedule communication resources for streaming applications. Most of these were restricted to simple application models, such as pipelined dataflow [10], acyclic homogeneous SDF [11,12], and message streaming [13]. Some of them target low-end communication architectures of shared memory and shared buses [10,12], while others take into account more complex Network-on-Chip (NoC) architectures [11,13].

Besides providing a single mapping decision for a single objective or cost function, a significant amount of work has been done on generating Pareto-optimal solution sets for design space exploration. Evolutionary algorithms (EA) have been proven to be effective in generating Pareto fronts in multi-objective optimization problems. [2] and [17] have shown application of EA in circuit synthesis and uniprocessor software synthesis from SDF graphs. [**?**] applied a combination of global EA and local search techniques to multiprocessor SDF scheduling for minimizing overall power consumption and memory cost. However, to the best of our knowledge, there are currently no approaches for mapping of SDF graphs onto general multiprocessor platforms across a variety of quality objectives, all while taking communication overhead into account.

In previous work [8], we have developed a first multi-objective optimization framework that is able to jointly optimize throughput, latency and system cost for SDF mapping onto heterogeneous multiprocessors. The inclusion of latency into the optimization allows us to give hard real-time guarantees on the task completion deadline. In this paper, we extend this approach to account for inter-processor communications (IPC). We target a bus-based communication architecture with both distributed and shared memory. We derive an integer linear programming (ILP) model that globally optimizes throughput, latency and system cost, and low-complexity EA-based heuristics to generate either a single mapping decision or a Pareto front for design space exploration. We thereby construct optimizations at varying complexity and runtime that can be applied with or without considering communication, depending, for example, on the stage of the design process or whether a design is purely computation dominated.

1.1 Synchronous Dataflow Model

Synchronous dataflow (SDF) is a directed graph modeling a set of application tasks and their data dependencies [4]. Each actor produces and consumes a fixed number of tokens per firing. The deterministic properties of SDF enable fully-static analysis and synthesis. In particular, for consistent SDF graphs, a periodic schedule and the minimum number of initial tokens on all edges can be determined by static analysis [4] (Fig. 1a).

In modeling point-to-point streaming applications, we assume that the SDF graph has a unique pair of source and sink actors (i.e. actors without any incoming edges, or any outgoing edges, respectively). Without loss of generality, we assume that both the source and sink actors fire once per iteration of the graph. Otherwise, pseudo actors with zero execution time could be added as source and/or sink.

1.2 Target Architecture and IPC Modeling

We target platform architectures that consist of a number of heterogeneous processors (e.g. general-purpose processors, FPGAs, DSPs or other processing elements) connected to a shared memory through multiple shared buses (Fig. 1a). Each processor has a local memory associated with it. For simplicity purpose we make the following assumptions about the target architecture:

1. The architecture is fully connected, i.e. the shared memory and all processors are accessible via any bus. Note that this assumption can be easily relaxed by including additional connectivity constraints in our ILP optimization model.
2. A bus is shared by multiple data transmissions in a time-division multiple access (TDMA) fashion.
3. The number of memory ports is no less than the number of buses that are connected according to the final mapping decision. This avoids memory access collisions even in the worst case when all buses are accessing the same memory.

4. The synchronization overhead (e.g. during connection setup and release) is negligible compared to the data transmission delay.
5. A processor is stalled (i.e. can not simultaneously compute) while communicating with the shared memory. In direct processor-to-processor communication, only one processor (typically the bus master) is stalled.

Inter-processor communication is required when two connected actors in an SDF are bound to different processors. Generally, tokens may be routed along a multi-hop path between the source and destination processors. However, in this work, we restrict the communication architecture to a fully-connected network with two simple routing policies: (1) direct connection between end processors and (2) two-hop connection via the shared memory.

To characterize the IPC behaviors, we augment an SDF graph by inserting a pair of send and receive actors on each edge [3] (Fig. 2). We assume that data is transmitted in token units. Therefore, each send/receive actor has an input/output rate of one and it is guaranteed that the augmented SDF graph does not violate the original balance equations. In other words, the augmented SDF is still a consistent graph with the same repetition vector of firing rates for the computation actors. In the two-hop IPC scenario, the send/receive actors model the process of writing/reading a token to/from the shared memory, while the source-to-send, send-to-receive and receive-to-destination edges represent the local buffer on the source processor, a piece of the shared memory, and the local buffer on the destination processor, respectively. Direct processor-to-processor connections and processor-internal communication can be considered as two special cases of this generic IPC model, where either one (for direct connections) or both (for communication within processors) of the send and receive actors are inactive (execute in zero delay).

The send and receive actors are distinguished from computation actors in resource occupancy. While being executed, a computation actor only keeps a single processor busy. By contrast, an active send/receive actor occupies a bus while also possibly stalling the connected processor communicating on the bus.

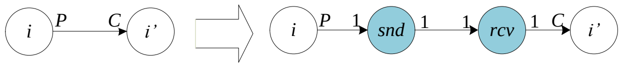


Fig. 2: An edge in an SDF graph (left) is augmented by inserting a pair of send and receive actors (right).

1.3 Problem Definition

Given an SDF model for a real-time streaming system, we assume that static analysis has been performed to provide a consistent repetition vector and the number of initial tokens on all edges. In addition, the execution profile on a multiprocessor platform is provided in the form of processor-dependent worst-case execution times of all computation actors. We aim to partition the SDF graph onto various computation and communication resources, and construct fully-static schedules on processors and buses. We assume that each actor is statically bound to a unique processor and/or bus, and actors mapped to the same processor or bus must be executed sequentially. The schedules are represented by a set of finite-length time-indexed sequences, each initialized with a startup phase followed by one period of the stable phase (Fig. 1b). The period is defined as a duration of time over which the SDF graph completes one iteration according to the repetition vector and returns to an initial state. The inverse of the period is the throughput. The latency is defined as the time interval between the start of the source and the end of the sink within one graph iteration.

Based on worst-case actor execution time assumptions, the fully-static mapping provides hard, worst-case guarantees on real-time performance and system cost. In reality, due to unpredictable events such as interrupts and malfunctions, a strictly static schedule may become infeasible at run-time, and dynamic scheduling needs to be invoked to increase system flexibility. Despite of this, static analysis results can still be used to provide worst-case guarantees and to provide a reference point for the run-time optimizer to reduce the dynamic scheduling overhead. For example, the partial execution ordering indicated by the static scheduler can be used to define additional, artificial actor dependencies to be maintained at run-time. Such a static-order schedule will allow the dynamic scheduler to preserve worst-case guarantees while determining exact starting times of actors at run time, e.g. to recover any execution time slack for reallocation to other non-real-time tasks. In this sense, the problem is related to classical real-time analysis of aperiodic, dependent tasks [**?**]. In contrast to homogenous task-graph models in classical formulations, however, SDF graphs allow us to cover a wider range of application scenarios.

In some cases it is advantageous to unfold a graph and schedule multiple iterations of the graph together in order to exploit inter-iteration parallelism more effectively. Unfolding is also useful in grouping small chunks of IPCs together to mitigate the synchronization overhead incurred in practice. However, a disadvantage of

Parameter	Definition
O_k	The number of initial tokens on edge k.
$P_k(C_k)$	The number of tokens produced (consumed) on edge k.
N_i	The number of executions of actor i in one iteration of the graph.
D_{ij}	Execution time of actor i on processor j.
BW_l	Bandwidth (in bits/second) of bus l.
$Size_k$	Token size (in bits) on edge k.
DC_{kl}	Time of transmitting a token on edge k using bus l.
$Cost_j$	Cost (user-defined measure of area, price, etc.) of processor j.

Table 1: Input parameters of the ILP model.

unfolding is the increase in design complexity at compile time and of memory consumption at run time. Due to complexity concerns, our algorithms do not automatically unfold the graph. However, unfolding can be performed as a pre-processing step where the unfolded SDF can be represented as another (larger) SDF at the input of our flow, if the performance metrics of mapping the original graph are not satisfactory.

2 A Global ILP Model

The global optimization of multiprocessor SDF partitioning and scheduling is NP-hard [4]. In this section, we formalize the problem as an integer linear programming (ILP) model. Despite of its intractability in practical implementation, the ILP formulation is useful to explore the problem structure, which provides cues for low-complexity heuristics as described later.

2.1 Parameters and Variables

The inputs to the ILP include an SDF graph (whose static behavior has been analyzed at compile time), and the computation and communication profile of the modeled application on the selected target platform. In the following, we describe the input parameters to our ILP model, which are summarized in Table 1.

Let $\mathcal{I}$ and $\mathcal{K}$ be the set of computation actors and edges, respectively, in a given SDF graph. For $i \in \mathcal{I}$, $k \in \mathcal{K}$, the structure and static behavior of an SDF graph can be represented by: (1) the number of initial tokens O_k and the number of tokens P_k and C_k produced and consumed, respectively, on edge k; and (2) the repetition vector of the original SDF, whose i-th element is N_i.

Let $\mathcal{J}$ define the set of processors, and $\mathcal{L}$ the set of buses available on a target platform. For $j \in \mathcal{J}$, the computation profile of an SDF graph on the platform is given by the execution time of actor i on processor j, denoted by D_{ij}. The communication delay for transmitting a single token over a link between two memory locations depends on the bandwidth (in bits/second) of the allocated bus, BW_l ($l \in \mathcal{L}$), and on the token size (in bits) on edge k, $Size_k$. The token sizes are generally indicated by the modeled application, and in reality might be adjusted to the bit-width of the bus interface. Without loss of generality, we assume uniform bit-width among all bus interfaces. Therefore, the delay it takes to transmit a token on edge k using bus l is $DC_{kl} \triangleq Size_k/BW_l$. Furthermore, each processor is associated with certain user-defined measure of cost (e.g. area, price, etc.), denoted as $Cost_j$.

Given the input parameters, the procedure of partitioning the SDF graph onto various computation and communication resources consists of: (1) binding each computation actor to a unique processor; (2) deciding the IPC routing policy (either direct connection between end processors or via the shared memory); and (3) allocating each of the send and receive actors to a unique bus. The following decision variables are able to completely capture the partitioning:

- a_{ij}: Binary indicator of whether actor i is bound to processor j;
- ch_k: Binary indicator of whether the IPC on edge k (if any) is routed through the shared memory;
- bs_{kl}, br_{kl}: Binary indicator of whether the send or receive actor on edge k is allocated to bus l.

To facilitate our discussion, we define auxiliary variables $\{ipc_k\}$ that distinguish edges on which IPC is required from others. They are related to $\{a_{ij}\}$ via $ipc_k = \sum_j a_{src(k)j} \cdot a_{dst(k)j}$, where $src(k)$ and $dst(k)$ are defined as the indices of the source and destination actors of edge k, respectively.

The schedules for the computation, send and receive actors can be described by three sets of time-indexed counting processes starting from 0 with unit increments, namely $s_i(t)$, $ss_k(t)$ and $sr_k(t)$, respectively. They can be interpreted as how many times the corresponding item has been fired at or before time t. For notation simplicity, we define corresponding variables to count the number of ended executions of the send/receive actors, i.e. $es_k(t)$ and $er_k(t)$. They are related to the other variables through

$$\begin{aligned} es_k(t) =& ipc_k \sum_l bs_{kl} \cdot ss_k(t - DC_{kl}) \\ &+ (1 - ipc_k) ss_k(t), \end{aligned} \tag{1}$$

$$\begin{aligned} er_k(t) =& ipc_k \cdot ch_k \sum_l br_{kl} \cdot sr_k(t - DC_{kl}) \\ &+ (1 - ipc_k \cdot ch_k) sr_k(t). \end{aligned} \tag{2}$$

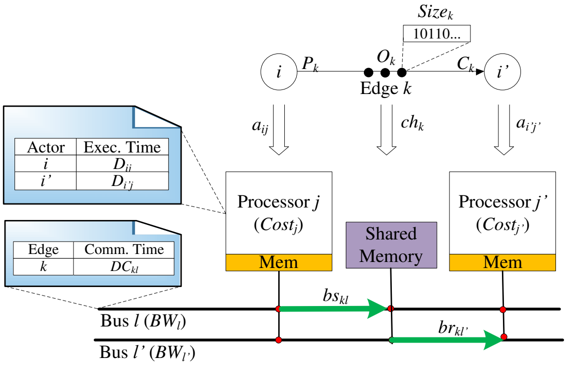


Fig. 3: Parameters and variables of the global ILP.

This is equivalent to saying that a send/receive actor either has a bus-dependent delay, or is inactive (with delay 0). The placement of the variable ch_k thereby determines whether the send or receive processor is stalled (i.e. is the bus master) during a direct processor-to-processor communication. Furthermore, we introduce an additional sequence $start(t)$ that consists of a single impulse indicating the beginning of the first period, i.e. that marks the transition between the startup and the periodic phase. For clarity purposes, we summarize some fundamental parameters and variables in Fig. 3.

2.2 Constraints

Constraints need to be imposed in order to satisfy the semantics of the augmented SDF graph and the restrictions imposed by the underlying target architecture. Note that an ILP without communication considerations can be constructed by removing all decision variables, terms and constraints related to communication actors, as shown in [8].

Static partitioning. Each actor is statically bound to a unique processor, and each send or receive actor to a unique bus. This requires that $\forall i, k$,

$$\sum_j a_{ij} = 1, \sum_l bs_{kl} = 1, and \sum_l br_{kl} = 1. \tag{3}$$

Counting processes. The following three sequences are unit incremental counting processes starting from 0:

$$\begin{aligned} 0 \le s_i(t) - s_i(t-1) \le 1, \quad & s_i(0) = 0, \\ 0 \le ss_k(t) - ss_k(t-1) \le 1, \quad & ss_k(0) = 0, \\ 0 \le sr_k(t) - sr_k(t-1) \le 1, \quad & sr_k(0) = 0. \end{aligned} \tag{4}$$

Precedence. Enough tokens have to be accumulated on an input edges before an actor is allowed to fire. For any edge k, let $i_1 = src(k)$ and $i_2 = dst(k)$, and

$$\begin{aligned} er_k(t) - s_{i_2}(t) C_k + O_k \ge 0, \\ es_k(t) - sr_k(t) \ge 0, \\ \sum_j a_{i_1 j} \cdot s_{i_1}(t - D_{i_1 j}) P_k - ss_k(t) \ge 0. \end{aligned} \tag{5}$$

Resource sharing. Items sharing the same resource (processor or bus) need to be executed sequentially. A processor can be busy executing a computation actor, or stalled during an IPC transmission, giving

$$\begin{aligned} & \sum_i [s_i(t) - s_i(t - D_{ij})] \cdot a_{ij} \\ & + \sum_k [ss_k(t) - es_k(t)] \cdot a_{src(k),j} \\ & + \sum_k [sr_k(t) - er_k(t)] \cdot a_{dst(k),j} \le 1. \end{aligned} \tag{6}$$

IPCs are scheduled on a bus by time-division multiplexing, leading to

$$\begin{aligned} & \sum_k [ss_k(t) - es_k(t)] \cdot bs_{kl} \\ & + \sum_k [sr_k(t) - er_k(t)] \cdot br_{kl} \le 1. \end{aligned} \tag{7}$$

Periodicity. The schedule constructed initializes with a startup phase before it reaches a single execution of the periodic pattern, the beginning of which is marked by the single impulse in $start(t)$,

$$\sum_t start(t) = 1. \tag{8}$$

The period corresponds to one iteration of the entire SDF graph. Let w_i, ws_k and wr_k denote the number of time slots that a computation, send or receive actor has been executed at or before time t, respectively. They are related to the decision variables as follows:

$$\begin{aligned} w_i(t) &= \sum_{\tau \le t} [s_i(\tau) - s_i(\tau - D_{ij})] \cdot a_{ij}, \\ ws_k(t) &= \sum_{\tau \le t} [ss_k(\tau) - es_k(\tau)], \\ wr_k(t) &= \sum_{\tau \le t} [sr_k(\tau) - er_k(\tau)]. \end{aligned} \tag{9}$$

The number of time slots that each actor executes during one period is therefore:

$$\begin{aligned} r_i &= w_i(T_{max}) - \sum_t w_i(t) \cdot start(t), \\ rs_k &= ws_k(T_{max}) - \sum_t ws_k(t) \cdot start(t), \\ rr_k &= wr_k(T_{max}) - \sum_t wr_k(t) \cdot start(t), \end{aligned} \tag{10}$$

where the periodicity requires that each actor completes a specific number of iterations (indicated by the repetition vector) within the period, i.e.

$$r_i = N_i \sum_j a_{ij} D_{ij}, \tag{11}$$

$$rs_k = N_{src(k)} P_k \cdot ipc_k \sum_l bs_{kl} \cdot DC_{kl}, \tag{12}$$

$$rr_k = N_{dst(k)} C_k \cdot ipc_k \cdot ch_k \sum_l br_{kl} \cdot DC_{kl}. \tag{13}$$

2.3 Objectives

To jointly optimize multiple quantities in an ILP setting, one common way is to define the objective function as a linear combination of single objectives:

$$min \quad \{\lambda_1 \cdot Period + \lambda_2 \cdot Cost + \lambda_3 \cdot Latency\}, \tag{14}$$

where λ_1, λ_2 and λ_3 are non-negative weights. The period of the schedule can be determined as

$$Period = T_{max} - \sum_t t \cdot start(t). \tag{15}$$

The processor cost is defined as

$$Cost = \sum_j I\{\sum_i a_{ij} > 0\} \cdot Cost_j, \tag{16}$$

where $I\{\sum_i a_{ij} > 0\}$ is the indicator function of whether processor j is allocated. Note that (16) can be readily extended to also include bus costs.

By the assumption of a unique pair of source and sink actors in an SDF graph (see Section 1.1), the latency (in the periodic phase) is defined as the interval between the time the source actor starts a particular iteration and the sink actor ends the same iteration. Let $ts_i^{(k)}$ be the time when actor i starts its k-th iteration, and define $u_i(t)$ as the sum of the time stamps when actor i starts execution up to time t:

$$u_i(t) \triangleq \sum_k ts_i^{(k)} I\{ts_i^{(k)} \leq t\} = \sum_{\tau \leq t} \tau[s_i(\tau) - s_i(\tau - 1)].$$

Suppose actor i executes once per period, as is the case for the source and sink actors, its starting time in the period then becomes $U_i \triangleq u_i(T_{max}) - \sum_t u_i(t) start(t)$. Hence, the latency can be determined from one period of the schedule as:

$$\begin{aligned} Latency = & U_I - U_1 + \sum_j a_{Ij} D_{Ij} \\ & + [s_1(T_{max}) - s_I(T_{max})] \cdot period, \end{aligned} \tag{17}$$

where the first line captures the difference in time between the end of the sink (indexed by I) and the start of the source actor (indexed by 1) within the period, and the second line takes into account the fact that the source and sink actors might be in different iterations.

2.4 Linearization

To eliminate the nonlinearity present in the model above, we apply conversions introduced in [6]. Let M be a tight upper bound of an integer variable x. We replace each product between a binary variable b and x with an auxiliary variable y, and add the following linear constraints:

$$y \geq x - M(1 - b), y \leq Mb, x \geq y, \tag{18}$$

We substitute the indicator function $I\{x > 0\}$ (x is a non-negative integer variable here) by a binary variable b, and put two constraints on b:

$$x \leq Mb, x \geq b. \tag{19}$$

The resultant ILP is equivalent to the nonlinear optimization model described above.

3 Heuristics

Despite its completeness and optimality, the global ILP model is NP-hard. Hence, efficient heuristics are desired to combat the exponentially increasing complexity.

In practice there are plenty of design scenarios where optimizing the throughput and the cost is prioritized over minimizing latency. We can observe that without explicit constraints on memory requirements, the partition of an SDF graph onto various resources uniquely determines not only the cost but also an upper bound of the throughput (or equivalently, a lower bound of the period), regardless of the scheduling. Given a partition, the period has a lower bound determined by the critical processor or bus, i.e. the one occupied for the longest duration in a period:

$$Period \geq Period_{min} = max_{j,l}\{Ocp_j, Ocp_l\}, \tag{20}$$

where Ocp_j and Ocp_l are the total duration that processor j or bus l is occupied during a period. Crucially, Ocp_j and Ocp_l only depend on partitioning:

$$\begin{aligned} Ocp_j = & \sum_i a_{ij} D_{ij} \\ & + \sum_k ipc_k \cdot a_{src(k)j} \sum_l bs_{kl} DS_{kl} \\ & + \sum_k ipc_k \cdot ch_k \cdot a_{dst(k)j} \sum_l br_{kl} DR_{kl}, \end{aligned} \tag{21}$$

$$\begin{aligned} Ocp_l = & \sum_k ipc_k \cdot bs_{kl} DS_{kl} \\ & + \sum_k ipc_k \cdot ch_k \cdot br_{kl} DR_{kl}. \end{aligned} \tag{22}$$

We empirically verify that if a partition is computationally dominated (i.e. $Ocp_j > Ocp_l, \forall j, l$), the lower bound in (20) is almost always achievable. In other words, there exists a valid schedule with a period of $Period_{min}$.

This observation leads to a two-stage decision procedure based on decomposition of partitioning and scheduling sub-problems. By optimizing the throughput jointly with the cost over the partitioning variables, we make a first-stage decision on partitioning, which is one or possibly multiple optimal solutions of $\{a_{ij}\}$, $\{bs_{kl}\}$, $\{br_{kl}\}$ and $\{ch_k\}$. Then, in the second stage, for each optimal partition a schedule that minimizes the latency can be found and the minimum achievable latency for different partitions are compared to each other to decide the best mappings.

3.1 Two-Stage ILP

The two-stage decision procedure readily converts into a heuristic of decomposing the global ILP into two sub-ILPs: a partitioning ILP followed by a scheduling ILP. Note that due to the exponential nature of ILPs, the complexity of this two-stage ILP combination is smaller than that of the original global ILP. The partitioning ILP is a simple formulation that minimizes the objective function $\lambda_1 \cdot Period_{min} + \lambda_2 \cdot Cost$ (where $\lambda_1, \lambda_2 \geq 0$) under the constraints of (3) and (22). The scheduling ILP is extracted from the global ILP (see Appendix A). With the partitioning variables fixed and a known period, it has a significantly reduced complexity.

The partitioning sub-problem is itself a two-objective optimization. Since minimizing throughput and cost happen to be conflicting goals, there is no longer a single optimal solution but rather a set of possible solutions of equivalent quality, i.e. a Pareto front. Furthermore, there may exist different partitions that achieve identical throughput and cost but different latencies (i.e. points on the Pareto front corresponding to multiple solutions). Similarly, partitions that are sub-optimal in throughput and/or cost may allow for lower latencies compared to those on the throughput/cost Pareto front. Based on these concerns, we would like to (1) find all partitions that achieve Pareto-optimal throughput/cost and feed them into the scheduling stage in order to pick the ones with the lowest latencies; and (2) find those partitions that are off the throughput/cost Pareto front but have lower latencies.

For (1), it is possible in the ILP settings to generate the Pareto front by solving a number of partitioning ILPs corresponding to various combinations of λ_1 and λ_2. Unless (λ_1, λ_2) are restricted to a small set of discrete values, however, the amount of ILPs to be solved is infinite. Therefore efficient searching algorithms are desired to find the set of Pareto-optimal solutions in the partitioning sub-problem. To address (2) within the two-stage optimization framework, a closed-loop searching algorithm is required for decision feedback from the scheduling stage to aid the searching in the partitioning stage.

3.2 Hierarchical MOEA-Driven Heuristics

An efficient technique of approximating the Pareto front and finding the solution pool for each point on the front are so-called multi-objective evolutionary algorithms (MOEA) [18]. The Strength Pareto Evolutionary Algorithm II (SPEA-II) [16] is one of the MOEAs with the best overall performance. As in other MOEAs, each possible solution is encoded into a chromosome. In addition to a small sized population of chromosomes, SPEA-II maintains a fixed sized archive that records the nondominated individuals found so far. The fitness is assigned for each individual based on how many solutions it dominates, it is dominated by, and a nearest neighbor density. Such fine-grained fitness assignment, together with an archive truncation operator, effectively guides the population to evolve towards a well-spread Pareto front.

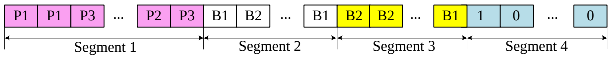


Fig. 4: A chromosome with four segments encoding actor to processor allocation, send actor to bus allocation, receive actor to bus allocation, and edge to shared memory allocation.

Applying SPEA-II to the partitioning sub-problem gives rise to a hierarchical, two-stage open-loop algorithm to generate a single mapping decision. In order to apply MOEA, we encode the partition-related variables $\{a_{ij}\}$, $\{bs_{kl}\}$, $\{br_{kl}\}$ and $\{ch_k\}$ into a chromosome (Fig. 4). The objectives of $Period$ and $Cost$ are evaluated for each chromosome by (20) and (16). To generate a single mapping, we first run MOEA to generate a two-dimensional (2D) Pareto front. At the end of the MOEA, we pick a point on the front, and feed all partitions achieving the same optimal throughput and cost to the scheduling ILP. The resulting single mapping decision falls onto the partition-schedule combination that achieves the minimum latency. This process is illustrated in Fig. 5a. By picking multiple points on the 2D Pareto front and feeding them all to the scheduling ILP, this method can be extended to generate multiple mapping decisions.

The MOEA-driven algorithm described above generally performs better than the two-stage ILP, due to its efficiency in finding multiple optimal partitions to feed to the scheduling sub-problem. However, both two-stage open-loop procedures (either MOEA-driven or ILP-driven) place strictly higher priority on through-

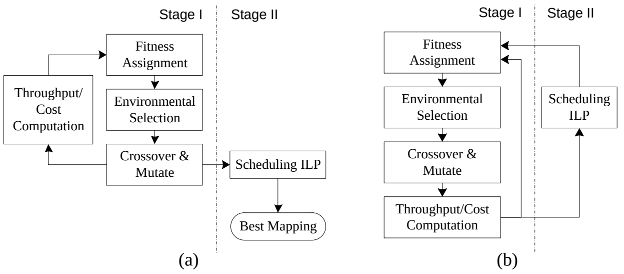


Fig. 5: Two MOEA-driven heuristics to generate (a) a single mapping decision, and (b) a 3D Pareto front.

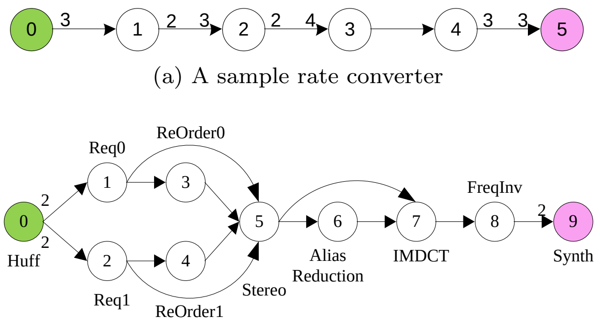
put and cost than on latency, and hence are blind to the mapping solutions that have worse throughput and cost but shorter latency. In other words, there may exist partitions that are not on the 2D Pareto front in the first stage but allow for smaller latency than those 2D Pareto-optimal points.

To balance the optimality of all three objectives, we extend the open-loop MOEA-driven algorithm to a hierarchical closed-loop method for generating a three-dimensional (3D) Pareto front. In this case, the fitness of a chromosome is based on a triplet of *Period*, *Cost* and *Latency*. The scheduling ILP is not only invoked for points on the Pareto front at the end of the MOEA, but for all partitions during every generation of the MOEA. For each chromosome, after computing its cost and minimum period, a scheduling ILP is solved and the optimal latency is attached to it as a third fitness value. This algorithm is described in Fig. 5b.

4 Experimental Results

We have evaluated our ILP model and the heuristic approaches on a variety of random homogeneous SDF (HSDF) graphs and two realistic SDF examples of the sample rate converter (Fig. 6a) and the MP3 decoder (Fig. 6b). We programmed the ILP models using CPLEX Concert Technology for C++, with the optimality gap controlled below 10^{-6}. The SPEA-II was implemented using the MOGALib Genetic algorithm framework. All experiments were run on a remote workstation with 8 quad-core 2.39 GHz AMD Opteron(tm) processors and 33 GB of shared memory.

We studied the optimality and run time complexity tradeoffs among the global ILP and the heuristics. We applied the global ILP (1ILP), the two-stage ILPs (2ILP) and the open-loop MOEA-driven heuristic (OL-EA) to mapping a set of HSDF graphs onto a 3-processor platform, assuming negligible IPC. The HSDF graphs were randomly generated by SDF3 [?], and include both cyclic and acyclic graphs with 5 to 15 actors, all weakly connected. Token rates, degrees of actors and execution times are randomly created within minimum and maximum bounds and with specified average and variance.



(a) A sample rate converter

(b) A simplified MP3 decoder

Fig. 6: The SDF models for two realistic examples.

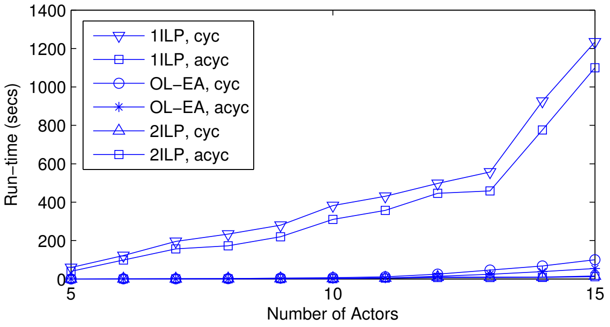


Fig. 7: Average run-time comparison for random SDF graphs mapped to a three-processor platform with randomized execution profiles without IPC overhead.

The optimized throughput, cost and latency metrics for four sample HSDF graphs are compared in Table 2. The global ILP applies the weights of $(0.8, 0.1, 0.1)$ to throughput, cost and latency, respectively, while the first stage of the two-stage ILP uses $(0.89, 0.11)$ as the weights of throughput and cost[1]. The optimality guarantee of the OL-EA is demonstrated by the fact that the solution of the global ILP falls on the 3D Pareto front produced by the OL-EA, except for the acyclic 10-actor

I	Type	(Period, Cost, Latency)		
		1ILP	2ILP	OL-EA
5	acyc	(11,30,11)	(11,30,14)	(11,30,11) (7,40,13) (6,60,14)
	cyc	(11,30,11)	(11,30,11)	(34,10,34) (23,20,23) (11,30,11) (7,40,13) (6,60,14)
10	acyc	(25,30,21)	(25,30,38)	(25,30,25) (17,40,24) (15,50,33) (12,60,26)
	cyc	(25,30,21)	(25,30,31)	(25,30,21) (18,40,31) (15,50,23) (12,60,27)

Table 2: Optimality comparison for random HSDF graphs with *I* actors mapped to a 3-processor platform with randomized execution profiles, without IPC overhead.

[1] The ratio of weights between throughput and cost is kept approximately the same as in the global ILP.

Parameter	Value
Population Size	30
Archive Size	20
Termination Condition	After 50 generations[2]
Crossover Probablity	0.9
Mutation Probability	0.1
Crossover Method	Uniform Crossover
Selection Method	Roulette Wheel

Table 3: Parameters for SPEA-II.

sample graph. The 2ILP method results in the same throughput and cost as the global ILP, while getting a slightly longer latency in some cases. This is because in the partitioning stage, the ILP solver terminates after finding an optimal solution, which, according to our discussion in Section 3.1, does not necessarily allow for the minimum latency compared to other unfound partitions on the throughput/cost Pareto front.

In generating a single mapping decision, the heuristics also significantly reduce the run time compared to the global ILP. In our experiments, for a 10-actor acyclic graph, it can take up to 5 days to solve a global ILP with all three objectives being optimized (i.e. all weights are non-zero). For run time comparison, we use throughput as the single objective in the global ILP and in the first stage of the 2ILP. Since adding other objectives would significantly increase the complexity of the global ILP and slightly increase that of the 2ILP, the corresponding curves should be understood as a lower bound of the run time. As shown in Fig. 7, the global ILP has much poorer scalability compared to both heuristics. Furthermore, in return for the the improved optimality in latency, the OL-EA method runs slightly slower than the 2ILP heuristic.

To show the capability of generating 2-dimensional (2D) or 3-dimensional (3D) Pareto fronts, we applied the MOEA-driven heuristics in Fig. 5 to the mapping of the sample rate converter and MP3 decoder to a 3-processor platform with a single shared bus. We applied artificial execution profiles both with and without communication overhead, where IPC delays were assumed to be fixed at one time unit per token. For the particular examples, the parameter settings of the MOEA are listed in Table 3. As suggested by [1], we use a population size of approximately $1.5 \log_2 N$, where N is the number of possible permutations of the chromosome. Run times for generating the 2D and 3D Pareto fronts are listed in Table 4.

Example	Heuristic	With IPC?	Run Time
Converter	OL-EA	Yes	231 seconds
	CL-EA	Yes	278 minutes
	CL-EA	No	17 minutes
MP3	CL-EA	Yes	41 hours
	CL-EA	No	23 hours

Table 4: Run times of generating 2D or 3D Pareto fronts for the sample rate converter (Converter) and the MP3 decoder (MP3) by the open-loop MOEA-driven heuristic (OL-EA) and the closed-loop MOEA-driven heuristic (CL-EA), with or without IPC overhead.

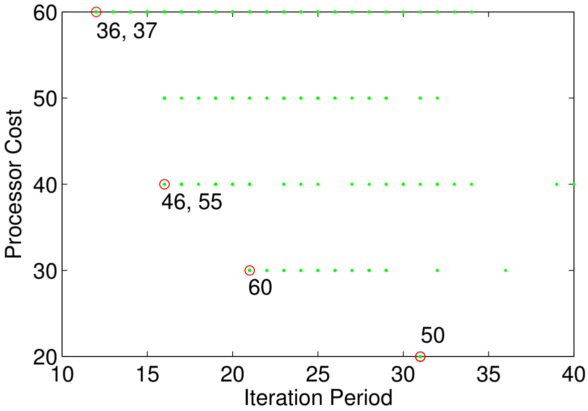
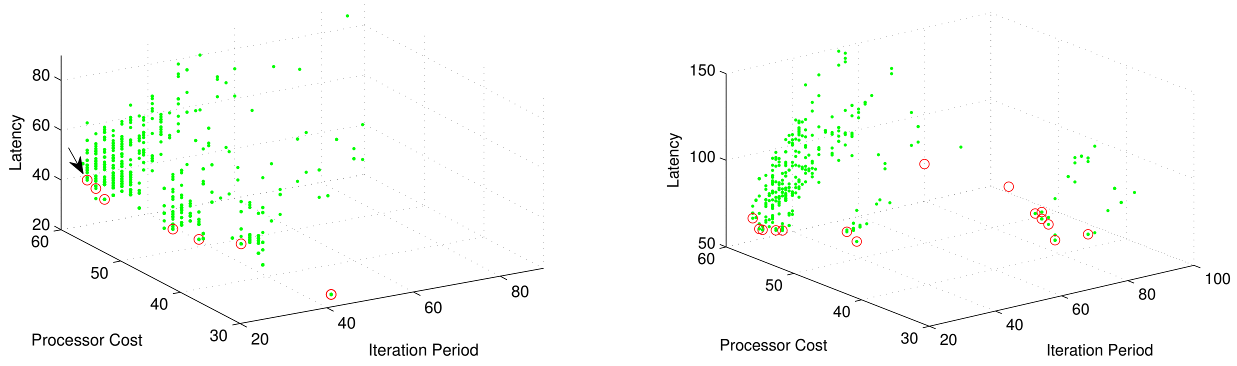


Fig. 8: 2D Pareto front for the sample rate converter, annotated with the latencies achieved by the corresponding partitions.

Fig. 8 depicts the 2D Pareto front generated by the open-loop MOEA-driven heuristic for the sample rate converter. Some of the points on the Pareto front correspond to multiple partitions with identical throughput and processor cost but with different minimum latencies. In particular, the red circle on the upper left corner represents two partitions with the same throughput of 1/12 and processor cost of 60. However the minimum latency achievable is 36 for the first partition, and 37 for the second. In comparison, we ran the two-stage ILPs with throughput as the single objective in the first stage. The two-stage ILP took 187 seconds to complete, but only produced a single solution with a latency of 37, since it was able to only find one of the two possible partitions in its first stage.

Fig. 9 and Fig. 10 show the 3D Pareto fronts (red circles) generated by the closed-loop MOEA-driven heuristic, along with the convergence history (green dots). In particular, the solutions pointed out by the arrows in Fig. 9a, $(26, 60, 35)$ and $(32, 40, 46)$, are the optimal solutions as generated by the global ILP with $(\lambda_1, \lambda_2, \lambda_3) = (0.8, 0, 0.2)$ and $(\lambda_1, \lambda_2, \lambda_3) = (0.8, 0.1, 0.1)$, which implies closeness of the generated solution set to the optimal Pareto front. The global ILP in this case took 1004 seconds (with $\lambda_2 = 0$) and about 5 days (with $\lambda_2 = 0.1$) to generate a single solution. In both examples, note that communication overhead can have a significant in-

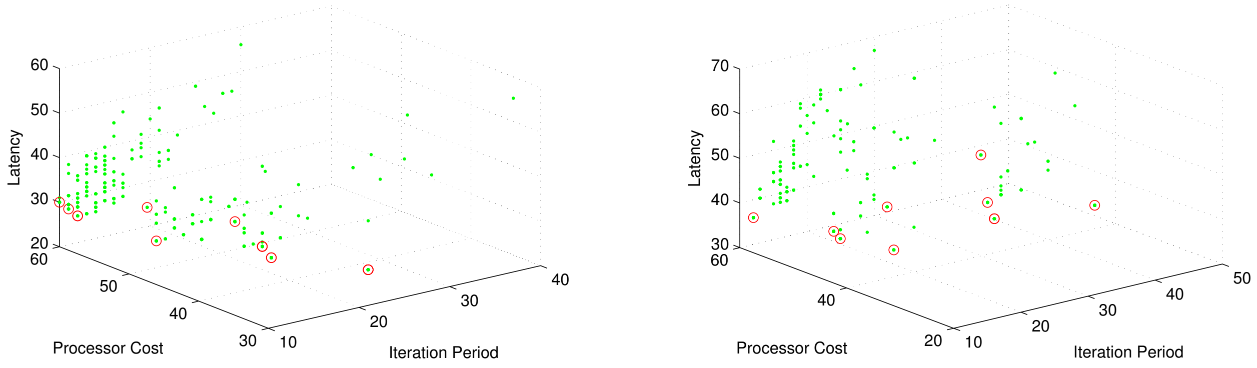
[2] In all tested cases, the EA converges after 50 generations.



(a) Without IPC

(b) With IPC

Fig. 9: Convergence to the 3D Pareto front for the MP3 decoder.



(a) Without IPC

(b) With IPC

Fig. 10: Convergence to the 3D Pareto front for the sample rate converter.

fluence on the overall performance (throughput and latency) of the set of Pareto-optimal solutions.

5 Summary and Conclusions

This paper derives communication-aware approaches to optimize multiprocessor mapping of real-time streaming systems for throughput, latency and processor cost. We propose both an optimal ILP model and two heuristics to improve the run-time efficiency without compromising throughput optimality. A two-stage ILP heuristic can generate a single solution at significantly reduced runtime. Hierarchical heuristics that integrate evolutionary algorithms with ILP solvers in an open-loop or closed-loop fashion can generate 2D or 3D Pareto tradeoff curves for multi-dimensional design space exploration. The various mapping approaches support different levels of optimality, complexity and runtime in variants both with and without communication considerations. This enables efficient design methodologies in which different search strategies can be selectively employed at different stages of the design process and for different characteristics of the application or target architecture.

A Scheduling ILP

When given a partition, the execution time of conventional SDF actors as well as the communication delays of the send and receive actors become fixed parameters. For conciseness, we slightly abuse the notation by defining parameters

$$\begin{aligned} D_i &= \sum_j a^*_{ij} D_{ij}, \\ DS_k &= ipc^*_k \cdot \sum_l bs^*_{kl} DC_{kl}, \\ DR_k &= ipc^*_k \cdot ch^*_k \cdot \sum_l br^*_{kl} DC_{kl}, \end{aligned} \qquad (23)$$

where v^* denotes a specific value of a variable v. Let us define the index set $\mathcal{T}_p = [T_{max} - Period^*, T_{max}]$, in which $Period^*$ is the minimum period determined by the partition. Also denote $\mathcal{K}_s \triangleq \{k | ipc^*_k = 1\}$ and $\mathcal{K}_r \triangleq \{k | ipc^*_k \cdot ch^*_k = 1\}$, i.e. the set of edges with an active send or an active receive actor, respectively.

The scheduling sub-problem can then be formalized as an ILP that minimizes

$$\begin{aligned} Latency = &\sum_{t \in \mathcal{T}_p} t[s_I(t) - s_I(t-1)] + D_I \\ &- \sum_{t \in \mathcal{T}_p} t[s_1(t) - s_1(t-1)] \\ &+ (s_1(T_{max}) - s_I(T_{max}))Period^*, \end{aligned} \tag{24}$$

subject to

Precedence

$$\begin{aligned} s_{src(k)}(t - D_i)P_k - ss_k(t) \geq 0, \\ ss_k(t - DS_k) - sr_k(t) \geq 0, \\ sr_k(t - DR_k) - s_{dst(k)}(t)C_k + O_k \geq 0, \forall k. \end{aligned} \tag{25}$$

Resource sharing

$$\begin{aligned} &\sum_i [s_i(t) - s_i(t - D_i)] \\ &+ \sum_{k \in \mathcal{K}_s} [ss_k(t) - ss_k(t - DS_k)]a^*_{src(k)j} \\ &+ \sum_{k \in \mathcal{K}_r} [sr_k(t) - sr_k(t - DR_k)]a^*_{dst(k)j} \leq 1, \forall j; \end{aligned} \tag{26}$$

$$\begin{aligned} &\sum_{k \in \mathcal{K}_s} [ss_k(t) - ss_k(t - DS_k)]bs^*_{kl} \\ &+ \sum_{k \in \mathcal{K}_r} [sr_k(t) - sr_k(t - DR_k)]br^*_{kl} \leq 1, \forall l. \end{aligned} \tag{27}$$

Periodicity

$$\begin{aligned} &\forall i, t \in [T_{max} - D_i, T_{max}] \\ &\quad s_i(t) - s_i(t - Period^*) = N_i; \end{aligned} \tag{28}$$

$$\begin{aligned} &\forall k, t \in [T_{max} - DS_k, T_{max}], \\ &\quad ss_k(t) - ss_k(t - Period^*) = N_{src(k)}P_k; \end{aligned} \tag{29}$$

$$\begin{aligned} &\forall k, t \in [T_{max} - DR_k, T_{max}], \\ &\quad sr_k(t) - sr_k(t - Period^*) = N_{dst(k)}C_k. \end{aligned} \tag{30}$$

References

1. Alander, J.: On optimal population size of genetic algorithms. In: Proc. IEEE Int. Conf. on Comp. Sys. and Software Eng., pp. 65–70 (2002)
2. Aslam, N., Arslan, T., Erdogan, A.: Algorithmic level design space exploration tool for creation of highly optimized synthesizable circuits. In: Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, vol. 2 (2007)
3. Bambha, N., Kianzad, V., Khandelia, M., Bhattacharyya, S.: Intermediate representations for design automation of multiprocessor dsp systems. Design Automation for Embedded Systems **7**(4), 307–323 (2002)
4. Bhattacharyya, S., Murthy, P., Lee, E.: Software synthesis from dataflow graphs. Springer (1996)
5. Bonfietti, A., Benini, L., Lombardi, M., Milano, M.: An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms. In: Proc. IEEE Conf. on Design, Automation and Test in Europe, pp. 897–902 (2010)
6. Glover, F.: Improved linear integer programming formulations of nonlinear integer problems. Management Science pp. 455–460 (1975)
7. Lee, E., Messerschmitt, D.: Static scheduling of synchronous data flow programs for digital signal processing. IEEE Trans. on Computers **36**(1), 24–35 (1987)
8. Lin, J., Srivatsa, A., Gerstlauer, A., Evans, B.: Heterogeneous multiprocessor mapping for real-time streaming systems. In: Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (2011)
9. Pino, J., Parks, T., Lee, E.: Automatic code generation for heterogeneous multiprocessors. In: Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, pp. 445–448 (1994)
10. Ruggiero, M., Guerri, A., Bertozzi, D., Poletti, F., Milano, M.: Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip. In: Proc. conf. on Design, Automation and test in Europe, pp. 3–8. European Design and Automation Association (2006)
11. Sih, G.: Multiprocessor scheduling to account for interprocessor communication. University of California at Berkeley, Berkeley, CA (1992)
12. Sriram, S., Lee, E.: Statically sceduling communication resources in multiprocessor dsp architectures. In: Signals, Systems and Computers, 1994. 1994 Conference Record of the Twenty-Eighth Asilomar Conference on, vol. 2, pp. 1046–1051. IEEE (1994)
13. Stuijk, S., Basten, T., Geilen, M., Ghamarian, A., Theelen, B.: Resource-efficient routing and scheduling of time-constrained streaming communication on networks-on-chip. Journal of Systems Architecture **54**(3-4), 411–426 (2008)
14. Stuijk, S., Geilen, M., Basten, T.: SDFˆ3: SDF For Free. In: Proc. IEEE Int. Conf. on Application of Concurrency to System Design, pp. 276–278 (2006)
15. Zhu, J., Sander, I., Jantsch, A.: Buffer minimization of real-time streaming applications scheduling on hybrid CPU/FPGA architectures. In: Proc. IEEE Conf. on Design, Automation and Test in Europe, pp. 1506–1511 (2009)
16. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the strength Pareto evolutionary algorithm. In: Eurogen, vol. 3242 (2001)
17. Zitzler, E., Teich, J., Bhattacharyya, S.: Evolutionary algorithms for the synthesis of embedded software. IEEE Trans. on VLSI Systems **8**(4), 452–455 (2000)
18. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. IEEE Trans. on Evolutionary Computation, **3**(4), 257–271 (1999)