HETEROGENEOUS MULTIPROCESSOR MAPPING FOR REAL-TIME STREAMING SYSTEMS

Jing Lin[⋆]*, Akshaya Srivatsa*[†]*, Andreas Gerstlauer*[⋆] *and Brian L. Evans*[⋆]

[⋆] The University of Texas at Austin, Austin, TX, USA
[†] Rice University, Houston, TX, USA

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 multi-processor platforms. The primary contributions include (1) an integer linear programming (ILP) model that globally optimizes throughput, latency and cost; (2) a low-complexity two-stage heuristic 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 a 10^{-6} gap from the ILP optimal solution, with up to 12x better run-time efficiency.

Index Terms— Synchronous data flow, streaming, heterogenous multiprocessor, mapping, scheduling

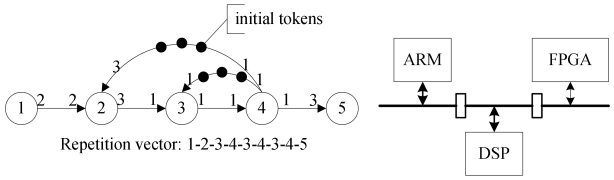
1 Introduction

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 fashion.

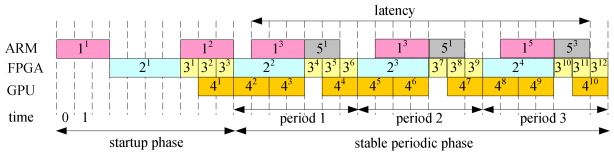
With the current trend towards heteregoeneous multiprocessor systems-on-chips (MPSoCs), the mapping of applications onto such platforms is among the most critical tasks in the system design process. Many real-time streaming systems can be modeled by synchronous data flow (SDF) graphs [1]. Multiprocessor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

Earlier research efforts were focused on SDF mapping on homogeneous multiprocessors [1][2]. More recent approaches have extended support to heterogeneous multiprocessor platforms [3]. In [4] the scheduling sub-problem was addressed through constraint programming, with the goal of minimizing memory requirements subject to throughput constraints. In [5] a graph-based solution was proposed to globally tackle the partitioning and scheduling problems for a sub-set of SDF graphs, named homogeneous SDF (HSDF) graphs, and several low-complexity heuristic acceleration techniques were presented. While all of the approaches above optimize for a single objective, 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. [6] and [7] have shown application of EA in circuit synthesis and uniprocessor software synthesis from SDF graphs.

To the best of our knowledge, we are the first to propose a multi-objective optimization framework that jointly optimizes throughput, latency and processor cost for general multiprocessor SDF mapping. We target platforms that are a combination of general processors, FPGA, DSP and other processing elements. The execution time of an actor is processor-dependent, and each actor is statically bound to a unique processor. Since multiple instances of an actor cannot be simultaneously executed, optimization techniques such as loop unfolding do not result in any throughput improvement, and hence are not considered in our work.



(a) An SDF graph and a multiprocessor platform.



(b) Partitioning and scheduling output.

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

This research was supported by an equipment gift from Intel.
A. Srivatsa conducted this research at the University of Texas at Austin.

2 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 (Fig. 1(a)). Also provided is a multiprocessor platform and the execution profile (i.e. the execution time of each actor on different processors).

An optimal processor partition and a parallel schedule are to be constructed. The schedule is represented by a finite-length time sequence consisting of a startup phase followed by one iteration of the stable phase (Fig. 1(b)). The throughput is the inverse of the iteration period. The latency is defined as the time interval between the source's start and the sink's end within the same iteration. Without loss of generality, we assume unique source and sink actors in the SDF graph, or otherwise pseudo actors with zero execution time could be added to represent the source and/or sink.

The following assumptions are made for the rest of the paper: each actor can only be statically mapped to a single processor; actors mapped to the same processor must be executed sequentially, and; to simplify the analysis, the inter-processor communication overhead is neglected (or alternatively, a global shared memory model is assumed, in which data cannot be transferred locally within one processor, i.e. memory access time can be folded into actor execution times).

3 A Global ILP Model

In this section, we develop an ILP model for simultaneous optimization of multiprocessor SDF partitioning and scheduling. First we formulate an integer nonlinear programming (INLP) model to capture the physical concepts. Then an equivalent ILP model is derived.

Let I be the number of actors in an SDF graph (suppose that actor 1 and actor I are the source and sink, repectively), and J be the number of processors. Parameters are defined in Table 1 for i, i_1, $i_2 \in \{0, \cdots I-1\}$ and $j \in \{0, \cdots J-1\}$.

We define the decision variables as the following. Let $S_i(t)$ and $E_i(t)$ represent the number of started (or ended) executions of actor i up to time t, respectively. They are two sets of time-indexed counting processes starting from 0 with unit increments. Also define two sets of binary variables: A_{ij} indicating whether actor i is mapped to processor j, and $start(t)$ indicating whether the periodic stable phase starts from time slot $(t+1)$. Note that $\sum_j A_{ij} = 1$ since every actor is statically mapped to a unique processor, and $\sum_t start(t) = 1$ since the length-T time window only includes one stable phase.

Parameter	Explanation
pc_j	Cost (user-defined measure of area, price, etc.) of processor j
d_{ij}	Execution time of actor i on processor j
$p_{i_1,i_2}(c_{i_1,i_2})$	The number of tokens produced (consumed) on edge (i_1, i_2)
n_i	The number of firings of actor i in one iteration
o_{i_1,i_2}	The number of initial tokens on edge (i_1, i_2)
T	The length of the time window in which the parallel schedule is constructed (i.e. the total length of the startup phase and one period of the stable phase)

Table 1: Input parameters of the ILP model.

The following constraints are added to satisfy the SDF execution semantics and the requirement posed by heterogeneous multiprocessor mapping.

Execution precedence. Enough tokens are accumulated on input edges before an actor fires:

$$c_{i_1,i_2} S_{i_2}(t) \le p_{i_1,i_2} E_{i1}(t) + o_{i_1,i_2} \tag{1}$$

Execution time of an actor. An execution starting at t ends after certain execution time:

$$S_i(t) = \textstyle\sum_j A_{ij} E_i(t + d_{ij}) \tag{2}$$

Sequential execution. Actors mapped to the same processor cannot be executed simultaneously:

$$\textstyle\sum_j A_{ij}(S_i(t) - E_i(t)) \le 1 \tag{3}$$

Periodicity in the stable phase. Each actor completes exactly one iteration in the stable phase:

$$W_i(T) - \textstyle\sum_t W_i(t) start(t) = n_i \sum_j A_{ij} d_{ij} \tag{4}$$

where $W_i(t)$ is the number of time slots that actor i has been executed untill time t, i.e. $W_i(t) = \sum_t (S_i(t) - E_i(t))$.

To jointly optimize the throughput (or equivalently the iteration period), latency and processor cost, we define the objective function to be a linear combination of them, i.e.

$$min \quad \{\lambda_1 \cdot Period + \lambda_2 \cdot Latency + \lambda_3 \cdot Cost\} \tag{5}$$

where λ_1, λ_2 and λ_3 are non-negative scalars to balance optimization across one or multiple objectives. To express the objectives in terms of decision variables, we use

$$Period = T - \textstyle\sum_t t \cdot start(t) \tag{6}$$

$$Cost = \textstyle\sum_j Alloc_j \cdot pc_j, \tag{7}$$

where $Alloc_j$ is the indicator of whether processor j is occupied. To represent latency, we add two sequences of variables, namely $U(t)$ and $V(t)$, which are defined as step functions that jump from zero to one in the time slot when $S_1(t)$ (for $U(t)$) or $S_I(t)$ (for $V(t)$) makes the first increment during the stable phase. Then we have

$$\begin{aligned} Latency = &\textstyle\sum_t (U(t) - V(t)) + \sum_j A_{Ij} d_{Ij} \\ &+ (S_1(T) - S_I(T)) \cdot Period, \end{aligned} \tag{8}$$

where the first two summations capture the time interval between the start of the source and the end of the sink during the stable phase, and the second product term captures their difference in iteration numbers.

In the model above, the nonlinearity comes from the product terms between binary and integer variables, and the indicator functions used to define $U(t)$, $V(t)$ and $Alloc_j$. These can be linearized according to [8], where the original INLP are transformed to an ILP in a higher dimensional space (i.e. new variables added) with more linear constraints. More

specifically, for each product term, a new variable is introduced, and is bounded by the original variables and a tight upper bound of the integer variable. The indicator functions are equivalently expressed as two linear inequalitites.

4 Two-Stage Heuristic Optimization

Despite of its completeness and optimality, the global ILP model is NP-hard [1], and hence efficient heuristics are desired to combat the exponentially increased complexity.

In practice there are plenty of design scenarios where optimizing the throughput and the cost is prioritized over minimizing latency. We observe that without explicit constraints on memory requirement, the maximum throughput of a mapped SDF graph is determined by the processor partitioning, regardless of the scheduling. Given the processor partition, the iteration period has a lower bound determined by the critical processor, the one that takes the longest time to execute one iteration of all the actors mapped to it, i.e.

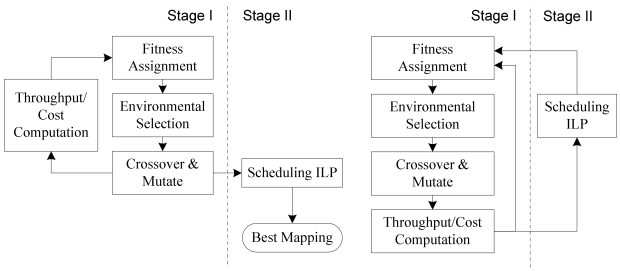
$$Period(A_{ij}) \geq max_j\{\sum_i n_i d_{ij} A_{ij}\} \quad (9)$$

By experiment we verify that the lower bound is achievable, provided a along enough startup phase, or enough initial tokens on feedback arcs for cyclic graphs. By optimizing the above jointly with the cost over all A_{ij} we make a first-stage decision on processor partitioning, which is one or possibly multiple optimal solutions of A_{ij}. Then, in a second stage, for each optimal processor partition a schedule that minimizes the latency can be found and the minimum achievable latencies for different partitions are compared to each other to obtain the best configuration of partition and schedule. Note that since the iteration period, the partition, and the execution time for each actor is known in the second stage, the ILP model is greatly simplified, with the complexity significantly reduced.

The two-stage decision process described above leads to a straightforward heuristic to decompose the global ILP into two sub-ILPs: a partitioning ILP that minimizes the weighted sum of $Period$ (as expressed in (9)) and $Cost$, followed by a scheduling ILP minimizing $Latency$. However in practice the first ILP can only generate a single optimal solution, while there could potentially exist other processor partitions that achieve the same throughput and cost but smaller latency. Therefore efficient searching algorithms are desired in the first stage to find as many optimal solutions as possible.

We propose an EA-driven heuristic that applies the Strength Pareto Evolutionary Algorithm II (SPEA-II) [9] on top of the scheduling ILP to reduce the optimality gap while maintaining a low computational complexity. A population of chromosomes, which are one-to-one mappings to the processor partition, reproduces and evolves over multiple generations under environmental pressure, posed by the fitness selection based on the achievable throughput, processor cost and latency. After a number of generations, the population converges and the best chromosomes are obtained.

The SPEA-II algorithm is capable of generating either a single sub-optimal mapping solution or a Pareto front for design space exploration. Fig. 2(a) illustrates the two-stage open-loop procedure to generate a single mapping decision. First we use SPEA-II to generate a 2-dimensional (processor cost vs. throughput) Pareto front. Then one or more points on the front are fed into the scheduling sub-ILP for latency optimization. Fig. 2(b) represents the two-stage closed-loop procedure to generate a 3-dimensional Pareto front.



(a) Generate a single decision. (b) Generate a Pareto front.

Fig. 2: Two-stage EA-driven heuristic algorithms.

Parameter	Value
Population Size	50
Archive Size	10
Termination Condition	Upon Convergence
Crossover Probablity	0.9
Mutation Probability	0.1
Crossover Method	Uniform Crossover
Selection Method	Roulette Wheel

Table 2: Parameters for SPEA-II.

5 Experimental Results

We have evaluated our model and the heuristic approaches on a variety of randomly generated SDF graphs and one simplified realistic example of the MP3 decoder (Fig. 3), both provided by SDF3[10]. We programmed the ILP models using CPLEX Concert Technology for C++ and used MOGALib Genetic algorithm framework to implement the SPEA-II. Table 2 lists parameters chosen for the SPEA-II. The choice of population size is according to the suggestion by [11] on the population size of $1.5 \log_2 N$, where N is the number of possible permutations of the chromosome. All experiments were run remotely on a single rack mounted Dell Poweredge 2950 workstation, having 2 dualcore, hyperthreading 3.73 GHz Xeon processors and 24 GB of shared memory.

To demonstrate the run-time efficiency of the proposed single-solution heuristics, we applied the global ILP model (1-ILP), the two-sub-ILP model (2-ILP) and the open-loop EA-driven heuristic (SingleEA) to the same set of homogeneous SDF (HSDF) graphs. We randomly generated cyclic and acyclic HSDF graphs with the number of actors ranging from 5 to 15, and recorded the average run-time of mapping these graphs onto a 3-processor platform with randomized execution profile (Fig. 4). For both cyclic and acyclic graphs, the SingleEA exhibits much better scalability than the 1-ILP in terms of run-time. The run-time of the 2-ILP is less than but very close to that of the SingleEA. As will be shown in the

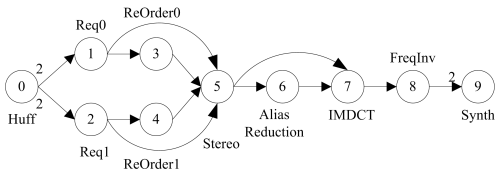


Fig. 3: A simplified SDF model for the MP3 decoder.

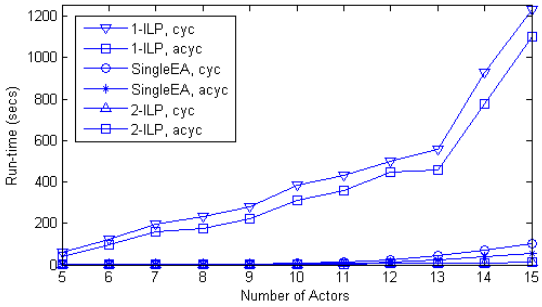


Fig. 4: Average run-time comparison for random SDF graphs mapped to a three-processor platform with randomized execution profiles.

following, the 2-ILP has the weakest optimality in latency.

To show the optimality gap of the SingleEA, we used throughput as a single objective in the 1-ILP, the first stage of the 2-ILP, and the first stage of the SingleEA. As illustrated by Table 3, for all tested cases both the SingleEA and the 2-ILP guaranteed the best throughput. The table also shows better or equal latency achieved by the SingleEA over the 2-ILP, since the best solutions of the former are picked from multiple scheduling ILPs for each of the optimal partitioning decisions. The latency optimality gap of the scheduling ILP in both heuristics is controlled within 10^{-6} in CPLEX.

For design space exploration using the proposed EA-driven heuristic, we applied the 1-ILP and the Closed-loop EA-driven heuristic (ParetoEA) in Fig. 2(b) to the MP3 decoder, which is a multirate acyclic SDF graph with 10 actors and 13 edges. The 1-ILP took an average of 1004 seconds to generate one optimal mapping solution. The ParetoEA took an average of 3624 seconds to converge to the Pareto front. Fig. 5 shows the convergence to the 3-dimensional Pareto front. In particular, the solution pointed out by the arrow, $(26, 60, 39)$, is the optimal solution by the global ILP with $\lambda_1 = 0.8$, $\lambda_2 = 0$ and $\lambda_3 = 0.2$, which implies closeness of the generated solution set to the optimal Pareto front.

6 Conclusion

This paper derives 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 effciency without compromising throughput optimality. The heuristics can generate either a single solution or the Pareto tradeoff curve for multi-dimensional design space exploration.

I	Type	Throughput			Latency		
		1-ILP	2-ILP	SingleEA	1-ILP	2-ILP	SingleEA
5	acyc	1/8	1/8	1/8	39	34	34
	cyc	1/12	1/12	1/12	60	47	35
10	acyc	1/20	1/20	1/20	73	58	51
	cyc	1/17	1/17	1/17	50	43	35
15	acyc	1/25	1/25	1/25	74	53	38
	cyc	1/25	1/25	1/25	62	33	33

Table 3: Optimality comparison for random SDF graphs with I actors mapped to a three-processor platform with randomized execution profiles.

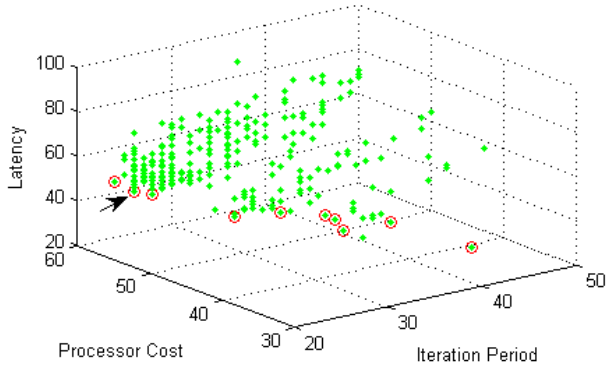


Fig. 5: Convergence to the Pareto front for the MP3 decoder.

References

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

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

[3] J.L. Pino, T.M. Parks, and E.A. Lee, "Automatic code generation for heterogeneous multiprocessors," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1994, pp. 445–448.

[4] J. Zhu, I. Sander, and A. Jantsch, "Buffer minimization of real-time streaming applications scheduling on hybrid CPU/FPGA architectures," in *Proc. IEEE Conf. on Design, Automation and Test in Europe*, 2009, pp. 1506–1511.

[5] A. Bonfietti, L. Benini, M. Lombardi, and M. Milano, "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*, 2010, pp. 897–902.

[6] N. Aslam, T. Arslan, and A. Erdogan, "Algorithmic level design space exploration tool for creation of highly optimized synthesizable circuits," in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2007, vol. 2.

[7] E. Zitzler, J. Teich, and S.S. Bhattacharyya, "Evolutionary algorithms for the synthesis of embedded software," *IEEE Trans. on VLSI Systems*, vol. 8, no. 4, pp. 452–455, 2000.

[8] F. Glover, "Improved linear integer programming formulations of nonlinear integer problems," *Management Science*, pp. 455–460, 1975.

[9] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," in *Eurogen*, 2001, vol. 3242.

[10] S. Stuijk, M. Geilen, and T. Basten, "SDF^3: SDF For Free," in *Proc. IEEE Int. Conf. on Application of Concurrency to System Design*, 2006, pp. 276–278.

[11] J.T. Alander, "On optimal population size of genetic algorithms," in *Proc. IEEE Int. Conf. on Comp. Sys. and Software Eng.*, 2002, pp. 65–70.