Fault-Tolerant Deployment of Dataflow Applications Using Virtual Processors

Reinier van Kampenhout*, Sander Stuijk* and Kees Goossens*†
*Eindhoven University of Technology, the Netherlands
{j.r.v.kampenhout,s.stuijk,k.g.w.goossens}@tue.nl
†Topic Embedded Products, the Netherlands

***Abstract*—Multi-processors are suited to host a dynamic mix of real-time dataflow applications, but are increasingly subject to faults because of the decreasing feature size. Applications can start and stop as needed if they execute on a private set of Virtual Processors (VPs) that are deployed on the physical processors. This allows online software updates, but makes it impossible to predict the deployment. If a fault renders a processor unusable, the free resources on other processors may be too fragmented to allow its VPs to be re-deployed. We show that mapping an application to more VPs reduces the maximum VP size. This increases the probability of successfully dealing with faults, at the cost of an increase of the total size. Such a mapping can either be run from the start, or we can split the VPs only when a fault occurs. Experiments confirm the feasibility of our approach, and show a trade-off between improved fault-tolerance and resource usage for both strategies.**

I. Introduction

Advanced control systems such as driving assistance in automotive require processing of high-bandwidth data streams from sensors and external sources. Processing steps with data-dependent execution times such as filtering and feature recognition must be performed in real-time (RT), because control algorithms depend on their output. These algorithms start and stop over time and contain repetitive operations that can benefit from hardware accelerators. Heterogeneous multi-processor platforms where general-purpose (GP) processors are supported by accelerators (ACC) are good candidates to host such a dynamic mix of demanding applications.

The continuous decrease of the feature size in VLSI design leads to an increased power density, which translates to higher local temperatures or hot spots. These cause intermittent faults and processor shutdowns on the short term, and speed up the silicon aging process which leads to permanent faults on the long term [1], [2]. It is costly to add circuitry for fault correction. Instead we propose a method for, and analysis of, fault-tolerant deployment of applications to handle intermittent and permanent processor faults.

Our contributions are valid for each *Model of Computation (MoC)* that allows the design of data-dependent RT applications. One such a MoC is *dataflow*, which we will use as a running example throughout this work [3]. We consider a design flow in which clusters of dataflow *actors*, the nodes of a dataflow graph, are mapped to *Virtual Processors* (VPs) at design-time, see application α on the left side of Figures 1a-c. We consider heterogeneous platforms that are Globally Asynchronous, Locally Synchronous (GALS) [4].

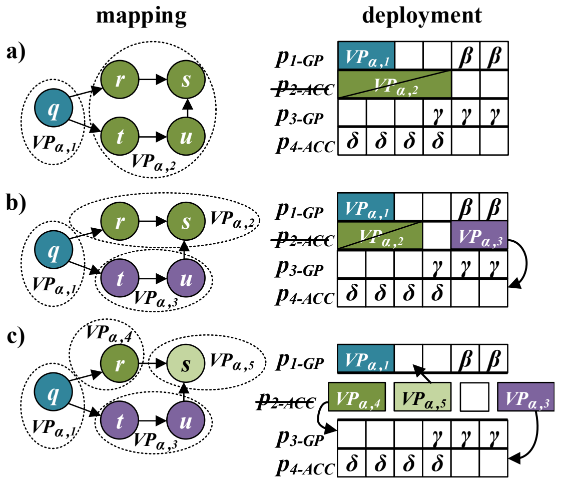

Fig. 1: Application α with actors $q..u$ is mapped to an increasing number of VPs (left) which are deployed on processors p_1, p_2 (right). When p_2 encounters a fault:
a) $VP_{\alpha,2}$ cannot be re-deployed because of its size and type;
b) $VP_{\alpha,2}$ cannot be re-deployed because of its type, but $VP_{\alpha,3}$ can be re-deployed on p_4;
c) we switch to another mapping at runtime in which $VP_{\alpha,2}$ is split and resized for a GP, and re-deployed on p_1 and p_3.

At runtime the VPs of an application may be *deployed* on any physical processor p_i with sufficient free capacity using Time Division Multiplexing (TDM) arbitration, see the right side of Figures 1a-c [5]. This combination of design-time mapping and runtime deployment ensures that applications can be started and stopped at any time independent of the current deployment, as long as sufficient capacity is available. Therefore applications can be added at runtime, which enables online software updates. Deployment is an instance of the bin-packing problem, and it is impossible to predict what the distribution of VPs will be at a given time. When a fault occurs the free capacity may be too fragmented to allow re-deployment, see the right side of Figure 1a.

Our first contribution is improved fault-tolerance through a design-time mapping strategy. By mapping each application to more VPs of a smaller maximum size the probability of successful re-deployment is increased, at the cost of an increase in total size. Consider the improvement of the deployment in Figure 1b over that in 1a. The total size of application α has

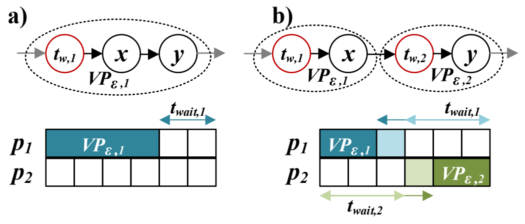

Fig. 2: a) $VP_{\epsilon,1}$ has a waiting time $t_{wait,1}$ of 2 slots, captured by actor $t_{w,1}$ b) After splitting naively into $VP_{\epsilon,1}$ and $VP_{\epsilon,2}$ (dark blue and dark green) the waiting time is 4 slots for both. The throughput constraint is not satisfied, so additional slots must be reserved (light blue and light green) to reduce the actor WCRT.

increased from 6 to 7, but $VP_{\alpha,3}$ can now be re-deployed on p_4. While $VP_{\alpha,3}$ would fit on GP processor p_3, it is targeted for an ACC and cannot be re-deployed.

Our second contribution solves this by *splitting* and *resizing* VPs at design-time, and switching to that new mapping at runtime. As shown on the left side of Figure 1c, $VP_{\alpha,2}$ is further split into $VP_{\alpha,4}$ and $VP_{\alpha,5}$ that are targeted for a GP. The total size of that mapping increases to 8 slots because the execution time of the actors is higher on a GP. When the fault occurs we switch to the new mapping, and $VP_{\alpha,4}$ and $VP_{\alpha,5}$ can be re-deployed on p_3 and p_1 respectively. The other VPs are not affected by the split and can continue without interruption.

The trade-off between improved fault-tolerance and resource usage is evaluated for the different strategies in Section VI. We find that the platform utilization at the moment that the fault occurs plays an important role. There are two thresholds on this utilization. Below the lower threshold a re-deployment can always be found with every strategy, while above the higher threshold no deployment can be found with any strategy. The area of interest is in between these thresholds, where the strategies help to maximize fault-tolerance for the highest possible platform utilization.

Mapping dataflow actors to VPs is detailed in Section II, the deployment of VPs to the platform in III. The fault model is presented in Section IV, followed by our contributions in Section V. The experiments in Section VI confirm the validity of our contribution. We discuss related work in Section VII, and present the conclusions in Section VIII.

II. Mapping Dataflow Graphs

Our contributions can be applied to any RT MoC that can guarantee a minimum throughput or maximum latency given the actor (or task) worst-case execution time (WCET) and a method to compute the resulting worst-case response time (WCRT) of the application. To illustrate this we use the dataflow MoC, which can guarantee a minimum bound on the throughput given the actor WCET by means of computing the resulting WCRT of the application [6], [7]. The contributions trade fault-tolerance against the size of the applications, measured in TDM slots. To provide insight in this cost trade-off we now take a detailed look at the design-time, intra-application *mapping*.

Dataflow actors are mapped to VPs. The size of each VP is determined by calculating the WCRT of each path through the dataflow subgraph that is mapped to the VP. Because we consider GALS systems, the TDM wheels of the physical processors may be misaligned. As each VP may be deployed on a different physical processor, the WCRT must account for the worst-case TDM wheel misalignment between VPs.

Consider for example application ϵ in Figure 2a. Actors x and y are mapped to $VP_{\epsilon,1}$, which uses 4 TDM slots out of a total wheel size of 6. The waiting time $t_{wait,1}$ is two slots, which is captured by the red actor $t_{w,1}$ that is inserted in front of the VP during timing analysis. The WCRT for $VP_{\epsilon,1}$ thus consists of the sum of $t_{wait,1}$ and the actor execution times.

The essence of our first contribution is to increase the number of VPs an application is mapped to. We split $VP_{\epsilon,1}$ into $VP_{\epsilon,1}$ and $VP_{\epsilon,2}$ as shown in Figure 2b. If the slot allocation of the new VPs would solely be based on the ET, both receive two slots as indicated by the dark blue and dark green slots on p_1 and p_2 respectively. The waiting time is now 4 slots for both VPs. This may lead to a violation of the throughput requirement. To counter this, we can reduce the WCRT by reserving additional TDM slots, as indicated by the light blue and light green slots in the figure. Note that no actual work will be performed in those slots, they are necessary because of the assumed worst-case TDM wheel misalignment. The extra slots are required to reduce the latency and increase the throughput. We see that the application now requires 6 slots in total, and may still be executed on one physical processor.

III. Deployment of Virtual Processors

The intra-application mapping of actors to VPs described in Section II is performed at design-time. At runtime the VPs must be assigned to and started on a physical processor, through an inter-application *deployment*. This two-layer design approach has the advantage that the compute intensive timing analysis and mapping are only performed at design-time. No further calculations apart from the deployment itself are necessary at runtime.

Analysis can only bound the throughput of an application if VPs are deployed on a *predictable* platform. An architecture is predictable if the WCRT of each actor can be captured at design-time. Any predictable real-time platform such as the Time-Triggered Architecture (TTA) or CompSOC is suitable for our method [8], [5]. In this work we select the latter to illustrate our contributions.

On the CompSOC platform timing interference between applications is prevented by allocating a budget for each application on all hardware resources such as processors, memories, and the Network-on-Chip (NoC). Processor virtualization is of specific interest in this work. Applications are mapped to one or more VPs, which are deployed using TDM arbitration. As the timing analysis accounts for the worst-case TDM wheel misalignment between physical processors, the VPs of an application can be deployed on any processor of the correct type. This is a requirement for GALS systems, and has the advantage that new applications that have been analysed offline can be added to the platform at runtime without re-analysis, thus separating mapping from deployment. Such online software updates are becoming standard practice in many domains such as consumer electronics, medical and automotive. We use this feature at no extra cost to achieve fault-tolerance by re-deploying VPs in case of a fault, see Section V.

Deployment is an instance of the one-dimensional bin-packing problem, which is known to be NP-hard [9]. Use of heuristics is inevitable to find a solution at runtime. It is intuitive that the probability of success increases if the heuristic is provided with more but smaller items to fill the bins, *if* the sum of item sizes is the same. The latter is not true, as we will see in Section VI. Instead the sum of the items (VPs) size grows if an application is mapped to more VPs. We can only expect a benefit if the increase in probability of success caused by the smaller VPs outweighs the decrease caused by the growth in total size.

Deployment of VPs on physical processors at runtime must be performed by a resource manager that is capable of dynamic loading [10]. In this work we will assume that such a manager is all-knowing, i.e. that it considers all possible solutions to the bin-packing problem if it must (re-)deploy an application. In reality it will only have limited time to find a subset of the solutions using a heuristic. Both the selection of such a heuristic and the practical implementation are outside of the scope of this work. The results that we find are therefore an upper bound on the fault-tolerance that a real resource manager will be able to achieve.

IV. Fault model

Advances in VLSI design lead to a decrease of the feature size, which increases the power density of chips and causes higher local temperatures. These hot spots prevent powering on all transistors simultaneously at the nominal voltage, an effect known as dark silicon [1]. During operation hot spots may cause intermittent faults that lead to temporary processor shutdowns. In the long term, hot spots cause increased electromigration and speed up the aging process [2]. These effects make multi-processor platforms more susceptible to faults.

In this work we focus on intermittent faults due to hot spots and dark silicon, and permanent faults due to aging and electromigration. As the power density is at its highest in the processors, such faults are most likely to appear there. We therefore focus on processor faults and will not consider the NoC, memory, and other components. Transient faults do not require re-deployment and may instead be addressed with techniques such as checkpoint and restart [11].

To re-deploy a running application in case of a fault, its state must be consistent. Therefore we will now investigate the storage of state in the dataflow MoC. A dataflow application consists of actors that fire as soon as data is available, i.e. they are self-timed and data-driven. An application is captured in a directed graph such as in the left side of Figures 1a-c. The nodes are the actors and the edges are the channels. Actors are stateless, and communicate by producing and consuming data tokens into and from the channels. All state is therefore stored in the channels.

An actor firing consists of three parts: consumption of the input tokens, execution, and production of the output tokens. During firing an actor may temporarily have internal state that is not in the tokens. To ensure that state cannot be lost, we impose the requirement that the input tokens of an actor may only be discarded after the firing is complete and the output tokens are updated. This ensures that an actor can always be re-fired, which guarantees that any fault can be corrected.

The channels must be accessible and in a consistent condition after a fault. To simplify our fault model we use the conservative assumption that the channels as well as the instruction memory are stored in a central, protected memory and are fetched each time when required by a processor. In reality this will have a major impact on the performance, and there are schemes to avoid this and still guarantee consistent memory, e.g. error-correcting codes. Discussion of such schemes is outside of the scope of this work.

Techniques exist for detection of faults on a processor, e.g. through acceptance tests based on timing, coding, reasonability and structure, and may be implemented in hardware, software or both [11], [12], [13]. Upon detection we assume that an exception handler halts all VPs deployed on that processor. Because all input tokens of an actor that was firing are still valid, it can be restarted on another processor as long as the position in the schedule is known. The current position of the schedule can be stored in a (self-) edge.

There are many types of dataflow, such as Kahn process networks, synchronous dataflow, and scenario-aware dataflow. While each has its own specific rules and restrictions, our work is sufficiently general to be applied to all these types.

V. Fault-Tolerance Concept

Heterogeneous multi-processor platforms feature multiple types of processors that differ in clock speed, architecture and instruction set. The contributions that we present in this section are not limited by the number or variety of processor types available on a platform. The minimum number of processor types necessary to explain our contributions is two, see Section I. For the sake of brevity we will continue with such a platform that only consists of general purpose (GP) and accelerators (ACC) processors throughout this work.

We assume, without loss of generality, that all actors in a dataflow graph can be executed on a GP. A subset actors can benefit from execution on an ACC because they have a shorter WCET on that type. All others cannot be executed on an ACC at all.

A. Re-deployment

A fault on a processor affects all applications of which one or more VPs are deployed on that processor. There are different strategies to deal with such faults. Those applications may for example be dropped, switched to a safe mode, or

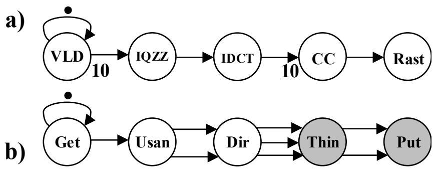

Fig. 3: Two dataflow graphs of a) the JPEG decoder, and b) the Susan edge detection algorithm. Actors *Thin* and *Put* can benefit from an accelerator.

the VPs may be re-deployed on different processors [14]. We argue that re-deployment is the preferred strategy because the functionality is maintained and the interruption of service minimized. Whether sufficient free capacity is available for re-deployment can be checked by a resource manager. If this is not the case, it may still choose another strategy to deal with the fault. From the system perspective every increase of the probability of successful re-deployment is significant, because it reduces the risk of having to drop applications.

Re-deployment of VPs as introduced in Section I is enabled by two principles. Firstly, the MoC provides a method to compute the WCRT on a predictable platform given the actor WCET and a platform model, from which the VP size can be calculated. Secondly, the WCRT accounts for the worst-case TDM wheel misalignment between VPs so that a VP may be deployed on any processor of the correct type (e.g. general purpose) that has sufficient TDM slots available.

Our first contribution is to improve fault-tolerance on a platform by increasing the probability on successful re-deployment. To do this we calculate alternative mappings for each application in which it is mapped to more VPs than in the *baseline* mapping that is explained next.

We consider two baseline mappings for each application. Firstly an application may be mapped to GP processors only, in which case it will not profit from the accelerators. We vary the amount of processors and select the version with the smallest overall size. This is the *homogeneous* baseline mapping. We will see in Section VI that this always results in a mapping to one VP, denoted with $[gp]$.[1] Secondly an application may be mapped to at least one GP and one ACC processor. Again we vary the amount of GPs and ACCs and find that the $[gp, acc]$ mapping (i.e. one GP and one ACC) always results in the smallest overall size, we refer to this mapping as the *heterogeneous* baseline mapping.

B. Resize and Split

Re-deployment of a gp is straightforward, it may only be deployed on a GP processor with sufficient free capacity. For an acc on the other hand there are two options. It may be deployed on an ACC processor with sufficient free capacity, or we may generate a second mapping in which the acc is resized (i.e. enlarged) to allow for a GP processor mapping. This expands the solution space and increases the probability of successful re-deployment when an ACC processor fails.

Consider for example a failure of the ACC processor to which the second VP of mapping $[gp, acc]$ is deployed. At design-time a second mapping $[gp, gp]$ is generated in which the first gp is forced to be identical, and the second VP accommodates the acc actors from the first mapping. During runtime the resource manager can switch to this second mapping if it cannot re-deploy the acc. The disadvantage is that the second mapping must be stored during runtime, and that the resource manager needs the capacity to switch between mappings.

The resizing strategy can potentially also be applied to a homogeneous mapping. As explained in Section II, some TDM slots may be reserved solely to reduce the WCRT. Such slots may be removed from one VP and transferred to another without violating the throughput constraint, i.e. the VPs act like communicating vessels. This could be used to manipulate the location of available slots over the platform. As it does not change the total size of an application however, we expect only a minor benefit and will not explore this strategy further.

Instead we propose as our second contribution to *split* and resize VPs only when the ACC processor to which the acc is deployed encounters a fault at runtime. Consider again the mapping $[gp, acc]$, for which we generate a second mapping $[gp, gp, gp]$ to which we switch when a fault occurs on the ACC. The first gp is again identical, but acc VP is now spread over the two gp VPs. A split can also be applied to a homogeneous mapping.

The advantage is that the additional cost for splitting a VP is only paid when the fault actually occurs, which lowers the cost and platform utilization in normal operation. This comes again at the price of storing the additional mapping and extending the resource manager with the capacity to switch between mappings. Switching between mappings is more complicated in comparison to the resize strategy, as the VP has an internal schedule that must be split in two in a consistent manner. This split operation may be costly depending on the size and type of the schedule, and will cause additional switching overhead at runtime.

VI. Experimental evaluation

A. Preliminary

We use eight different applications for our experiments. Two of these are real-world streaming algorithms, namely a JPEG decoder and the SUSAN edge detection algorithm. Their dataflow graphs are depicted in Figure 3.

We furthermore use six synthetic applications to explore the effect of typical graph structures, see Figure 4. The worst-case execution times of the JPEG and Susan actors are measured on an FPGA instance of the CompSOC platform. Each actor in the synthetic applications has an ET of 10k cycles on a GP. Actors that can benefit from running on an accelerator are indicated in gray in the figures, their ET is 1k cycles on an ACC, representing a speedup of 10x. All other actors cannot be mapped to an ACC at all.

[1]We use the abbreviations GP and ACC to refer to physical processors of type general-purpose and accelerator, and gp and acc for VPs that may be deployed on these respective processor types.

	JPEG	Susan	seq	sota	par	seqpar	parseq	diamond
$[gp]$	$[40]$	$[40]$	$[40]$	$[40]$	$[40]$	$[40]$	$[40]$	$[40]$
$[gp, gp]$	$[33, 10]$	$[30, 10]$	$[35, 5]$	$[37, 40]$	$[10, 30]$	$[36, 5]$	$[22, 20]$	$[20, 20]$
$[gp, gp, gp]$	$[52, 10, 53]$	$[29, 10, 2]$	$[30, 5, 5]$	$[38, 40, 40]$	$[5, 25, 10]$	$[33, 5, 5]$	$[14, 27, 4]$	$[16, 4, 20]$
$[gp, gp, gp, gp]$	$[1, 9, 53, 52]$	$[1, 3, 1, 36]$	$[25, 5, 5, 5]$	$[25, 5, 5, 31]$	$[5, 20, 10, 5]$	$[29, 5, 5, 5]$	$[10, 27, 4, 4]$	$[4, 4, 8, 27]$

TABLE I: Mappings for the homogeneous platform for 8 different applications. The number of VPs varies from $1..4$.

	Susan	seq	par	seqpar	parseq	diamond
$[gp, gp]$	$[30, 10]$	$[35, 5]$	$[10, 30]$	$[36, 5]$	$[22, 20]$	$[20, 20]$
$[gp, \mathbf{gp}, \mathbf{gp}]$	$[-]$	$[-]$	$[-]$	$[-]$	$[-]$	$[-]$
$[\mathbf{gp}, gp, \mathbf{gp}]$	$[29, 10, 2]$	$[5, 5, 30]$	$[5, 30, 5]$	$[5, 5, 33]$	$[-]$	$[-]$
$[\mathbf{gp}, \mathbf{gp}, \mathbf{gp}]$	$[-]$	$[-]$	$[-]$	$[-]$	$[-]$	$[-]$

TABLE II: The split strategy applied to the $[gp, gp]$ mapping. The new VPs are indicated in bold. An empty set indicates that no mapping can be found.

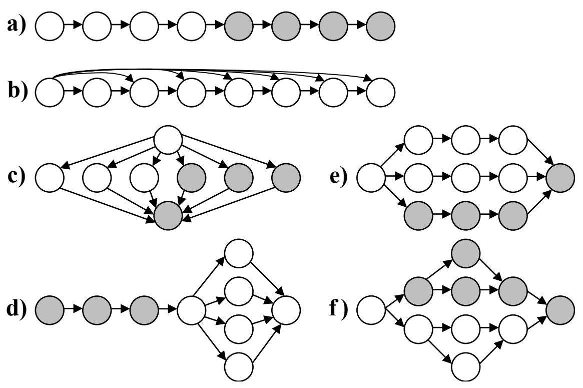

Fig. 4: Six dataflow applications with different topologies: a) sequential (seq), b) sequential one-to-all (sota), c) parallel (par), d) sequential-parallel (seqpar), e) parallel-sequential (parseq), f) diamond. Actors that benefit from an accelerator are indicated in gray.

We employ two different platforms to evaluate our contributions. A homogeneous platform with four GP processors will be used to investigate re-deployment of VPs to processors of the same type. A heterogeneous platform with two GPs and one ACC processor is applied for re-deployment of VPs to processors of a different type. Both platforms have a TDM wheel size of 60 slots on all processors, which allows for easy comparison. On many platforms the wheel size and slot length of GP and ACC processors will differ. It is important to note that the cost expressed in TDM slots cannot be compared between processors if this is the case.

Subsection VI-B describes how we generate different mappings for both platforms. In Subsection VI-C we create sets of applications for each mapping, and generate all possible deployments. In each deployment we simulate a failure of each processor and calculate the average probability of finding a valid re-deployment. The resize and split strategies are evaluated similarly in Subsection VI-D

The platform utilization is obtained by dividing the number of used TDM slots used in a certain deployment by the total number of TDM slots available on the platform. We explore the trade-offs for different utilizations in VI-E.

B. Mapping

We employ the publicly available SDF^3 tool that can generate a number of Pareto-optimized actor-to-VP mappings for each application [15]. In the context of this work we modified the tool to select the mapping that uses the lowest total number of TDM slots. In other words, we ignore memory usage and bandwidth that dominate other Pareto points.

We create four mappings per application for the homogeneous platform by varying the number of VPs to which an application is mapped between one and four, i.e. $[gp], .., [gp, gp, gp, gp]$. The setpoint for the throughput is determined by mapping each application to a single VP with a size of exactly 40 TDM slots. The two, three and four VP mappings are generated with the same throughput setting. We will deploy four applications so 160 out of 240 TDM slots or two-thirds of the platform will be used in the baseline mapping. We consider this a reasonable utilization for a fault-tolerant platform.

Table I shows the mapping of all applications to $1..4$ VPs using SDF^3. The numbers in each cell give the size of the VPs, the total cost of a mapping is their sum. We observe that the total cost increases as the number of VPs increases. The reason for the increase is that the WCRT increases for each additional VP, as explained in Section II. In case the cost does not increase (e.g. Susan between 1 VP and 2 VPs), the number of required cycles increases but is absorbed by the unused part of the TDM slots already allocated.

The results of the JPEG and sota applications in Table I stand out. When either is mapped to 2 (sota only), 3 or 4 VPs the total number of required TDM slots is larger than the TDM wheel size. In case of the JPEG this is caused by the fact that the ET is concentrated in the IQZZ and IDCT actors. For the sota application it is caused by the fact that every actor receives data from the first, therefore the algorithm cannot find a mapping of actors that reduces the WCRT. As they cannot be deployed on one processor when mapped to 3 or 4 VPs, we conclude that our concept does not work for these types of applications and we exclude them from the re-deployment experiments.

Additional mappings must be generated to evaluate the split strategy proposed in Subsection V-B. A fault may affect any

	Susan	seq	par	seqpar	parseq	diamond
$[gp, acc]$	$[25, 1]$	$[23, 3]$	$[22, 4]$	$[24, 2]$	$[24, 2]$	$[23, 3]$
$[gp, \mathbf{gp}]$	$[25, 6]$	$[23, 23]$	$[22, 21]$	$[24, 12]$	$[24, 15]$	$[23, 22]$
$[gp, \mathbf{gp,gp}]$	$[25, 2, 5]$	$[23, 18, 7]$	$[22, 12, 12]$	$[24, 8, 4]$	$[24, 7, 7]$	$[23, 10, 15]$
$[gp, acc, acc]$	$[25, 1, 1]$	$[23, 2, 1]$	$[22, 2, 1]$	$[24, 1, 1]$	$[24, 1, 1]$	$[23, 1, 2]$
$[gp, \mathbf{gp,gp}]$	$[25, 2, 5]$	$[23, 18, 7]$	$[22, 20, 7]$	$[24, 8, 4]$	$[24, 10, 4]$	$[23, 10, 15]$
$[gp, gp, acc]$	$[4, 22, 1]$	$[7, 18, 3]$	$[17, 7, 4]$	$[10, 16, 2]$	$[10, 14, 2]$	$[19, 5, 3]$
$[gp, gp, \mathbf{gp}]$	$[4, 22, 6]$	$[7, 18, 23]$	$[17, 7, 21]$	$[10, 16, 12]$	$[10, 14, 15]$	$[19, 5, 22]$

TABLE III: Mappings for the heterogeneous platform for 6 different applications, including the results of the *resize* and *split* strategies. VPs that are resized or added are indicated in bold.

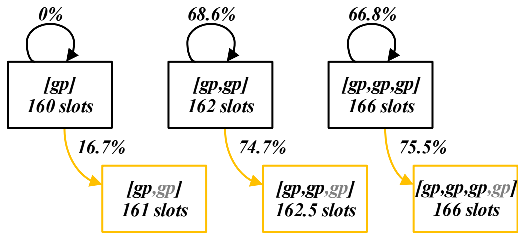

Fig. 5: Re-deployment results for the homogeneous platform. Arrows indicate the probability of successful re-deployment, the boxes contain the number of VPs and cost of the solution. Black boxes and arrows indicate the default strategy, yellow the split strategy.

combination of VPs of one application, so we generate a mapping for each possible permutation of faults. Each faulty VP must be split, while the others keep their size. In the $[gp]$ mapping there is only one VP to be split, resulting in the $[gp, gp]$ mapping in the third row of Table I. Applied to the $[gp, gp]$ mapping for the homogeneous platform, we obtain the result shown in Table II. The new VPs are indicated in bold type. We see that no mappings can be found if a fault is encountered in either the second VP or in both VPs at the same time. This is because of the drastic increase of the WCRT in these two cases, violates the throughput constraint.

We will deploy three applications to the heterogeneous platform using three different mappings, namely $[gp, acc]$, $[gp, acc, acc]$ and $[gp, gp, acc]$, see Table III. The setpoint for the throughput is chosen so that the sum of both VPs in the $[gp, acc]$ baseline mapping is 26 slots. Table III includes the results of the resize and split strategies, VPs that are resized or added are indicated in bold. It is not possible to find a solution for all applications when using the split strategy for mappings $[gp, acc, acc]$ and $[gp, gp, acc]$. As expected, the required number of slots increases steeply when resizing an acc for a GP processor because the actor execution times increase tenfold. We note that splitting the VP comes at a cost of one or two slots, but the maximum size of the two new VPs is always smaller than that achieved with the resize strategy.

C. Re-Deployment

For each of the four mappings on the homogeneous platform we create fifteen unique sets of four applications, which are all permutations of four out of six applications. We generate all possible deployments for each set and consider unique deployments only, i.e. permutations achieved by swapping processors are excluded. The total number of deployments for a 2 VP mapping onto 4 processors is in the order of 10^2, for a 3 VP mapping it is 10^5 and for a 4 VP mapping 10^7.

In each deployment we simulate a failure of each processor and calculate the average probability of finding a valid re-deployment per mapping. These results are summarized in the upper row of black boxes in Figure 5. The percentages on the arrows represent the probability of successful re-deployment averaged over the processors, sets and deployments. The numbers in the boxes denote the average cost of the re-deployment, which is in this case equal to the initial mapping.

The baseline $[gp]$ mapping has a cost of 40 slots. There is exactly one possible deployment, namely one VP per processor, which leaves 20 free slots per processor. The probability that a VP can be re-deployed when a processor fails is then 0%. The $[gp, gp]$ mapping has a probability of successful re-deployment of 68.8%, at an average cost increase of 2 slots or 0.83% of the overall capacity. The $[gp, gp, gp]$ mapping results in a probability on successful re-deployment of 66.8% at a cost of 6 slots. This is lower than for the $[gp, gp]$ mapping, which can be attributed to the fact that the utilization of the platform is higher even before the fault occurs, which reduces the solution space.

The results confirm the validity of our first contribution, i.e. the fault-tolerance increases when an application is mapped to more VPs. The success of the approach will however depend on each unique combination of TDM wheel size, number of applications and processors, throughput constraints, initial VP sizes, and dataflow graph topologies. The possible gains from this strategy should therefore be analysed for each system independently.

D. Resize and Split

We repeat the re-deployment experiment on the homogeneous platform for the split strategy. As shown in Table II it is not possible to split every VP, those VPs are re-deployed without any changes. The results are shown in the lower row of yellow boxes in Figure 5. Note that after re-deployment the platform contains applications mapped to different numbers of VPs, indicated by the gray font in the figure. When the split strategy is applied to the baseline mapping the probability on successful re-deployment is 16.7%, for the $[gp, gp]$ and $[gp, gp, gp]$ mappings the probabilities are 74.7% and 75.5% respectively. We see that the split strategy offers an improvement over the default strategy in all three cases. The cost

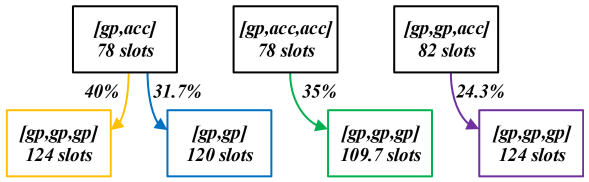

Fig. 6: Re-deployment on the heterogeneous platform. Arrows indicate the probability of successful re-deployment, the boxes contain the VP size and cost in slots. Black boxes indicates the initial deployments, yellow the split strategy and blue, green and purple the resize strategy for the different mappings.

increases marginally in the 1 and 2 VP case, and not at all for the 3 VP case because all VPs can be split without additional cost. We conclude that the split strategy further increases the fault-tolerance on the homogeneous platform.

For each of the three mappings the heterogeneous platform we create twenty sets of three applications, all permutations of three out of six applications. We simulate a failure of the ACC processor only to zoom in on re-deployment between processors of different types. The initial deployments are depicted in the upper row of black boxes in Figure 6. An acc cannot be re-deployed without either resizing or splitting it. The resizing strategy for the $[gp, acc]$ mapping is depicted in the blue box (second from left) in the second row. We see that the probability on successful re-deployment is 31.7%, while the total size increases with 42 slots, or 23.3% of the total platform capacity. The split strategy for the same deployment is shown in the leftmost yellow box, and yields a 40% probability of success at a cost of 46 slots. Again we see that the fault-tolerance increases at the cost of TDM slots, confirming the second contribution. The higher costs are caused by the conversion of acc VPs to gp.

The probability on successful re-deployment of the $[gp, acc, acc]$ mappings with the resize strategy is 35%, which is a minor improvement over the $[gp, acc]$ mapping but still not as good as the split strategy. The resource usage for the $[gp, acc, acc]$ mapping however is significantly lower than for both $[gp, acc]$ solutions. This is because the actors are distributed over 3 VPs from the beginning, providing a better spread of actors than the $[gp, acc]$ split strategy. The $[gp, gp, acc]$ mapping does not perform well compared to the others because the utilization of the GP processors is higher from the beginning, leaving less available capacity for re-deployment.

E. Trade-Offs

The experiments on both the homogeneous and heterogeneous platforms showed a trade-off between fault-tolerance and TDM slots for a fixed platform utilization at the time of the fault. Because in reality the utilization is unpredictable, we vary the utilization on the heterogeneous platform on the x-axis of Figure 7. The graph furthermore shows the solution cost for a selected number of data points. The solution shown in Figure 6 is indicated in the graph by the red box.

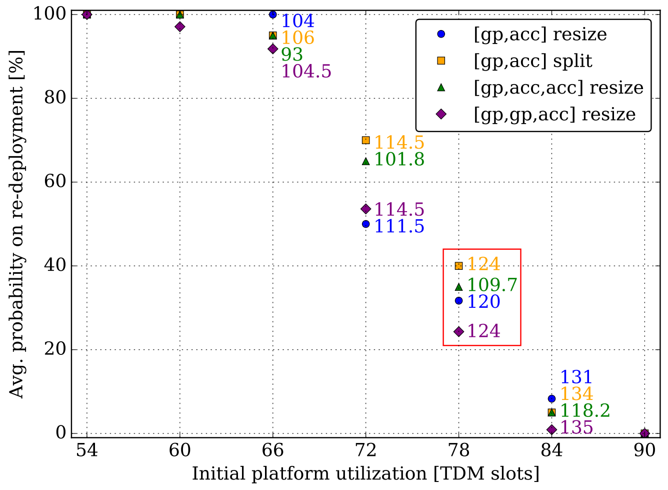

Fig. 7: The probability of re-deployment for a varying initial platform utilization and different strategies. The cost of the solution in TDM slots is printed in color for selected data points.

If the utilization is 54 slots or lower, all strategies result in successful re-deployment because there is always sufficient free capacity to re-deploy each VP. Likewise, if the utilization is 90 or higher, no re-deployment can be found for any of the strategies because there is insufficient free capacity. These lower and higher thresholds will exist on every combination of platform and application set, but their value cannot be predicted.

In between these thresholds each strategy has a curve that trades off fault-tolerance with cost. We see that the $[gp, acc]$ resize strategy provides slightly better fault-tolerance at the edges of this curve, while $[gp, acc]$ split performs well in the middle. The $[gp, acc, acc]$ strategy consistently outperforms the others in terms of the solution cost and always has a (shared) second place in fault-tolerance. This may well make it the strategy of choice in this particular setup, unless the fault-tolerance must be maximized at all cost. In that case the $[gp, acc]$ split strategy provides the highest probability of successful re-deployment, averaged over all utilizations.

VII. Related Work

Mapping strategies fall in three categories: design-time, runtime and hybrid [16]. The hybrid that we use combines elements proposed in [3], [5] and allows any deployment of RT applications at runtime at the cost of rigorous timing analysis at design-time. An alternative method is to calculate and analyse a fixed number of VP deployments at design-time [17], [18]. This allows more precise timing analysis, but the number of deployments grows exponentially with the number of VPs, processors and re-deployment events. These deployments must also be stored during runtime, and unlike our solution no new applications can be accepted.

An online resource manager that combines a greedy heuristic with actor clustering is presented in [19]. Clustering increases the probability of successful deployment because it reduces the required bandwidth, except when processors are almost full. This is in line with our findings. Fault-tolerance however is not considered. Design-time mapping for mixed-criticality and runtime deployment is proposed in [14]. Low criticality tasks can be dropped to guarantee the WCRT of highly critical tasks in case of faults, but not re-deployed.

Kahn Process Networks (KPN) as used in [18], [20] is a dataflow MoC that cannot be statically analysed. Therefore there is no method to compute the WCRT, and our method for fault-tolerance cannot be applied for such applications but can still be of use if occasional deadline misses are acceptable.

Spare processors are reserved to cope with faults in [18], [21]. While this method is safe and easy to verify, it is not affordable in low-cost, high-performance systems, for which our method is targeted. In [17] tasks from faulty processors are re-deployed (migrated) with the goal of minimizing migration cost and the degradation of application throughput at the same time. They also generate mappings at design-time, but do not calculate the probability of successful re-deployment.

Re-deployment strategies focused on performance and the reduction of the communication energy are presented in [20], [21]. The former formulates the problem as a Integer Linear Programming (ILP) problem to generate all deployments for their KPN applications at design-time. They also propose an online heuristic for use at runtime. The latter proposes a spare processor placement technique and assesses its impact on the fault-tolerant properties. They consider the effect of system fragmentation but do not try to remedy this by splitting up applications as we do in this work.

VIII. Conclusion

Heterogeneous multi-processor platforms are suitable for executing a dynamic mix of RT streaming dataflow applications. The decreasing feature size in VLSI design however increases the probability of intermittent and permanent faults. We consider a design flow in which dataflow applications are analysed and mapped to a set of Virtual Processors (VPs) at design-time. At runtime a VP may be deployed on any physical processor of the target type that has sufficient available TDM slots. This gives the flexibility to add new applications during runtime, but makes it impossible to predict what the deployment will be at any given time. When a processor fails, the available slots on the platform may be too fragmented to re-deploy the VPs executing on that processor.

In our first contribution we show that the probability of successful re-deployment increases if applications are mapped to more VPs of a smaller size, at the cost of additional TDM slots. This feature can be exploited for fault-tolerance by starting each application with such a multi-VP mapping, but the extra slots are also used when no fault occurred. This is overcome with our second contribution, which is to split a VP only when a fault occurs. Because of the lower overall resource usage this further increases the probability for successful re-deployment at the cost of storing multiple mappings and splitting the schedule at runtime.

We evaluate the proposed strategies experimentally and show that mapping an application to more VPs indeed increases the probability of successful re-deployment, both for processors of the same type and processors of different types using resizing. This works up to a certain limit, after which the probability decreases because of the increased total cost. Furthermore we find that splitting VPs only when a fault occurs increases the probability even more. When running the experiments for sample deployments with different utilizations, a lower threshold is revealed below which a re-deployment can be found with all strategies, as well a higher threshold above which no re-deployment can be found with any strategy. In between the thresholds we provide insight in the trade-offs between maximizing fault-tolerance and minimizing costs offered by the different strategies. The numbers reported in this work cannot be generalized for other platforms and applications. In low-cost, high-performance systems however, any increase of the fault-tolerance that is achieved without spare processors is an improvement of significance.

Acknowledgments

This work was partially funded by EU grants
ECSEL 692455-2 ENABLE-S3, 737422 SCOTT, 737453 I-MECH.

References

[1] J. Henkel et al., "Towards performance and reliability-efficient computing in the dark silicon era," in *DATE*, 2016.
[2] A. M. Rahmani et al., "Reliability-aware runtime power management for many-core systems in the dark silicon era," *IEEE Transactions on VLSI Systems*, 2017.
[3] S. Stuijk et al., "A predictable multiprocessor design flow for streaming applications with dynamic behaviour," in *DSD*, 2010.
[4] M. Krstic et al., "Globally asynchronous, locally synchronous circuits: Overview and outlook," *IEEE Design Test of Computers*, 2007.
[5] K. Goossens et al., *NOC-Based Multi-Processor Architecture for Mixed Time-Criticality Applications*, S. Ha and J. Teich, Eds. Springer, 2017.
[6] A. Lele et al., "A new data flow analysis model for TDM," in *EMSOFT*, 2012.
[7] H. A. Ara et al., "Tight temporal bounds for dataflow applications mapped onto shared resources," in *SIES*, 2016.
[8] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, 2003.
[9] G. Scheithauer, *Introduction to Cutting and Packing Optimization*, C. C. Price, Ed. Springer, 2017.
[10] S. Sinha et al., "Composable and predictable dynamic loading for time-critical partitioned systems," in *DSD*, 2014.
[11] E. Dubrova, *Fault Tolerant Design*. Springer, 2013.
[12] G. A. Reis et al., "Design and evaluation of hybrid fault-detection systems," in *ISCA*, 2005.
[13] S. K. S. Hari et al., "mSWAT: Low-cost hardware fault detection and diagnosis for multicore systems," in *MICRO*, 2009.
[14] S.-h. Kang et al., "Static mapping of mixed-critical applications for fault-tolerant MPSoCs," in *DAC*, 2014.
[15] S. Stuijk et al., "SDF^3: SDF For Free," in *ACSD*, 2006.
[16] A. K. Singh et al., "Mapping on multi/many-core systems: Survey of current and emerging trends," in *DAC*, 2013.
[17] C. Lee et al., "A task remapping technique for reliable multi-core embedded systems," in *CODES+ISSS*, 2010.
[18] L. Schor et al., "Scenario-based design flow for mapping streaming applications onto on-chip many-core systems," in *CASES*, 2012.
[19] O. Moreira et al., "Online resource management in a multiprocessor with a network-on-chip," in *SAC*, 2007.
[20] O. Derin et al., "Online task remapping strategies for fault-tolerant network-on-chip multiprocessors," in *NOCS*, 2011.
[21] C. L. Chou and R. Marculescu, "FARM: Fault-aware resource management in NoC-based multiprocessor platforms," in *DATE*, 2011.