Near-optimal Deployment of Dataflow Applications on Many-core Platforms with Real-time Guarantees

Stefanos Skalistis
Rigorous System Design Laboratory (RiSD)
École Polytechnique Fédérale de Lausanne (EPFL)
Email: stefanos.skalistis@epfl.ch

Alena Simalatsar
Rigorous System Design Laboratory (RiSD)
École Polytechnique Fédérale de Lausanne (EPFL)
Email: alena.simalatsar@epfl.ch

***Abstract*—*Safe* and *optimal* deployment of data-streaming applications on many-core platforms requires the realistic estimation of task Worst-Case Execution Time (WCET). On the other hand, task WCET depends on the deployment solution, due to the varying number of interferences on shared resources, thus introducing a cyclic dependency. Moreover, WCET is still an over-approximation of the Actual Execution Time (AET), thus leaving room for run-time optimisation. In this paper we introduce an offline/online optimisation approach. In the offline phase, we first break the cyclic dependency and acquire safe and near-optimal solutions for tasks *partitioning/placement, mapping, scheduling* and *buffer allocation*. Then, we tighten the WCETs and update the *scheduling* function accordingly. In the online phase we introduce a safe distributed readjustment of the offline schedule, based on the AET. Experiments on a Kalray MPPA-256 platform show a tightening of the guaranteed latency up to 46% in the offline phase, and 41% latency reduction in the online phase. In total, we achieve more than 50% of latency reduction.**

I. Introduction

In terms of timing, safety-critical applications require to provide real-time guarantees. On the other hand, the increasing size of data to be processed enforces the optimal use of resources of the underlying platform. For a uniprocessor system, safe and near-optimal deployment has been addressed, e.g. in [1]. However, the increasing processing demands calls for more powerful platforms, such as many-core. *Data streaming* applications [2] are composed of computation tasks with data dependencies and are often represented with *data-flow* models [3], [4]. The deployment of such applications onto many-core platforms exhibits a high degree of task parallelism, the data exchange among which results in intensive resource sharing. Safe and optimal deployment of such applications on many-core platforms requires the realistic estimation of the WCET of tasks. However, task WCET varies according to the deployment, due to *interference* delays when tasks simultaneously access shared resources, e.g. buses/memories, which results in a cyclic dependency. Moreover, even if the WCET is realistically estimated, it can still largely over-estimate the AET, resulting in under-utilisation of computational resources.

The main contribution of this paper is an offline/online multistage optimisation approach for deployment and execution of data-streaming applications on many-core platforms, shown in Figure 1. We assume, as in [5], that task WCET is composed of the Worst-Case Computation Time (WCCT) when executed in isolation, which includes delays to load/store data, and the delays due to *interferences*. In the offline phase, similarly to [6], [7], we first perform the application i) *partitioning*, by balancing the computation load using the WCCT, and ii) *placement*, by assigning different partitions to compute clusters while minimising the communication overhead. Subsequently, a *unified system model* is derived, which captures the operation of both the computation and communication elements of the platform. Then we over-approximate the WCETs for tasks of the unified system model and use these WCETs to derive near-optimal solutions for the application *mapping, scheduling* and *buffer allocation* similar to [6], [7], though providing real-time guarantees. The solutions to the optimisation problems are provided by SMT solvers, via sets of constraints describing the unified system model and limitations of the target platform. These solutions are then analysed using our interference-based WCET analysis of [5] acquiring tight WCET and thus tighter latency guarantees. This novel stage-by-stage approach to WCET analysis breaks the cyclic dependency and provides near-optimal deployment solutions with real-time guarantees. In the *online* phase we introduce a novel distributed adjustment of the offline schedule, based on the AET known at run-time, which we prove to preserve the guarantees of the *offline* phase.

The research work of this paper was funded by the Swiss Confederation through the UltrasoundToGo project of the Nano-Tera.ch initiative.

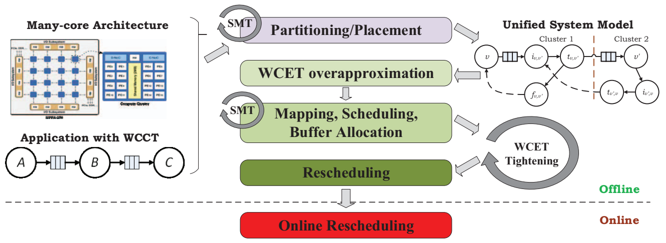

Fig. 1: Overview of the approach.

We evaluate the approach by deploying a set of StreamIt [2] applications on the Kalray MPPA-256 many-core platform [8] finding that the offline solutions provide a tightening of the guaranteed latency up to 46% and are within 18% of the best-effort approach [7]. The online optimisation can offer a further latency reduction up to 41%, with a combined improvement of more than 50%. We show that the Pareto fronts of the solutions with real-time guarantees and of the best-effort approach [7] are close, indicating that the near-optimal performance of the solutions in [7] is preserved.

This paper is organised as follows: Section II discusses related work. Section III presents the models and theory used in the proposed approach. Section IV gives the details of the *offline/online* optimisation flow and proves its safety. In Section V the experimental results are presented.

II. Related work

Optimal deployment of an application, which may have data dependencies, on a many-core platform is often regarded as a multi-criteria optimisation problem [6], [7], [9], [10], [11], [12]. For instance in [7], [9], the authors proposed to divide such an optimisation problem into several stages. However, the approach is focused on achieving performance and not on providing real-time guarantees, since the optimisation steps are performed based on the Average Case-Execution Time (ACET) for the computation tasks. We are focusing on providing real-time guarantees to a similar optimisation problem.

Most works that address time guarantees for such problem either assume that the WCET is given [13], [14], or estimate the WCET based on some given deployment onto an architecture [15], [12]. Some of them, e.g. [16], assume that tasks have priorities which makes WCET analysis more predictable. In [13], the authors present an optimal scheduling algorithm for periodic task execution on multi-cores, which has bounded overhead. Similarly, the authors of [14] present an optimal scheduling algorithm that bounds the number of migrations/preemptions, by reducing the problem to uniprocessor scheduling. We consider that the overhead of online scheduling is not always covered by the gain of avoiding migrations, as the application size grows. The authors of [17], [18] present an approach to deploy an application, modelled as Synchronous Data-Flow (SDF) model, onto a multi-core platform. In line with our approach, the deployment solutions are derived using a constraint programming algorithm, however, they do not provide real-time guarantees. In [19], the deployment of SDF graphs is also addressed using parallel simulated annealing. Such works suggest that our approach is also parallelisable, when acquiring incomparable solutions. The authors of [20] explore the data-/pipeline-parallelism trade-off of data-flow applications, while optimising buffer sizes. This research is an interesting direction for our future work. Other papers focus mainly on the WCET estimation of tasks with data dependencies deployed on multi-[21], [22] and many-core [5], [23] architectures. Works on HW/SW co-design, e.g. [12], can avoid interferences by choosing the number of resources, contrary to the commercial hardware that we consider here.

In [24], the authors present a comprehensive theory for mixed-criticality scheduling on cluster-based many-core architectures with shared resources. To derive a feasible schedule the authors estimate the tasks' worst-case response time (WCRT), which we call WCET. Similarly to us, they consider that the WCRT, for the same criticality level, is composed of a worst-case execution time (i.e. WCCT in our paper) and the total delay due to contention on shared resources. Our model considers no criticality levels but has a detailed representation of the communication mechanism over the NoC. Also, in addition to the arbitration delays when accessing shared memory blocks accounted in [24], we consider the arbitration delays occuring at shared buses. Moreover, we are tightening the WCET by excluding non-interfering tasks, i.e. tasks non-overlapping in time or sharing a resource, which is highly complex for the models used in [24].

III. Background

In this section we present the models for many-core platforms and applications considered in our analysis. Using as input a platform and an application model we first acquire the *partitioning/placement* function and then build the *unified system model*, which is further used to derive the set of functions defining the application deployment on the target platform, i.e. *task mapping*, *buffer allocation* and *scheduling*.

A. The platform architecture model

Similarly to [5], [7] we consider a generalised many-core architecture as a tuple $P = (X, K, M, N)$ where X is the set of identical *clusters* interconnected with a NoC and K, M, N are the sets of processing *cores*, *memory banks* and *NoC channels* per cluster, respectively. Each cluster has one NoC interface, connected to a dedicated NoC router via the NoC channels [25]. The *cores* of each cluster exchange data via the cluster *shared memory*. Data transfers among cores located on different clusters are handled by NoC. This model is inspired by Kalray MPPA-256 [8], but is generic enough to model other platforms. In this work we focus on providing timing guarantees for application execution assuming that necessary input data are already placed in the shared memories.

1) Memory access model: The cluster memory is organised in several memory banks accessible through data buses. For any memory operation there are two arbitration points: i) one to access the data bus, which connects a core to the memory banks, and ii) another one to access the target memory bank. Data load/store from/to the memory is performed as a sequence of word-by-word memory operations, called *requests*. For a task with no interference, the memory access delay is accounted in its WCCT. However, if two requests collide on the bus arbiter and/or memory arbiter, an *interference* delay proportional to the number of colliding requests is added.

2) NoC model: The NoC is composed of a set of routers, connected in a topology. Each cluster has a dedicated *NoC interface* [25], connected to a dedicated router with several channels to handle multiple data flows in parallel. Transmitting/receiving data are handled by two dedicated elements, thus avoiding interference between these two operations. An inter-cluster data transfer between two tasks v, v' occurs in three distinct steps. At the *initialisation* step, a core configures the NoC interface with memory locations used to load/store data and the assigned NoC channel to be used. At the *transfer* step, the source NoC channel fetches the data, forms the packets and forwards them to the target cluster over NoC. The target NoC interface receives the data and places them in memory. At the *finalisation* step, the core that initiated the

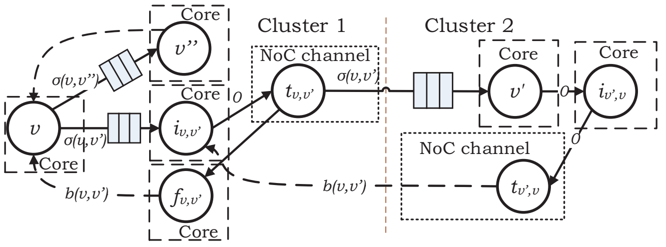

Fig. 2: Unified system model for three tasks v, v', v'', such that $(v,v'),(v,v'') \in E$ and v, v' placed on different clusters.

transfer polls the NoC interface to check if the transfer has finished, so as to release memory space. We model these three steps as *communication tasks* $i_{v,v'}, t_{v,v'}$ and $f_{v,v'}$ in the *unified system model* (Sec. III-C). The NoC model accounts for some platform-dependent parameters, which constitute the WCET of the *communication tasks*. We assume that they are known and bounded for a given platform P.

B. Application model

We model an application comprised of tasks with data dependencies as a directed acyclic graph $G = (V,E)$, with V being a set of *computation tasks* and E a set of dependencies among these tasks. Tasks communicate via bounded FIFOs, one for each $e \in E$ of the graph. An *application model* is an annotated task graph $A = (V,E,d,\sigma)$ where:

- $d : V \to \mathbb{N}_0$ is the *delay function*, representing the WCCT of a task v when executed in absence of interferences;
- $\sigma : E \to \mathbb{N}_0$ is the *data-size function* denoting the *amount of data* (in words) of a data dependency $(v,v') \in E$, i.e. the amount of data produced by v and consumed by v'.

C. Unified System Model

A faithful model for the execution of an application on a target platform should account for the NoC transfers. These are only known after solving the *partitioning/placement* problem. We model such a solution with the *placement function* $p : V \to X$, that assigns a task $v \in V$ to a cluster $x \in X$.

Given the application model A and the placement function p, we transform A into a *unified system model* S, according to [7] (see Figure 2). For each pair of dependent tasks (v,v') placed to different clusters, i.e. $p(v) \neq p(v')$, additional *communication tasks* are introduced to model the NoC initialisation, transfer and finalisation denoted with $i_{v,v'}$, $t_{v,v'}$, and $f_{v,v'}$, respectively. It must be noted that for any pair of tasks communicating via the NoC, two FIFOs must exist in two different clusters, for the writer v and reader v'. To model the bounded nature of FIFOs, for each pair of dependent tasks, as in [7], we add additional backward edges that model the capacities of the FIFOs.

This model faithfully describes the execution of the application A on a platform P by accounting for both computation and communication behaviour of the system. Since this model is an SDF graph, a valid scheduling solution guarantees that no FIFO overflows [7]. However, this model does not contain information about task mapping, execution order or the allocated FIFO sizes. For this reason we introduce the *task mapping*, *scheduling* and *buffer allocation* functions which constitute the solutions of the corresponding problems.

Given a unified system model S, a platform model P and a placement function p we define the *task mapping* function $p\mu_E : V \to X \times (K \cup N)$, which maps tasks to platform elements, i.e. either cores or NoC channels. In addition, with $s : V \to \mathbb{N}_0$ we denote the *scheduling* function which associates a start time to each task, such that for any pair of tasks v, v' mapped on the same platform element $\varepsilon \in X \times (K \cup N)$, the inequality $s(v) + d(v) < s(v')$ holds. A platform element can execute only one task at any time instance; thus in a correct solution, tasks mapped to the same element must be strictly ordered according to the scheduling function s. The set of backwards edges E_B of the unified system model is annotated with the buffer allocation function $b : E_B \to \mathbb{N}_+$, which models the FIFO capacity for the FIFO between two tasks v and v'.

IV. Safe deployment and execution

The optimal deployment of an application on a platform is a multi-criteria optimisation problem that is solved to find task *partitioning/placement*, *mapping*, *scheduling* and *buffer allocation* affecting the total execution latency. In our approach, similarly to [6], [7], [9] we use an SMT solver, which can be replaced by any equivalent optimisation method. The method starts from an initial solution and iteratively queries the SMT solver for a strictly better solution. When such solution cannot be found, the query is then modified to find incomparable solutions, thus exploring the whole cost space. In our case, we bound the cost space by the serial execution latency, i.e. the sum of all WCCTs of the application tasks, and by the total memory of the target platform.

A. Offline optimisation

Providing strong latency guarantees requires that task WCET is known. The parallel execution of tasks introduces mutual interferences when those tasks share resources, i.e. buses/memory banks. We focus on two types of interference:

- **Intra-cluster:** in which any two tasks v, v', running on separate platform elements of the same cluster, compete for the same resource, i.e. bus and/or memory bank.
- **Inter-cluster:** in which any task v and a transfer task t, of another cluster, try to access the same memory bank.

Interference can also occur on the NoC, but we assume that the WCET of transfer tasks is given by the platform via NoC guaranteed services, like in [26]. This assumption can be lifted by an interference analysis for the NoC, similar to [5].

The solutions of *partitioning/placement*, which balance the computation load of clusters and minimise the communication overhead, are modelled as a set of placement functions p. Based on each solution we derive a unified system model (as in Sec. III-C) and analyse it to derive conservative WCETs.

Given a unified system model S, let $\mathcal{P}_v$ be a set of tasks potentially executed in parallel with v:

$$\mathcal{P}_v = \left\{ v' \in V \mid v' \notin pred^*(v) \cup succ^*(v) \right\}$$

where $pred^*(v), succ^*(v)$ is the transitive closure of predecessors and successors of v. Let $\mathcal{P}_{vx}$ be the set of these tasks placed on cluster x and $T_{\overline{x}}$ the set of transfer tasks placed on any cluster other than x; given a task v placed onto a cluster $x \in X$ we derive the over-approximated worst-case delay function d_{wc}^{over} (WCET), similarly to [5], [23], [24], by adding up the interferences caused by all potentially parallel tasks:

$$d_{wc}^{over}(v) \overset{def}{=} d(v) + if_{mem}(v, \mathcal{P}_{vx} \cup (\mathcal{P}_v \cap T_{\bar{x}})) + if_{bus}(v, \mathcal{P}_{vx})$$

where if_{mem} and if_{bus} are functions that account for interference delays introduced to the task v by other tasks when sharing memory banks and buses, respectively. These functions compute the interference delays based on the arbitration policies of the target architecture as in [5].

Further, for each unified system model, using the over-approximated WCET, we acquire a set of solutions for the *mapping*, *scheduling* and *buffer allocation* problems, i.e. $(p\mu_E, s^{over}, b)$, approximating the Pareto front for the buffer-latency trade-off. We do this by querying the SMT solver for a valid deployment, i.e. for every dependent task pair (v, v') mapped on the same platform element, task v' starts after its predecessor v. Or more formally:

$$\forall (v, v') \in E: \bigwedge_{p\mu_E(v) = p\mu_E(v')} s^{over}(v) + d_{wc}^{over}(v) < s^{over}(v')$$

For the sake of space, we omit the rest of the constraints, e.g. respecting initial application data dependencies, not exceeding the total memory or the number of processing elements, etc. These solutions are safe and have a guaranteed latency, since they are based on over-approximated WCETs.

All solutions are then analyzed using our interference-based WCET analysis method [5] to tighten the task WCETs, denoted as d_{wc}^{tight}. The WCET tightening is achieved by excluding tasks from the set P_v that cannot possibly access the same resource at the same time. Finally, for each solution, the schedule function s for each task is updated to the earliest possible time such that no new interference is introduced. Both WCET tightening and rescheduling are proven to preserve safety guarantees, in our previous work [5].

We then transform the unified system model into the *scheduled system model* $DS = (V, E \cup E_S, \sigma, d_{wc}^{over})$, by adding additional *scheduling dependencies* E_S. These dependencies enforce that for any two non-overlapping tasks v, v' (i.e. $s(v) + d(v) < s(v')$), that share a resource, task v' must be executed only after v has finished, i.e. $v \rightarrow v'$. This, transformation guarantees that if tasks are rescheduled to earlier times, no new interference will be introduced (see Figure 3).

Subsequently, the deployment solution $(p\mu_E, s^{over}, b)$ is transformed to $(p\mu_E, s^{tight}, b)$, by rescheduling the tasks as early as possible according to the tightened WCET, while preserving the order of execution, as follows:

$$s^{tight}(v) = \max_{v' \in pred(v)} \{s^{tight}(v') + d_{wc}^{tight}(v')\}$$

with $s^{tight}(v) = 0$ for tasks with no predecessors.

The new deployment solution $(p\mu_E, s^{tight}, b)$ is safe by construction [5], with the improved total guaranteed latency of the scheduled system model $DS = (V, E, \sigma, d_{wc}^{tight})$ computed as $\max_{v \in V}\{s^{tight}(v) + d_{wc}^{tight}(v)\}$. This new scheduled system DS serves as the input to the online optimisation stage.

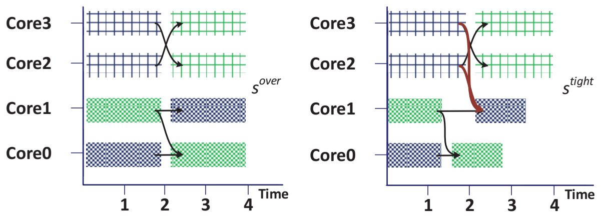

Fig. 3: Example of schedule s^{over} with data dependencies (black arrows) and s^{tight} with scheduling dependencies (red arrows) for one cluster with four cores; same patterned/colored tasks use the same bus/memory bank, respectively. The green task on Core0 is rescheduled, but the blue task on Core1 is not, as it would introduce new interference to Core2 and Core3.

B. Online optimisation

Task WCETs, however, can still be over-estimated, which results in non-optimal use of resources. Therefore, we introduce the *online* stage, where a novel optimisation based on the AET of tasks, known at run-time, is performed. This online optimisation resembles self-timed scheduling with the difference that the underlying *scheduled system model* guarantees by construction that rescheduling to earlier times will not introduce new interference. Each core has a light-weight run-time monitor invoked when one of its tasks finishes its execution and performs several updates.

For a task v that finished its execution earlier than planned, i.e. at time $t < s^{tight}(v) + d_{wc}^{tight}(v)$, the run-time observer computes the delay function d^{run} reflecting the AET.

$$d^{run}(v) \overset{def}{=} t - s^{tight}(v) \quad \leq d_{wc}^{tight}(v)$$

Further, the starting time of each immediate successor of task v is updated iff the successor is ready to be executed.

$$\begin{matrix}\forall v' \in succ(v). \\ \text{s.t. } ready(v')\end{matrix} : s^{run}(v') = \max_{v'' \in pred(v')} \{s^{run}(v'') + d^{run}(v'')\}$$

Notice that, in a *scheduled system model*, a task is considered to be *ready* if both data and scheduling dependencies are met. The observers are implemented such that they have small and bounded execution times (few hundreds of cycles) and memory requests (max. 16). We account for this overhead even though we omit it from the formal notation for clarity reasons.

C. Safety

Theoretical results [5], [7] show that the deployment solution $(p\mu_E, s^{tight}, b)$ is safe, i.e. guarantees that no deadlines will be violated and that no buffers will overflow. Therefore, we must ensure that the online optimisation, applied to *scheduled system model* $DS = (V, E, \sigma, d_{wc}^{tight})$, preserves these guarantees. For space reasons, we provide an informal reasoning.

a) Buffer overflow protection: The fact that no buffer overflow will occur due to the online optimisation is enforced by construction since the scheduling order is preserved, and thus none of the tasks will attempt to write to their FIFOs before the corresponding memory space is available.

b) Preserving real-time guarantees: The scheduled system model guarantees that no new interference will be introduced if ready tasks, i.e. with all dependencies met, are rescheduled to earlier times. Given that any task $v \in V$ finishes its execution at $t = s^{run}(v) + d^{run}(v)$, to prove that deadlines are respected, it is sufficient to show that none of the tasks $v \in V$ will be postponed, i.e. $s^{run}(v) \leq s^{tight}(v)$, since $d^{run}(v) \leq d_{wc}^{tight}(v)$. For tasks with no predecessors it holds trivially. Tasks with predecessors are scheduled to start immediately after their predecessors have finished. Since the AET d^{run} is not greater than the WCET d_{wc}^{tight}, intuitively the "first batch" of tasks cannot be postponed, neither the "second", etc., thus all tasks will meet their deadlines. This can be proven by induction.

Rescheduling the communication tasks may introduce additional overhead on the NoC. However, this is inconsequential since our target architecture provides guaranteed NoC services, which account for any interference and provide an upper bound to the WCET of transfer tasks.

V. Experimental evaluation

To experimentally evaluate our approach we use a subset of StreamIt [2] benchmark and a JpegDecoder. The benchmark set used for the experiments consists of 8 distinct applications. One application, i.e. Discrete Cosine Transformation (DCT), is modeled in 10 different ways, varying the level of task parallelism. The analysed solutions vary in size, containing from 3 up to 200 tasks. In all figures, the benchmarks are in increasing order of average number of tasks, in parentheses.

The benchmarks were deployed on Kalray MPPA-256 platform (ver 2.5) with the implementation of our optimisations. We acquired task WCCTs by profiling on a single core of the platform. Caches and prefetch buffers were disabled to avoid WCCT underestimations due to cache-hits and prefetches.

To acquire safe deployment solutions, we adapted the StreamExplorer tool which provides a set of near-optimal deployment solutions $(p\mu_E, b, s)$ using SMT solvers. Each query to the SMT solver has a time-out of 30 sec. and for each benchmark a total time-out of 6 min.

We evaluate each stage of our approach in terms of achieved latency reduction. For each application, a set of total guaranteed latencies is computed from the set of solutions $(p\mu_E, b, s^{over})$ representing the Pareto front of the over-approximated WCET d_{wc}^{over}. Next, the WCET of all tasks is improved, separately for each solution, using the interference-based WCET analysis [5]. A second set of improved total guaranteed latency is then computed after rescheduling the tasks as early as possible, i.e. s^{tight}, according to the tightened WCET d_{wc}^{tight}. Finally, to acquire the third set of latencies improved in the *online* stage, each deployment is realised and executed 100 times on the platform, measuring the AET d^{run}. Notice that we use the whole set of solutions of the Pareto front, and not the best-latency solution only.

In Figure 4, we present the latency improvements achieved from the offline and online optimisations. The height of the column of each benchmark represents the total latency gain (in %), moving from over-approximated WCET, through

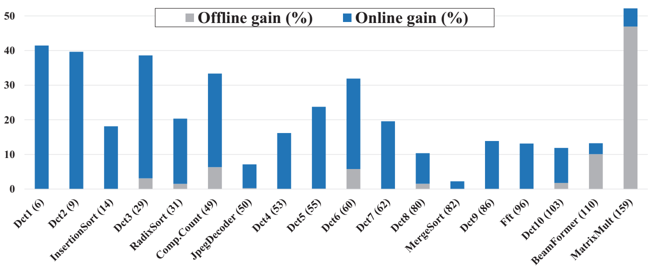

Fig. 4: Guaranteed (offline) and runtime(online) latency gain

tightened WCET, to AET. The guaranteed latency tightening achieved at the offline phase varies significantly. For some cases, the guaranteed latency tightening is less than 1%. The reason is two-fold. Some benchmarks are rather small, thus the resulting deployment has no interferences, e.g. InsertionSort, DCT1-2, but most importantly in other benchmarks, such as MergeSort or JpegDecoder, the interfering tasks are not in the critical path of the schedule, i.e. any increase of these tasks execution time does not necessarily increase the total latency. The MatrixMult benchmark is a special case, since it uses simple computational operations but requires that a significant amount of data is shared among tasks. This results in having many parallel tasks accessing the same memory bank, and thus dramatically increases the mutual interference. We conclude that in several cases the interference introduced by the parallel execution of tasks can have a significant impact on the WCET estimation, and an accurate interference-based analysis is necessary. Nevertheless, we achieve a very significant average tightening of guaranteed latency 4.3%, up to more than 45%.

In addition, the online rescheduling achieves a significant latency reduction, as seen in Figure 4. The latency reduction ranges from 3% to 41.5%. The reason for this variation is that the WCCT, and thus the WCET, are still over-approximations which the online optimisation aims to compensate for. This is most apparent in the cases that exhibit no improvement from the offline stage (e.g. DCT1-2, FFT, etc.), which benefit the most. The low latency reduction on some applications (e.g. MergeSort, Beamformer) is justified by the fact that their tasks have a small number of different execution paths, thus their WCCT is quite accurate, and their deployment exhibits no interference or the interference has been already excluded by the offline phase.

These results present interesting findings, as they confirm the need for both offline and online optimisations. For the offline optimisation the gain in guaranteed latency can go up to 46%, while the latency gain achieved by the online optimisation is approximately 18%, on average, and can reach more that 40%. Their combined gain is 22% on average and can be of more than 50%.

We also compare the Pareto front of the deployment solutions of our offline phase $(p\mu_E, b, s^{tight})$, i.e. computed with over-approximated WCET and then tightened, with the Pareto front of deployment solutions based on the ACET, i.e. derived from the best-effort approach of [7]. In Figure 5, these

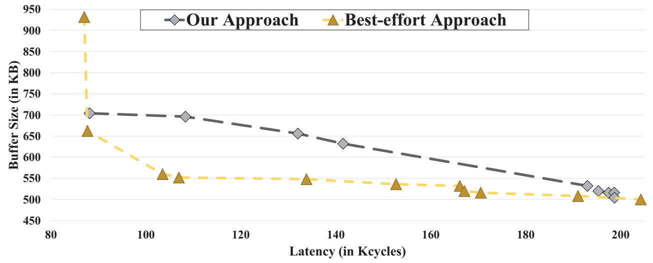

Fig. 5: Acquired Pareto-fronts (latency, buffer-size) for Beamformer, using as input ACET and over-approximated WCET

Pareto fronts are presented for the Beamformer benchmark. Notice that they are not far apart. For approximately the same latency, our method generates a solution which requires at most 150 $KBytes$ more memory compared to the ACET-based solutions. Similarly, for approximately the same buffer size, the latency of the WCET-based solutions, is at most 18% greater than the best-effort ones, while guaranteeing that deadlines are met. For all applications, the best latency for WCET solutions was $< 2\%$ greater than the best-effort ones.

Another important fact is that the Pareto fronts have different shapes. The one computed using the WCET tightening technique is more linear, stretched along the latency axis. This is due to the word-by-word mechanism to access the memory on Kalray MPPA-256 used in our experiments. As a result the number of interferences on memory accesses grows with the ammount of memory required to run the benchmark. Therefore, the tightening optimisation is more effective, i.e. the more memory required, the larger the latency reduction, which leads to this stretch along the latency axis.

VI. Conclusion

In this paper we describe an offline/online approach to efficiently deploy data-streaming applications on many-core architectures while providing real-time guarantees. At the offline phase, we brake the cyclic dependency between task WCET estimation and deployment solution, thus acquiring tight WCET and latency guarantees by excluding interference delays. At the online phase, we adapt the execution of tasks to their AET while guaranteeing that no new interference is introduced, and further reducing the total latency. Experimental results on Kalray MPPA-256 show a latency reduction in the offline phase up to 46%, in the online phase up to 41% and an overall reduction up to more than 50%.

References

[1] J. Combaz, J.-C. Fernandez, T. Lepley, and J. Sifakis, "QoS control for optimality and safety," in *EMSOFT'05*, pp. 90–99, ACM, 2005.

[2] W. Thies and S. Amarasinghe, "An empirical characterization of stream programs and its implications for language and compiler design," PACT '10, (New York, NY, USA), pp. 365–376, ACM, 2010.

[3] G. Kahn, "The semantics of a simple language for parallel programming," in *Information processing*, Aug 1974.

[4] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[5] S. Skalistis and A. Simalatsar, "Worst-case execution time analysis for many-core architectures with NoC," in *International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS*, 2016.

[6] S. Cotton, O. Maler, J. Legriel, and S. Saidi, "Multi-criteria optimization for mapping programs to multi-processors," in *IEEE International Symposium on Industrial Embedded Systems (SIES)*, IEEE, 2011.

[7] P. Tendulkar, P. Poplavko, I. Galanommatis, and O. Maler, "Many-core scheduling of data parallel applications using SMT solvers," in *Digital System Design (DSD), 2014 17th Euromicro Conference on*, IEEE, 2014.

[8] Kalray, "Kalray MPPA-256," 2015.

[9] P. Tendulkar, P. Poplavko, and O. Maler, "Symmetry breaking for multi-criteria mapping and scheduling on multicores," in *Formal Modeling and Analysis of Timed Systems*, pp. 228–242, Springer, 2013.

[10] I. Kadayif, M. Kandemir, and U. Sezer, "An integer linear programming based approach for parallelizing applications in on-chip multiprocessors," in *Proceedings of the 39th Annual Design Automation Conference*, DAC '02, (New York, NY, USA), pp. 703–706, ACM, 2002.

[11] J. Legriel, C. Le Guernic, S. Cotton, and O. Maler, "Approximating the pareto front of multi-criteria optimization problems.," in *TACAS*, pp. 69–83, Springer, 2010.

[12] R. P. Dick and N. K. Jha, "Mogac: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 920–935, 1998.

[13] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*, pp. 101–110, IEEE, 2006.

[14] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, "Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor," in *Real-Time Systems Symposium (RTSS)*, pp. 104–115, IEEE, 2011.

[15] A. Weichslgartner, D. Gangadharan, S. Wildermann, M. Glaß, and J. Teich, "Daarm: Design-time application analysis and run-time mapping for predictable execution in many-core systems," in *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, CODES '14, pp. 34:1–34:10, ACM, 2014.

[16] M. Bertogna, *Real-time scheduling analysis for multiprocessor platforms*. PhD thesis, 2008.

[17] A. Bonfietti, M. Lombardi, M. Milano, and L. Benini, "Throughput constraint for synchronous data flow graphs," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, vol. 5547 of *Lecture Notes in Computer Science*, 2009.

[18] 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 *Proceedings of the Conference on Design, Automation and Test in Europe*, European Design and Automation Association, 2010.

[19] F. Galea and R. Sirdey, "A parallel simulated annealing approach for the mapping of large process networks," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pp. 1787–1792, IEEE, 2012.

[20] J. P. Hausmans, S. J. Geuns, M. H. Wiggers, and M. J. Bekooij, "Unified dataflow model for the analysis of data and pipeline parallelism, and buffer sizing," in *Formal Methods and Models for Codesign (MEMOCODE), International Conference on*, IEEE, 2014.

[21] V. A. Nguyen, D. Hardy, and I. Puaut, "Scheduling of parallel applications on many-core architectures with caches: bridging the gap between WCET analysis and schedulability analysis," in *9th Junior Researcher Workshop on Real-Time Computing* , Nov 2015.

[22] P. Burgio, A. Marongiu, P. Valente, and M. Bertogna, "A memory-centric approach to enable timing-predictability within embedded many-core accelerators," in *Real-Time and Embedded Systems and Technologies (RTEST), CSI Symposium on*, 2015.

[23] A. Dkhil, S. Louise, and C. Rochange, "Worst-Case Communication Overhead in a Many-Core based Shared-Memory Model," in *Junior Researcher Workshop on Real-Time Computing, Nice*, octobre 2013.

[24] G. Giannopoulou, N. Stoimenov, P. Huang, L. Thiele, and B. de Dinechin, "Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources," *Real-Time Systems*, pp. 1–51, 2015.

[25] É. Cota, A. de Morais Amory, and M. S. Lubaszewski, *Reliability, Availability and Serviceability of Networks-on-chip*. Springer Science & Business Media, 2011.

[26] B. D. de Dinechin, D. van Amstel, M. Poulhiès, and G. Lager, "Time-critical computing on a single-chip massively parallel processor," in *Conference on Design, Automation & Test in Europe*, DATE '14, 2014.