Pareto Efficient Design for Reconfigurable Streaming Applications on CPU/FPGAs

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

***Abstract*—We present a Pareto efficient design method for multi-dimensional optimization of run-time reconfigurable streaming applications on CPU/FPGA platforms, which automatically allocates applications with optimized buffer requirement and software/hardware implementation cost. At the same time, application performance is guaranteed with sustainable throughput during run-time reconfigurations. As the main contribution, we formulate the constraint based application allocation, scheduling, and reconfiguration analysis, and propose a design Pareto-point calculation flow. A public domain solver - Gecode is used in solutions finding. The capability of our method has been exemplified by two cases studies on applications from media and communication domains.**

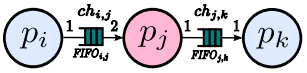
I. Introduction

To deliver high performance streaming media applications with reduced design costs and time-to-market, there are trends in embedded system design to use combined components of multi-CPU and custom circuits in commercial off-the-shelf (COTS) FPGAs, the so called hybrid multi-CPU/FPGAs [1]. While free FPGA gates are customized as application-specific reconfigurable components for performance-critical functions and CPUs as execution engines for software, such COTS chips can be used as embedded platforms with high throughput and run-time reconfiguration (RTR) requests [2]. One pragmatic application of them, for instance, is being used in a roaming smart-phone with reconfigurable communication protocols.

Unfortunately, the design space exploration techniques and optimization methodologies on hybrid multi-CPU/FPGAs are still immature. The difficulties exit in two categories:

- While applications scheduling with resource constraints on multi-processors has been know to be NP-complete [3], the reconfiguration analysis even increases the design complexity.
- To design systems that use less resources (cost) without losing performance guarantees, the decision-making over diversified COTS platforms remains difficult. For instance, the XC5VFX devices in Xilinx Virtex-5 family cover a wide range of 380∼2280KB reconfigurable logic blocks and up to two PowerPC cores [4].

To design predictable streaming applications, synchronous data flow (SDF) model [5] has been widely used. An example SDF application is depicted in Fig. 1(a). The nodes denote computation *processes*, and the edges denote communication *channels* associated with FIFO buffers. Each time a process executes, it *consumes* (*produces*) a fixed token rate *from input-side* (*into output-side*) FIFOs. These numbers are denoted as symbols at the each side of channels. For instance, process



(a) Example application model
(p_j is reconfigurable)

	HW reconfig. OH	Cost SW	Cost HW	WCET SW	WCET HW
p_i	-	-	1	-	1
p_j	2	1	2	3(2)	2(1)
p_k	-	1	2	2	1

(b) Design specifications
(p_j has two working modes)

Fig. 1: An example reconfigurable SDF application model and its design specifications on a CPU/FPGA platform.

p_j consumes 2 tokens from channel $ch_{i,j}$ and produces 1 token into channel $ch_{j,k}$ on each invocation (firing). A process is enabled and ready for execution when both the input-side FIFOs have sufficient data tokens and the output-side FIFOs have enough vacant space. While a process is computing, the data tokens remain in input-side FIFOs until the computation is completed and the output results are available in output-side FIFOs at the end of each execution. Especially, the computation of p_j is reconfigurable with two working modes. Each process (working mode) has a worst case execution time (WCET) when it is partitioned as either SW or HW. While software (SW) reconfigurations correspond to process context switch on processors, the overhead (OH) is negligible compared with process execution time. The hardware (HW) reconfiguration OH to manipulate reconfigurable logics on FPGAs at run-time is non-trivial, during which the computation is stalled. Assuming a specified number of processors on each chip, the implementation cost of each process is either the size of SRAM storage for SW or the area of logic blocks and (reconfiguration) memory for HW. In Fig. 1(b), the design specifications (normalized) to implement the illustrative application on a single-CPU/FPGA platform are exemplified.

Motivation. Given the example application and its design specifications in Fig. 1, we explore three design options and analyze their reconfiguration scheduling. With design option A (p_i and p_j partitioned to HW and p_k to SW), the schedule is illustrated in Fig. 2(a). The time range of the scheduling evolves horizontally, while the process and FIFO status are listed out vertically, i.e., at each time tag a process in *executing* state has a number to denote the remaining execution time slots, a FIFO status is denoted as the using storage space in tokens, and processes with *stalled* computation or FIFOs not used have otherwise blank status. 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, the schedule enters a periodic phase from time tag

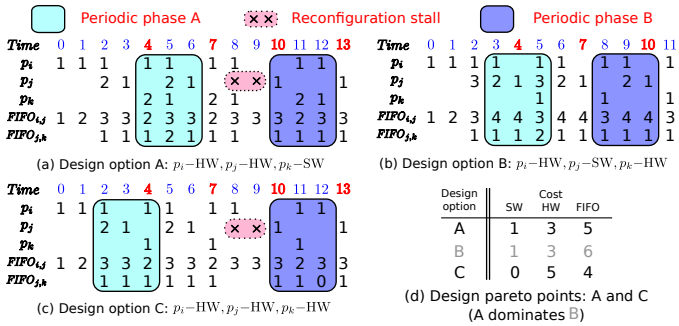


Fig. 2: Example application scheduling (p_j has two working modes A and B) and design Pareto points.

4 to 6, in which p_j works in *mode A* (with WCET 2 slots). From time tag 8, p_j starts the reconfiguration[1] with an OH 2 and switches to working *mode B* (with WCET 1 slot). Again, the schedule enters another periodic phase from time tag 10 to 12. The application throughput is guaranteed in both periodic phases, since all processes have the same execution iterations in a time period 3. In addition, the application throughput is guaranteed even during mode transition, e.g., between time tags 4, 7, and 10 the sink process p_k always executes once. For each FIFO, the required FIFO size is the maximal buffer usage in scheduling. Similarly, the reconfiguration scheduling of two other design options are illustrated in Fig. 2(b-c), in which p_j is implemented in SW in design option B with neglected reconfigurable OH. Apparently, all design options have the same application throughput guarantees. Their SW/HW cost, and FIFO buffer requirement are evaluated in Fig. 2(d). Although these quantities have a partial order to indicate the precedence of design options, A dominates B in the sense that the former one is always preferred with not worse (bigger) evaluations on all criteria. The set of optimal design options A and C are called Pareto points [6], i.e., either one has at least one criteria better than the other (A has less HW cost, and B has less SW and FIFO cost). In this paper, we use Pareto points to indicate the trade-offs of multiple criteria in design space exploration, which can not be optimized independently. However, a systematic way to calculate design Pareto points in reconfigurable streaming applications is still lacking.

As the **contribution in this paper**, we propose a new Pareto efficient design framework for reconfigurable SDF streaming applications on hybrid multi-CPU/FPGAs platforms. The problem is formulated as constraint based allocation and scheduling of streaming applications with guaranteed throughput even during the run-time reconfigurations. The pruned Pareto-optimal points found can be used in cost-efficient selection from variably priced CPU/FPGA platforms.

This paper is structured as follows: the related work is introduced in Section II. Our application model and architecture platform are introduced in Section III. We present our constraint based framework for design Pareto-point calculation in Section IV. Section V shows our experimental results. Finally, Section VI concludes the paper.

[1]For clarity, we fix the reconfiguration time slot to 8. But, our method to be presented in Section IV is more general and fits run-time reconfigurations.

II. Related Work

Bilsen et al. [7] first present cyclo-static dataflow model, which supports cyclically changing of the number of tokens produced and consumed by processes. Since the mode changing behavior is predefined at compile-time, static schedules can be constructed when the necessary and sufficient conditions for scheduling hold. Furthermore, the buffer requirement is analyzed for cyclo-static dataflow models according to the specified throughput requirement in [8]. However, the mode changing problem to be addressed in this paper is more challenging in the sense that the reconfigurations are only known at *run-time* (unpredictable at compile-time).

In [9], simulation based techniques have been introduced for the analysis of different performance metrics of scenario-aware SDF models with stochastic mode changes. Schedulability analysis and reconfiguration methods for multi-mode (adaptive) real-time systems has been studied in [10, 11], where each mode consists of different tasks. They develop mode change protocols in mode transition stages, and exploit analysis techniques to ensure that no deadlines are violated during the transition periods. On RTR hybrid CPU/FPGAs, Yuan et al. [12] address SW/HW partitioning and scheduling of task graphs with the objective of maximizing application throughput. All these work do not address multi-dimensional optimization on platform resource, as we do in this paper.

An inspiring Pareto calculator has been proposed for the optimization of multi-dimensional space of attributes [6], which is a tool for general compositional computations. It has been used to calculate optimized design options for MPEG-4 media on mobile devices via a wireless connection, with the trade-offs on the quality of the video, energy consumption and transmission latency. Based on our previous work on performance analysis of SDF applications on reconfigurable FPGA [13] with integer linear programming (ILP) techniques, and the constraint based scheduling on a hybrid (non-reconfigurable and static allocation) SW/HW architectures [14], we propose, in contrast to the existing work, a new constraint based framework for SW/HW allocation, scheduling, and run-time reconfiguration analysis of SDF streaming applications on multi-CPU/FPGA platforms. A design Pareto-point calculation flow for SW/HW and buffer cost efficient design is proposed to integrate with domain specific composable (linear and non-linear) constraints.

III. Application and Architecture Models

We consider a subset of SDF models, which are said to be consistent [5]. Given a producer-consumer processes p_i and p_j with channel $ch_{i,j}$, p_i has output rate $n_{i,j}$ and p_j has input rate $m_{i,j}$. For consistent SDF models, p_i and p_j can run in a repetitive pattern with non-trivial (non-zero) firing times r_i and r_j, where r_i and r_j are the minimum integer solutions of a set of balance equations for all channels in the model.

$$r_i \cdot n_{i,j} = r_j \cdot m_{i,j} \quad (1)$$

A vector of the non-trivial firing times for each process in the model is called *repetition vector*. A regular SDF model can be transformed (expanded) to an equivalent homogeneous SDF (HSDF) model with all input/output rate 1, but this transformation dramatically increases the problem size (Section V). Hence, we work on regular SDF models directly.

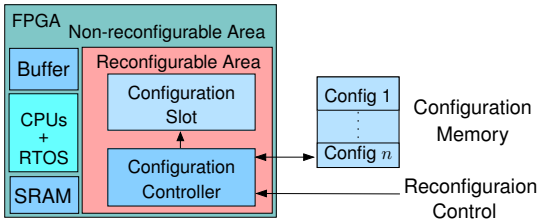


Fig. 3: Partially RTR CPU/FPGA architecture model.

The partially RTR multi-CPU/FPGA architecture is illustrated in Fig. 3. There are CPUs dedicated to SW processes and custom circuits to HW processes. While the cost of memory (SRAM) and custom circuits on FPGAs is determined by application allocation decision, the buffer requirement depends heavily on scheduling policies (RTOS). Given a reconfigurable process allocated as HW, new functionality can be loaded from the configuration memory into the configuration slot (reconfigurable circuits in FPGA) specified by the reconfiguration control. Since this loading takes time (reconfiguration stall), the system timing might be violated without reconfiguration analysis. On the other hand, processes improperly implemented as SW might degrade system performance, especially when the number of processors is limited. It is critical to meet throughput guarantees even during the run-time reconfiguration transitions.

Our work is based on the following assumptions:

- The reconfiguration of one process does not interfere the working of other running processes on such a partially RTR CPU/FPGAs platform.
- The hard-core multi-processors are *homogeneous*, on which each process has the same WCET being mapped onto different processors, e.g., PowerPC on Virtex-5.
- The cost to implement processes as either SW or HW is known at design-time in specifications.

IV. Methodology

Here, we present our methodology and constraint based framework. The problem has been formulated as both linear and non-linear (e.g., *multiplication*) constraints. The time tag t in our formulation is discrete numbers with $t \in \mathbb{N}_0$, when it is not otherwise clarified.

A. Allocation and mapping

A set of boolean decision variables μ_i denote whether each process p_i is allocated to CPUs ($\mu_i = 0$) or FPGAs ($\mu_i = 1$). We assume that different computation iterations (instances) of a process allocated to CPUs can only execute on one specified processor. Then, a set of boolean variables $\alpha_{i,\mu p_n}$ indicate the presence of p_i on processor μp_n, with $\alpha_{i,\mu p_n} \equiv 0$ for processes allocated to FPGAs. Processes allocation and mapping can be formalized as the following.

Constraint 1. *(Allocation & mapping) While each process p_i can be allocated to CPUs or FPGAs, a process allocated to CPUs needs one (and only one) processor for different process instances.*

$$\sum_{\mu p_n \in \boldsymbol{U}} \alpha_{i,\mu p_n} = \neg \mu_i, \quad \forall p_i \in \boldsymbol{P} \tag{2}$$

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

The mapping problem on homogeneous multi-processors contains symmetries, i.e., for some mapping decisions, there are duplicated equivalent solutions in the searching space. Thus, we apply a stronger restriction (Eq. 3) to order the processors in allocation to exclude revisiting symmetrically equivalent states.

$$\sum_i 2^i \alpha_{i,\mu p_n} \geqslant \sum_i 2^i \alpha_{i,\mu p_{n+1}}, \quad \forall \mu p_n, \mu p_{n+1} \in \mathbf{U} \tag{3}$$

in which the equality holds when neither of the consecutive processors μp_n and μp_{n+1} has processes allocated.

B. Extended execution semantics

In our previous work [14], event models based on cumulative functions were used to capture process working load and pressing capabilities. For instance, in the example SDF application in Fig. 1(a), 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.

$$D_{i,j}(t) = \begin{cases} R_{i,j}(t) + n_{i,j}, & \text{if } p_i \text{ is executing;} \\ R_{i,j}(t), & \text{if } p_i \text{ is stalling.} \end{cases} \tag{4}$$

Accordingly, the execution semantics of SDF applications has been formalized (refer to [14]). For instance, the buffer usage in scheduling (denoted as numbers for each FIFO in Fig. 2) can be derived as the following.

Property 1. *(Buffer usage) In scheduling, the buffer space in usage $B'_{i,j}(t)$ for $FIFO_{i,j}$ is the difference between $D_{i,j}(t)$ and $C_{i,j}(t)$ plus an offset of the initial data tokens $B^0_{i,j}$ in $FIFO_{i,j}$.*

$$B'_{i,j}(t) = D_{i,j}(t) - C_{i,j}(t) + B^0_{i,j} \tag{5}$$

In this paper, we extend the execution semantics to be *allocation* aware, and captures the run-time *reconfiguration* of the reconfigurable process as well. Besides the allocation decision variables μ_i, we use a boolean function $\xi(t)$ to denote the working mode at time tag t of the reconfigurable process p_j, i.e., $\xi(t) = 0$ indicates that p_j works in mode A , otherwise p_j is in (or being reconfigured to) mode B. For instance, the process computation latency is formulated in the following constraint.

Constraint 2. *(Computation latency) While each process can be allocated to CPUs or FPGAs, its WCET in implementation*

is denoted as $t_{C_{sw},j}$ or $t_{C_{hw},j}$ with the following constraints.

$$C_{i,j}(t+t_{C,j}) = C_{i,j}(t) + m_{i,j}K_j(t), \quad \forall K_j(t)\in\{0,1\} \tag{6}$$

$$where \quad t_{C,j} = \neg\mu_j t_{C_{sw},j} + \mu_j t_{C_{hw},j} \tag{7}$$

in which $K_j(t)$ denotes the incremental properties of $C_{i,j}(t)$. Especially, for a reconfigurable process p_j, the WCET varies from $t^A_{C_{sw},j}(t^A_{C_{hw},j})$ to $t^B_{C_{sw},j}(t^B_{C_{hw},j})$ during the reconfiguration from working mode A to B. Thus, $t_{C,j}$ in Eq. 6 can be defined:

$$\begin{aligned} t_{C,j} =& \neg\xi(t)(\neg\mu_j t^A_{C_{sw},j} + \mu_j t^A_{C_{hw},j}) + \\ & \xi(t)(\neg\mu_j t^B_{C_{sw},j} + \mu_j t^B_{C_{hw},j}) \end{aligned} \tag{8}$$

However, the constraints in Eq. (6-8) can not be implemented in Gecode solver directly and need to be rewritten. For instance, Eq. (6-7) are equivalent to the following constraint applicable in Gecode.

$$\begin{aligned} &\neg\mu_j C_{i,j}(t+t_{C_{sw},j}) + \mu_j C_{i,j}(t+t_{C_{hw},j}) \\ &\quad = C_{i,j}(t) + m_{i,j}K_j(t), \quad \forall K_j(t)\in\{0,1\} \end{aligned} \tag{9}$$

Without being explicitly mentioned in this paper, $t_{C,j}$ has the same definitions as in Eq. (6-8) to be allocation decision and reconfiguration (for the reconfigurable process) aware. Similarly, other SDF execution semantics on computation and buffer resources can be extended, which are omitted for clarity in this paper.

Furthermore, the process scheduling with computation and buffer resource constraints is refined to be aware of the allocation and mapping decisions.

Constraint 3. *(Allocation & scheduling association) While the processes allocated to FPGAs are concurrent in scheduling, the processes allocated to one single processor can only execute sequentially. This mapping and scheduling association is formalized as:*

$$\sum_{p_j\in\boldsymbol{P}} \alpha_{j,\mu p_n} W_j(t) \in \{0,1\},\ \forall \mu p_n \in \boldsymbol{U} \tag{10}$$

$$W_j(t) = \sum_{\Delta_t} \frac{C_{i,j}(t+\Delta_t+1) - C_{i,j}(t+\Delta_t)}{m_{i,j}}, \quad \Delta_t\in[0,t_{C,j}) \tag{11}$$

in which $W_j(t)$ denotes the 1-0 *(computing or stalling) status of process p_j at time tag t on CPUs or FPGAs, and $t_{C,j}$ has the same definitions as in Eq. (6-8) to be allocation decision and reconfiguration aware.*

C. Reconfiguration analysis

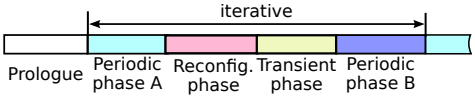


Fig. 4: Timing phases of reconfiguration analysis.

We decompose the reconfiguration analysis into different stepwise timing phases, as illustrated in Fig. 4.

- **Prologue**: is the start-up phase with no throughput guarantees. The length of the prologue phase can be controlled and specified by τ_0 in Constraint A-1 (Appendix).
- **Periodic phase A(B)**: are phases with guaranteed application throughput as defined in Constraint A-1 and A-2. The throughput requirement can be distinct for different working modes. The length L_{period} is throughput relevant, and can be specified in constraints. Periodic phase A guarantees a sustainable throughput in mode A of reconfigurable process p_j before the run-time reconfiguration request.
- **Reconfiguration phase**: consists of a period working in mode A, during which the reconfiguration starting time tag t' is explored (optimized) upon the reconfiguration request. The reconfiguration stall takes $t_{R,j}$ (Constraint A-3). The length of this phase is specified to be the worst case $L_{period} + t_{R,j}$.
- **Transient phase**: has a length τ_1, in which throughput is met but no periodic properties in scheduling yet.

To make the phases in gray (colored) iterative, we can use the timing analysis to explore more reconfiguration scenarios, and adopt the worst case requirement for each buffer size.

D. Design Pareto points calculation

In design specifications, the cost to implement each process p_i in SW (if feasible) on CPUs is measured as the code size π_i^{SW} in SRAM or the area of custom circuits π_i^{HW} in HW. The only unit we use for HW area cost is the number of logic elements (LE). The memory cost for a configuration stored in the configuration memory is thus calculated as the equivalent LE cost which is technology dependent. The SW and HW implementation cost on our platform are defined.

Property 2. *(SW & HW implementation cost) The process allocation determines the total SW cost π^{SW}_{Sum} and total HW cost π^{HW}_{Sum}.*

$$\pi^{SW}_{Sum} = \sum_{p_i\in\boldsymbol{P}} \neg\mu_i \cdot \pi_i^{SW},\ \pi^{HW}_{Sum} = \sum_{p_i\in\boldsymbol{P}} \mu_i \cdot \pi_i^{HW} \tag{12}$$

Assuming different FIFOs are implemented disjointly, the buffer cost on the platform is formalized.

Property 3. *(Buffer cost) In scheduling, the buffer cost in total corresponds to the sum of the maximal buffer usage of all FIFOs.*

$$\gamma_{Sum} = \sum_{\forall ch_{i,j}} \max_{\forall t} B'_{i,j}(t) \tag{13}$$

To evaluate the quality of design options in solutions finding, a constraint on SW/HW implementation cost and buffer requirement is formulated to prune design options.

Constraint 4. *(Design options pruning) Given a set of design Pareto points $ParetoSet$ found in design space exploration. Besides meeting all other constraints formulated above, a solution sol should not be dominated ($\nprec$) by any Pareto points ϕ.*

$$\begin{aligned} sol \nprec \phi \Longleftrightarrow & sol.\pi^{SW}_{Sum} < \phi.\pi^{SW}_{Sum} \ \vee\ sol.\pi^{HW}_{Sum} < \phi.\pi^{HW}_{Sum} \\ & \vee\ sol.\gamma_{Sum} < \phi.\gamma_{Sum}, \quad \forall\phi\in ParetoSet \end{aligned} \tag{14}$$

in which $sol \nprec \phi$ always holds when $ParetoSet = \emptyset$.

To maintain a set of Pareto points in design space exploration, we present such a calculation flow in Algorithm 1. In line 4, while all the formulated constraints are used in

solutions finding, a $ParetoSet$ is dynamically maintained[2] during exploration and used in Constraint 4. From line 8 to 10, if a sol dominates ($\prec$) a current Pareto point, the dominated design option is moved into another $DominatedSet$.

Algorithm 1: Design Pareto-point calculation flow.

```
   Output: ParetoSet
1  ParetoSet ⟵ ∅;
2  DominatedSet ⟵ ∅;
3  /* A dynamic ParetoSet is used in Constraint 4 */
4  while (sol = solutionsFinding(ParetoSet)) ≠ Null do
5      if |ParetoSet| > 0 then
6          for k ← 1 to ParetoSet.size do
7              /* If sol dominates a Pareto point */
8              if sol ≺ ParetoSet.at(k) then
9                  DominatedSet.insert(ParetoSet.at(k));
10                 ParetoSet.erase(k);
11     ParetoSet.insert(sol)
```

Discussion. The SW/HW and buffer cost in the proposed formulation are based on high-level estimations, e.g., buffer cost based on symbolic token units. However, it is possible to extend our method within a practical design flow, once these vendor and technology dependent factors can be formalized as design constraints. For instance, assuming FIFO buffers are implemented as Block RAM (BRAM), a more practical buffer cost γ_{Sum}^{BRAM} can be re-formalized from Eq. 13.

$$\gamma_{Sum}^{BRAM} = k_{BRAM} \sum_{\forall ch_{i,j}} d_{BRAM} \lceil \frac{\max_{\forall t} B'_{i,j}(t)}{d_{BRAM}} \rceil \qquad (15)$$

in which k_{BRAM} and d_{BRAM} are the cost factor and depth of BRAM.

V. Experimental results

To evaluate the potential of our methodology, we use it on a *Cd2dat* [15] application from the media domain, and a *Wireless* [16] application from the communication domain, besides the example application Fig. 1(a). We implement our method on top of the public domain constraint solving toolkit *Gecode* [17], which is a library written in C++. All experiments are carried out on a HP xw4600 Linux workstation with a Quad-Core[3] 2.40GHZ processor and 4GB of RAM.

We adopt a reconfigurable FPGA platform with two processors, e.g., Xilinx Virtex-5 FPGAs. The design specifications of different SDF applications are presented in Table I. For each application, the number of SDF processes and the equivalent HSDF process number are listed out. Especially, part of the SDF processes have been pre-allocated on the platform, either as SW or HW modules. In each application, we specify one process to be reconfigurable with two working modes. Assuming the same specified (feasible) application throughput requirement needs to be sustained (not only in different working modes but also in reconfiguration phases), we explore the application allocation, scheduling, and reconfiguration analysis between mode transitions. However, within the scope of this paper, we do not investigate the impacts of varying application throughput on design Pareto points.

The peak memory and running time on the experimental workstation have been measured in the solver, which are shown in Table I as well. We see that the memory and time usage increase exponentially with the problem size. Our method has been computation efficient to solve the NP-hard allocation and scheduling problem with a reasonable problem size, e.g., in 1.9s for the small example case and 105s for *Cd2dat*. The design Pareto points for different application specifications are illustrated in Fig. 5. For the example application, Pareto points with even less FIFO cost than in Fig. 2 can be found, since a lower throughput requirement has been specified in Table I (0.2 instead of 0.33). We have distinguished Pareto points to implement the reconfigurable process in either SW or HW in the graph, marked as 'Pareto point (SW)' or 'Pareto point (HW)' respectively. The buffer costs found are very tight[4], in the sense that for each application the minimal buffer cost in all Pareto points is the same as the minimal dead-lock avoiding bounds [18]. Another set of temporal Pareto points $DominatedSet$ dominated by new solutions found in design space exploration (line 8 to 10 in Algorithm 1) are presented. However, more design options which are pruned by Pareto points are discarded. For instance, 263 failure nodes are pruned for *Wireless*, while each node corresponds to a set of design options for our formulation.

Memory issues. In our experiments, the peak memory usage increases dramatically with the problem size, e.g., $4.18e6$KB (very close to the 4GB RAM capacity on our workstation) is used for *Cd2dat*. On the other hand, memory usage up to 2GB has also been reported in a model checking method in [18], in which a relatively simpler problem (only the scheduling of SDF applications) is considered. In another experiment, we fixed the processes allocation to either SW or HW (i.e., process number allocated to SW/HW is 0 in Table I) for *Wireless*. A Pareto point has been found in 2.1s with peak memory 414MB. Thus, it might be possible to use heuristics in the constraint based techniques to improve the searching efficiency, in terms of computation time and memory usage.

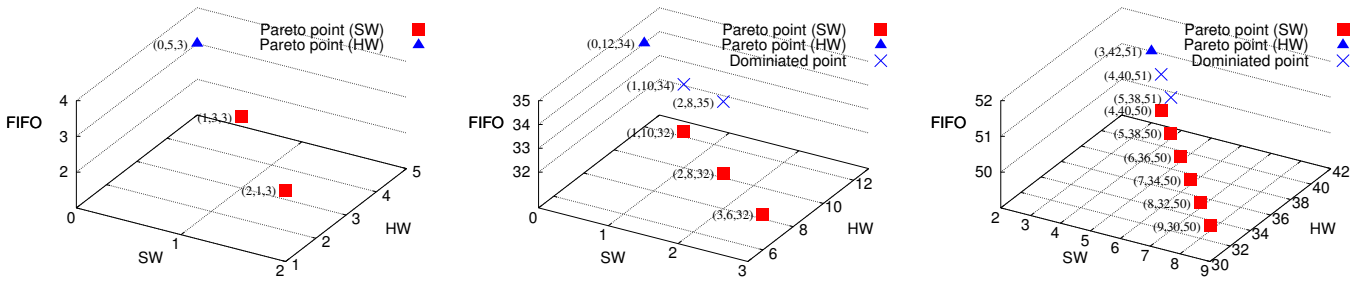
VI. Conclusion

In this paper, we have presented a Pareto efficient design method for reconfigurable streaming applications on off-the-shelf CPU/FPGA platforms. The problem is formulated as constraint based application allocation, scheduling, and reconfiguration analysis. A design Pareto-point calculation flow for SW/HW and buffer cost is implemented on a public domain constraint solver Gecode [17], and is exemplified by two case studies from different application domains.

[2]To our best knowledge, it is infeasible with ILP modeling techniques.

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

[4]They are throughput and reconfiguration stall $t_{R,j}$ relevant. For *Cd2dat* and *Wireless*, we specify $t_{R,j}$ to be 1-2 orders-of-magnitude higher than $t_{C,j}$.



(a) Example (b) Cd2dat (c) Wireless

Fig. 5: Design Pareto points for different applications.

TABLE I: Design specifications and experimental results.

application	*process #* [a]		*pre-allocation* [b]			*thru. req.*	*mem.* [c]	*time* [d]
	SDF	*HSDF*	*SW*	*HW*	*SW/HW*			
Example	3	4	0	1	2	0.2	1.29e3	1.90e3
Cd2dat	6	612	0	3	3	0.5	4.18e6	1.05e5
Wireless	24	32	3	15	6	0.028	3.80e6	1.94e4

[a] The number of SDF processes and equivalent HSDF processes.
[b] The number of processes pre-allocated as SW or HW on the platform.
[c] The peak memory consumption (KB) in solutions finding.
[d] The solutions finding time (ms).

Recently, the latest Gecode 3.1.0 starts to support parallel search in multiple threads. However, the searching speed on multi-thread heavily depends on whether the search tree can be distributed to each thread efficiently, and it takes more memory (more than single thread) as well [17]. To consider using multiple threads in searching and using heuristics with reduced peak memory in the exploration of search tree remain to be our future work.

ACKNOWLEDGEMENTS

This research has been partially supported by the SYSMODEL project through the European ARTEMIS programme.

APPENDIX

Constraint A-1. *(Application throughput) After some start-up time period τ_0 ($\tau_0 > 0$, with no stable output tokens), a specified throughput ρ_k should be met to sustain the required output rate at the application sink process p_k.*

$$C_k(\tau_0 + cL_{period}) - C_k(\tau_0) \geqslant \rho_k cL_{period}. \quad \forall c \in \mathbb{N}_0 \qquad \text{(A-1)}$$

Empirically, we choose $L_{period} = q \cdot \lceil \frac{r_k}{\rho_k} \rceil, q \in \mathbb{N} \backslash \{\infty\}$, with r_k the firing times of p_k in the repetitive pattern.

Constraint A-2. *(Periodic phase) The repeated process and FIFO status at time tag t' and $t' + L_{period}$ determines a periodic phase between them with length L_{period}.*

$$B'_{i,j}(t') = B'_{i,j}(t' + L_{period}), \quad \forall FIFO_{i,j} \qquad \text{(A-2)}$$

$$W'_i(t') = W'_i(t' + L_{period}), \quad \forall p_i \in \boldsymbol{P} \qquad \text{(A-3)}$$

$$\textit{where} \quad W'_i(t') = \sum_{k=1}^{t_{C,i}} k \cdot C_i(t' + k)$$

in which variables $W'_i(t')$ and $W'_i(t' + L_{period})$ are process status (denoted as numbers for each process in Fig. 2).

Constraint A-3. *(Reconfiguration stall) When the reconfigurable process p_j is allocated to FPGAs, the computation of p_j stalls for $t_{R,j}$ time period after the reconfiguration starts at time tag t'. On the other hand, p_j allocated to CPUs has ignorable reconfiguration overhead, i.e., no mandatory computation stalls.*

$$\mu_j \cdot \xi'(t') \cdot W_j(t' + \Delta_{t'}) = 0, \quad \forall \Delta_{t'} \in [0, t_{R,j}) \qquad \text{(A-4)}$$

$$\xi'(t') = \xi(t) - \xi(t-1), \quad \forall t \in \mathbb{N}, \xi'(0) = 0 \qquad \text{(A-5)}$$

in which $\xi'(t')$ is the derivation of $\xi(t')$, and $\xi'(t) = 1$ indicates that the reconfiguration starts at time tag t'.

REFERENCES

[1] D. Andrews, D. Niehaus, R. Jidin, M. Finley, W. Peck, M. Frisbie, J. Ortiz, E. Komp, and P. Ashenden, "Programming models for hybrid FPGA-CPU computational components: A missing link," *IEEE Micro*, vol. 24, no. 4, pp. 42–53, 2004.

[2] M. Barr, "A reconfigurable computing primer," *Multimedia System Design*, pp. 44–47, Septemer 1998.

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

[4] Xilinx Ltd, http://www.xilinx.com.

[5] 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.

[6] M. Geilen and T. Basten, "A calculator for pareto points," in *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, San Jose, CA, USA, 2007, pp. 285–290.

[7] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete, "Cyclo-static dataflow," *IEEE Transactions on Signal Processing*, vol. 2, no. 44, pp. 397–408, 1996.

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

[9] B. D. Theelen, M. Geilen, T. Basten, J. Voeten, S. V. Gheorghita, and S. Stuijk, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis." in *MEMOCODE*. IEEE, 2006, pp. 185–194.

[10] Y. Shin, D. Kim, and K. Choi, "Schedulability-driven performance analysis of multiple mode embedded real-time systems," in *DAC '00: Proceedings of the 37th conference on Design automation*, New York, NY, USA, 2000, pp. 495–500.

[11] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Syst.*, vol. 26, no. 2, pp. 161–197, 2004.

[12] M. Yuan, X. He, and Z. Gu, "Hardware/software partitioning and static task scheduling on runtime reconfigurable FPGAs using a SMV solver," in *RTAS '08: Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 295–304.

[13] J. Zhu, I. Sander, and A. Jantsch, "Performance analysis of reconfiguration in adaptive real-time streaming applications," in *Proceedings of the 6th Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia '08)*, Atlanta, USA, October 2008, pp. 53–58.

[14] ——, "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.

[15] 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.

[16] 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.

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

[18] 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.