Efficient Reconfiguration of Processing Modules on FPGAs for Space Instruments

Sándor Fekete[1], Björn Fiethe[2], Stephan Friedrichs[1], Harald Michalik[2], and Christos Orlis[1]

[1]Institute of Operating Systems and Computer Networks (IBR), Dept. of Computer Science, TU Braunschweig
Mühlenpfordstr. 23, 38106 Braunschweig, Germany
s.fekete@tu-bs.de, {sfriedr,orlis}@ibr.cs.tu-bs.de

[2]Institute of Computer and Network Engineering (IDA), Dept. of Electrical Engineering, TU Braunschweig
Hans-Sommer-Str. 66, 38106 Braunschweig, Germany
{b.fiethe,michalik}@tu-bs.de

***Abstract*—We consider optimization techniques for a problem that requires a valid scheduling and allocation of tasks on Field Programmable Gate Arrays (FPGAs). A concrete application on a scientific space instrument arises in the context of ESA's Solar Orbiter mission; making use of dynamic reconfiguration allows a good and flexible use of resources, but the resulting packing and scheduling problems in the presence of inhomogeneous allocation resources are quite challenging. In our scenario, modules are described by three parameters: their *resource demands* for different types of resources, their *priority*, and their *clock frequency*. These are to be allocated on an FPGA that provides a number of different resources that are available at specific locations. We first present an Integer Program that partitions the tasks in such a way that all constraints can be met and the reconfiguration overhead is minimized, and then give methods for allocating the processing modules of the partitioned tasks on the FPGA. We evaluate our methods on a real application of the Solar Orbiter PHI instrument. The results obtained indicate computational efficiency and a remarkable solution quality.**

I. Introduction

In recent years, the demand for on-board processing capabilities of space missions has been increasing steadily. In the future, spacecrafts will have to handle very high data rates, and final physical values will have to be extracted and processed by an autonomous, intelligent and reliable application on-board the spacecraft, adapting itself to the changing needs. Classical ground-processing steps need to be performed on-board, making this a design driver for future robotic missions and planetary landers. The benefits of an adaptable processing platform will be a superior data yield and a reduced risk of a total instrument loss [15]. An additional advantage of adaptability is the possibility to time-share resources, when dedicated functions are not necessary at the same time. All this promises a more efficient hardware and power utilization.

To cope with these sophisticated on-board processing capabilities, state-of-the-art radiation-tolerant, space-qualified SRAM-based Field Programmable Gate Arrays (FPGAs) provide large gate count and high clock speeds [22]. Additionally, they offer an attractive solution to speed up processing by utilizing their parallel structures. On the other side, reconfigurable SRAM-based FPGAs are very susceptible to radiation effects. Therefore, dedicated mitigation techniques against SEEs (Single Event Effects) have to be implemented. We have already demonstrated the successful usage of SRAM-based FPGA devices for scientific instruments with the first European SRAM FPGA-based computer in space for the Venus Monitoring Camera (VMC) on ESA's Venus Express mission launched in 2005 [14], as well as the Dawn Framing Camera on NASA's Dawn mission launched 2007 [25].

Optimizing partition and allocation of tasks and modules to the FPGAs poses a number of computational challenges. Even simple one-dimensional variants of the involved packing and scheduling problems are NP-hard, which is typically taken as a reason to resort to mere heuristics [16]. In addition, the problems faced for this task are even more challenging: Because the layout on the FPGA involved is two-dimensional, considering allocation over time yields three-dimensional problems [7], [8]; furthermore, the placement involves several different resources, adding yet another layer of difficulty. Despite of this, we demonstrate that real-world instances of relevant size can still be solved to optimality, based on state-of-the-art optimization techniques, and provide methods and results for efficient use of the strictly limited resources.

In Section II we introduce a scientific space instrument with high demands for on-board processing. The specific requirements of the problem and parameters are described in Section III. Section IV gives an overview of the algorithms, while Section V provides some results for the specific space instrument. Section VI concludes with a final summary.

II. Space Instrument

The Polarimetric and Helioseismic Imager (PHI) instrument was selected as part of the scientific payload for ESA's Solar Orbiter mission. The instrument will provide maps of the

978-1-4799-5356-1/14/$31.00

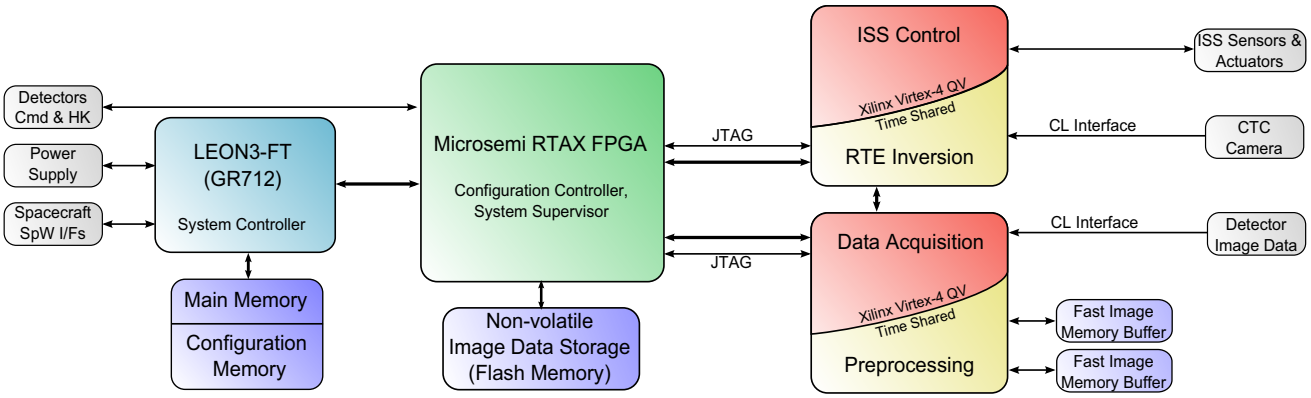


Fig. 1: PHI data processing module block diagram with reconfigurable FPGAs.

continuum intensity, the magnetic field vector and the Line-Of-Sight velocity in the solar photosphere [27]. The huge amount of scientific raw data on the one side, and the limited telemetry rate on the other side require a sophisticated on-board scientific data processing and data evaluation. Moreover, because of long mission duration, it is mandatory that the processing is adaptable to changing mission and scientific valuable requirements. A non-linear, least-square, iterative process implemented in a reconfigurable FPGA is used for this Radiative Transfer Equation (RTE) inversion, which determines the full polarization state of the incoming solar light [23].

The complete high-level architecture of this processing module including two reconfigurable FPGAs is shown in Figure 1, for more details see [13]. For the PHI data processing, two modes can be defined:

- Image acquisition mode;
- data processing mode.

In acquisition mode, one of the Xilinx Virtex-4 SX55 FPGAs is used for control of the instruments own Image Stabilization System (ISS), while the second FPGA accumulates the incoming image data stream. The raw images are stored in the non-volatile flash memory. In the following processing mode, the images are read out from the non-volatile memory; one FPGA performs pre-processing, as well as final data compression, while the complete second FPGA performs the RTE inversion.

Before the RTE inversion can be carried out, a complete chain of data pre-processing has to be performed, e.g. dark-field (DF) and flat-field (FF) correction, binning (B), fringe correction (FC), deconvolution from the Point Spread Function (PSF), polarization demodulation (PD), cross-talk (CC) and polarimetric correction (PC) and a few more [17], see Table I. Finally, a wavelet-based data compression (WDC) has to be performed on the dataset before transmission to the spacecraft. Of course SEU effects have to be considered in a dynamic reconfigurable system. But because the data processing is completely decoupled by intermediate storage, no special mitigation is needed for this configuration. To detect any failure, these parts have to be tested regularly with predefined test vectors of values with known answers. In case of any failure, the configuration has to be corrected by scrubbing and the data have to be re-processed.

In addition to the standard processing of data, calibration tasks have to be performed from time to time. This is not time critical and could be done completely in software running on the LEON processor; however, dedicated processing modules within a minimized number of FPGA configurations would help to exploit all available HW resources, and thus lower the necessary power.

III. Problem Description

The ability of SRAM-based FPGAs to support dynamic reconfiguration allows a very flexible use of the available HW platform in a Time-Space Partitioning (TSP) manner, even for complex algorithms. For data-processing applications like the one introduced in the previous section, different pre-processing algorithms have to be mapped onto a single reconfigurable FPGA under various constraints. Assuming the complete processing is performed offline with a huge number of datasets acquired and stored in non-volatile memory, and each algorithm needing only a small portion of FPGA space, several datasets can be processed in parallel by the same FPGA. The number of FPGA logic resources is limited, so not all of the processing modules may be mapped onto one FPGA at the same time, but can be split into several complete configurations. A dedicated processing chain is not required, but the number of necessary reconfigurations needs to be minimized in order to get the best performance for complete processing of a set of data. Partial reconfiguration at runtime adds much higher complexity to the overall system and is not implemented in the PHI processing module. Efficiency in FPGA resource management has become a key issue, especially in time-critical applications. To achieve this, all needed processing modules for all processing steps have to be combined already at design time to the best solution that requires a minimum number of FPGA configurations. This is particularly important for space missions, because memory capacity to store FPGA configurations and telecommand bandwidth to upload new ones is very limited.

Basically, the single processing modules cannot be freely placed on the reconfigurable device, but the overall distribution has to account for the given static constraints of required heterogeneous resources and speed prority of each module, e.g., those modules needing highest throughput should be placed first and as densely as possible. The granularity of configuration has to take into account *configuration frames* as smallest addressable units to be configured independently. This will allow future use also for partially reconfigurable systems. Such a basic configuration does not preempt the detailed routing and optimization of the FPGA's vendor-specific tools, as these are required to be performed in an additional stage.

These challenges make it highly desirable to employ state-of-the-art models and methods from mathematical optimization to compute feasible module placements meeting all constraints. As we demonstrate, the relevant instance size can still be solved to optimality, making use of the right tools.

IV. Models and Methods

A. Models

Most FPGAs consist of an array of $m \times n$ blocks providing various resources. For Xilinx-4/5 FPGAs, there are the following resource types: Logic slices (SLC), block RAM (BRAM), dedicated multiplication/addition blocks (DSP), input/output blocks (IOB), and timer blocks (CLK) [28], [29], compare Figure 2. We refer to the resource types by the set $R = \{\text{SLC}, \text{BRAM}, \text{DSP}, \text{IOB}, \text{CLK}\}$; I is the set of blocks on the FPGA. Furthermore, for a block $i \in I$, we denote by $R(i) \in R$ the type of resource provided by block i.

We are presented a set M of modules that has to be placed on the FPGA. Each module occupies some amount of resources. Let $S : R \times M \to \mathbb{N}$ be a function, such that $S(r, m)$ denotes the amount of resources of type $r \in R$ required by the module $m \in M$. The overall amount of resources of type $r \in R$ available on the FPGA is depicted by $S(r)$.

Futhermore, the modules are grouped into tasks. The idea is that modules belonging to the same task colaborate and thus must be present on the FPGA simultaneously. Let the set T denote all tasks. For the sake of simplicity, we define the resource requirements of a task as

$$S : R \times T \to \mathbb{N}, \quad S(r, t) \mapsto \sum_{m \in t} S(r, m) \tag{1}$$

In addition to the resource demands, every module has two more parameters: its priority and its clock frequency. High-priority modules are needed by every single task and may not be moved in between configurations. High-frequency (i. e., high-throughput) modules have a limitation on the maximum allowed distance of the allocation. All blocks occupied by such a module must be able to communicate within one clock cycle, which translates into a limit of distance between those blocks as worst-case boundary to ensure high-frequency communication after final vendor specific place and route. This requirement is more relaxed for low-frequency modules. Because the communication time between blocks corresponds to the number of axis-parallel hops between them (for an axis-aligned orientation of the FPGA), the frequency requirements impose upper bounds on the Manhattan (L_1) distance between any two blocks of a processing module $m \in M$. We call this number the *maximum diameter* $d(m)$ of m. For allocating a high-frequency module m with a clock frequency of e.g. 100 MHz on a member of the Virtex-4 device family, the maximum diameter $d(m)$ of m is:

$$d(m) = \left\lfloor 7 + \frac{\max_{r \in R} S(r, m)}{4} \right\rfloor, \tag{2}$$

whereas, for a low-frequency module m, a considerably larger maximum diameter suffices:

$$d(m) = 4 \cdot \left\lfloor 7 + \frac{\max_{r \in R} S(r, m)}{4} \right\rfloor. \tag{3}$$

Thus, the maximum feasible shapes correspond to unit balls in the Manhattan metric, i. e., diamonds. For an example, consider Figure 2 for a schematic image of the involved FPGA, and the required resources of a module. For a high-frequency module, we get a maximum feasible diameter of $7 + \left\lfloor \frac{24}{4} \right\rfloor = 13$.

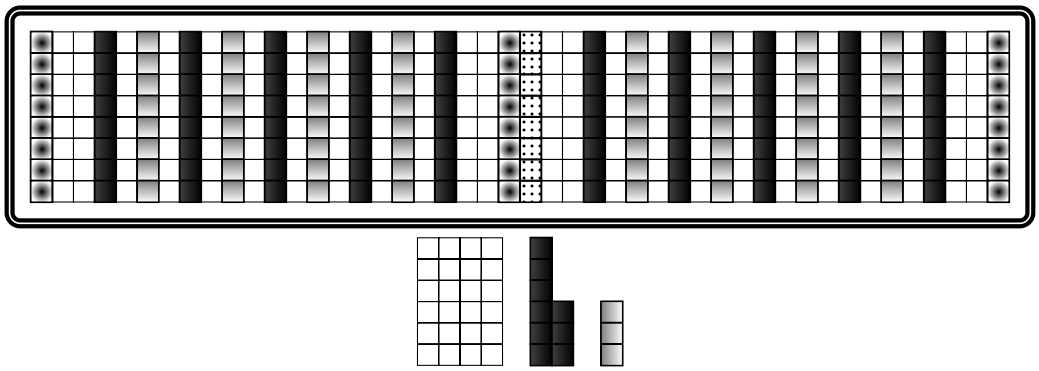
As described above, the FPGA may be too small to accomodate all modules of all tasks while satisfying all resource and diameter constraints. Thus, we have to find a partition of the involved tasks and their modules into a number of different configurations, such that every task is contained in one configuration; because high-priority modules are always needed, they must be part of *every* configuration; in order to restrict reconfiguration effort, it is desirable to keep them at the same positions in each configuration. In order to optimize resource usage and to minimize the overall reconfiguration overhead, we aim for a minimum number of configurations. With these considerations, the objective of our module allocation problem (MAP) is to find a partition of the tasks into a minimum number of configurations, such that:

- All high-priority modules are part of every configuration.
- Each module is allocated to a number of blocks that corresponds to its resource demands.
- All diameter constraints are maintained.
- No block in a configuration is used for more than one module.
- A high-priority module is allocated to the same set of blocks in all configurations to allow implementation of partial reconfiguration.
- The total number of configurations is minimized.

B. Methods

As mentioned in the introduction, our task allocation problem is the generalization of several classical problems that are known to be NP-hard [16]. In particular, even the simplification of MAP to a one-dimensional FPGA with a single resource (i. e., block type) corresponds to the NP-complete problem Partition; trying to minimize the number of configurations in this setting is precisely the classical Bin Packing Problem. It is also known that two-dimensional generalizations

Fig. 2: Schematic description of the array T for the Xilinx Virtex-4 SX55 FPGA and of a module (type 14, "Linear fitting" in Table I) that requires three different types of resources: 24 SLC blocks, 9 BRAM blocks and 3 DSP blocks. In this FPGA, there are 5 different types of resources. SLC-type resources are described by white tiles, BRAMs are described by black tiles, DSPs are described by gray tiles, IOBs are described by the tiles with the one big black spot and CLKs are described by the dotted ones.



of the Bin Packing Problem with a single resource and without diameter constraints are even harder from a practical point of view [10]. For slightly more general problems in the context of reconfigurable computing (which are still simpler than MAP), consider [2], [7], [12], [19]–[21], [26], and [1] for a relatively recent survey of optimization methods in reconfigurable computing.

Therefore, it is clear that MAP is an extremely difficult problem in terms of computational complexity. In the following, we decompose the problem into two that are still NP-hard, but simpler from a practical point of view. Despite the tremendous complexity of the individual combinatorial optimization subproblems, each of them can be solved with sophisticated IP solvers. What is more, we demonstrate that this is still possible to solve the *overall* optimization problem to optimality, so it is not necessary to resort to heuristic methods that produce suboptimal solutions.

1) An IP for minimizing the number of configurations: As a first step, we partition the set T of tasks into subsets for which the available FPGA resources suffice to allocate them into a single configuration. We model this problem as a Multi-Dimensional Bin Packing Problem; note that each of the "dimensions" represents a different type of resource in each configuration, i.e., *not* a geometric dimension. While this problem is still NP-hard, it allows relatively benign formulations as Integer Linear Programs (IPs) that can be efficiently attacked by state-of-the-art IP solvers. See [5] and its references for a thorough treatment. Fast lower bounds for bin packing and multidimensional bin packing problems can be found in [9], while [24] is a recent study on heuristics that can also be used as solution approaches and lower bounds.

For the IP formulation, we use B as a known upper bound on the number of bins required; our software uses the Next Fit algorithm [18] to find one. Note that if there are high-priority modules that require resources on the FPGA, this can simply be realized by reduced FPGA capacities $S(r)$.

$$\min \sum_{b=1}^{B} y_b \tag{4}$$

$$\text{s.t.} \sum_{b=1}^{B} x_{tb} = 1 \qquad \forall t \in T \tag{5}$$

$$\sum_{t \in T} S(r,t) x_{tb} \leq S(r) y_b \qquad \forall 1 \leq b \leq B, \forall r \in R \tag{6}$$

$$y_b \in \{0,1\} \qquad \forall 1 \leq b \leq B, \tag{7}$$

$$x_{tb} \in \{0,1\} \qquad \forall 1 \leq b \leq B, \forall t \in T \tag{8}$$

In this formulation, (7) are the *bin variables* y_b that are 1 if and only if the b^{th} bin (i. e., configuration) is used. *Assignment variables* x_{tb} in (8) describe whether task t is placed on the b^{th} bin. (5) ensures that every task t is assigned to one and only one bin (configuration), while (6) guarantees that (i) the variables of all selected bins are set to 1 and (ii) that the resources available in each bin suffice for the assigned tasks. Finally, (4) minimizes the number of configurations. This IP ensures all task requirements are met while using a minimum number of bins, so its solution vector y^{IP} describes an optimal partitioning of the tasks in a minimum number of configurations.

Note that up to this point, we only consider the partitioning of the tasks into configurations, while ignoring the actual placement of the tasks' modules on the FPGA.

2) Efficient allocation of the modules: In a second stage, we allocate the modules of all tasks that have been assigned to the same configuration to specific positions on a single FPGA. Given a set of modules along with their parameters, especially resource demands and maximum diameters, we have to allocate them in a way that satisfies all constraints. This preliminary floorplanning provides a configuration-aware placement of modules on the FPGA, which does not preempt detailed place and route of the FPGA's vendor-specific tools.

Thus, specific communication architectures or interfaces are not explicitly taken into account.

To this end, we present a second-stage IP for the allocation of a set of tasks' modules on the FPGA.

$$\min \quad 0 \tag{9}$$

$$\text{s.t.} \sum_{m \in M} x_{im} \leq 1 \qquad \forall i \in I \tag{10}$$

$$\sum_{\{i \in I | R(i) = r\}} x_{im} = S(m) \qquad \forall m \in M, \forall r \in R \tag{11}$$

$$x_{im} + x_{jm} \leq 1 \qquad \forall m \in M, \forall i < j \in I, \; L_1(i,j) > d(m) \tag{12}$$

$$x_{im} \in \{0,1\} \qquad \forall m \in M, \forall i \in I \tag{13}$$

In this formulation, (13) are variables that are 1 if and only if the module m is placed in block i. Every block i may be occupied by at most one module, which is ensured by Constraint (10), Constraint (11) guarantees that the resource demands are met for module m, and Constraint (12) prevents m from occupying two blocks that are further apart than its maximum diameter. The objective function (9) of this IP is zero, because we are just interested in feasible solutions.

We subdivide this stage into two substages:

(2a) Allocate the high-priority modules on an empty FPGA. We keep their positions fixed in the following steps. Let J denote the set of blocks occupied by high-priority modules.

(2b) After fixing the blocks for high-priority modules: Allocate the low-priority modules in each configuration. This is achieved by replacing I with $I \setminus J$ in the IP (9)–(13).

Using methods for improving the solution quality (as discussed in Subsection IV-B5) for the first substage has a very beneficial effect on reducing the compactness of the overall solutions. (In particular, minimizing the high-priority diameters already suffices for producing compact allocations.)

3) Solutions to the two-stage approach: Pursuing a two-stage approach splits the overall problem into several subproblems; while maintaining fixed positions for all high-priority modules, we try to identify feasible allocations for each subproblem. Each of the stages and subproblems still are NP-hard, but smaller in size, and thus considerably easier to solve to optimality. We note that this may be sufficient to find optimal solutions in relatively short time, as discussed in Section V. The correctness is based on the following theorem.

Theorem 1: Consider a solution of the IP described by (4)–(8) that yields feasible solutions for each of the subproblems described by (9)–(13). Then y^{IP} is a minimum number of configurations for the overall problem, so we have found an optimal solution.

Proof: Any optimal feasible solution must satisfy the constraints (5)–(8), so solving the IP (4)–(8) computes a valid lower bound. Satisfying the subproblems (9)–(13) establishes a feasible solution with this value, so we have found an optimal solution. ■

As it turns out, this suffices to solve the relevant instance for the PHI processing module to optimality in a few seconds of computation time.

4) Dealing with infeasibilities at the second stage: Even though the second stage did produce feasible solutions for all relevant problem instances in our practical setting, it is clear from a theoretical point of view that this may not be the case for even more complicated instances. There are a number of different approaches that can be pursued if that happens.

- We can relax the diameter constraints (12) and consider minimizing the maximum weighted diameter of any module over all configurations. (Here the weight factor maps the distinction between high-frequency and low-frequency modules, as described in (2) and (3).)
- We can enforce the diameter constraints and replace the first stage by a branch-and-bound framework for partitioning the modules into different configurations.
- Allow (some) high-priority tasks to be moved.

The preference depends on whether it is more desirable to achieve a small number of configurations (at the expense of lowering module frequencies) or maintain high frequencies (at the expense of increasing the number of configurations). We show in Subsection IV-C that there is an alternative one-stage approach that promises to be superior to either alternative, though resulting in large IPs and thus computation times.

5) Improving the solution quality: Any solution produced by the second stage yields a way to minimize the number of configurations while maintaining all diameter constraints. However, there may be a large number of such feasible solutions, some of which are more fragmented than others. This can be dealt with by adding an objective function to the second-stage IPs that minimizes the total diameter of all modules within an allocation, which aims at minimizing the *average* communication cost inside of all modules, in addition to maintaining diameter constraints. (See [4] for a more detailed discussion of the involved problems.)

$$\min \sum_{m \in M} \sum_{i<j \in I} y_{ijm} \tag{14}$$

$$\text{s.t.} x_{im} + x_{jm} \leq 1 + y_{ijm} \qquad \forall m \in M, \forall i < j \in I \tag{15}$$

$$\sum_{m \in M} x_{im} \leq 1 \qquad \forall i \in I \tag{16}$$

$$\sum_{\{i \in I | R(i) = r\}} x_{im} = S(m) \qquad \forall m \in M, \forall r \in R \tag{17}$$

$$x_{im} + x_{jm} \leq 1 \qquad \forall m \in M, \forall i < j \in I, \; L_1(i,j) > d(m) \tag{18}$$

$$x_{im} \in \{0,1\} \qquad \forall m \in M, \forall i \in I \tag{19}$$

$$y_{ijm} \in \{0,1\} \qquad \forall m \in M, \forall i < j \in I \tag{20}$$

In this formulation, the auxiliary binary variables, see (20) and (15), y_{ijm} are 1 if the pairs of blocks i and j are both occupied by the same module m. The objective function (14) accounts for all intramodule distances, other than that, this IP is equivalent to the one in (9)–(13). As a result, we get

almond-like module shapes, roughly corresponding to optimal shapes as discussed in [3], [6]. Note that we get additional modifications, due to the inhomogeneous resource distribution on the FPGA.

This objective function is just one of the possible ways to improve solution quality. It is easy to adjust the formulation to minimizing the maximum diameter, just consider the diameters of high-frequency modules, or aim at weighted linear combinations of the involved parameters. In principle, it is also possible to perform defragmentation in an online fashion; see [11] for further discussion.

C. A One-Stage Approach

The first stage of the two-stage approach finds a split of tasks into a smallest possible number of different subsets, such that the resource constraints for each subset are satisfied. The second stage then searches for feasible allocations. We can roll both stages into one, by considering I to be the set of all blocks in all configurations in the second IP. In that case, the distance between two blocks i and j in different configurations needs to be larger than the maximum diameter of any module; conversely, allocation of different modules for the same task can be handled by limiting their maximum distance. This can also be used for the refined approach with a defragmenting objective function of type (14).

V. Application Example

In this section we demonstrate the practical usefulness of our approach by showing an optimal solution for the PHI data processing application (see Table I for a detailed description of the application and Figure 3 for an allocation of some tasks). The experiments were run on an Intel Xeon 3.20 GHz (Quadcore) with 3 GB of memory, running 64 bit Ubuntu 13.10 with Linux 3.11. Our algorithms were not parallelized and the version of CPLEX that we used is 12.5.0.0. In order to test the robustness of our approch to additional demands, we expanded this benchmark by adding further modules, one at a time, and running the whole algorithm from scratch.

As it turns out, we managed to solve even instances with 28 more modules of different sizes (a load increase by more than a factor of 2.5 compared to the original PHI data processing application) without any problems regarding the resource demands and the diameters of the modules; see Figure 4. We also performed tests for a calibration application. The experimental setting here is the same as before; the solution quality is also comparable.

VI. Conclusions

Architectures based on reconfigurable FPGAs will satisfy the demand of future space missions for high-performance onboard processing with the requirement to update processing modules in-flight. This is a favorable solution for the sophisticated data processing requirements within the very tight power and thermal constraints of scientific space missions.

Accounting for all involved constraints and objectives brings together a large number of computationally hard optimization problems, so the resulting tasks are quite challenging. As we have demonstrated, making use of state-of-the-art modeling and solution techniques still allows obtaining provably optimal solutions.

Our IP supports heterogeneous configuration resources at the lowest device configuration granularity. Most remarkably, our case study shows that it is possible to compute optimal solutions for relevant instances within a few seconds by using appropriate tools for modeling an solving mathematical optimization tasks. Thus, there is no need to resort to mere heuristics, which may end up requiring more computation time for inferior solutions.

It is clear that modifications of constraints and objectives will lead to modified solutions. For example, we may aim for rectangular module layout, which can be modeled by making use of the L_∞ instead of the L_1 (Manhattan) metric; the resulting layout are bounded in an axis-parallel fashion, making them easier to pack than the diamond shapes resulting from the Manhattan metric. The resulting problems are easier in nature, so our solution approach still works. The same applies if we aim for coarser granularity; e.g., reserving FPGA columns for only one module at a time turns the two-dimensional problem into a one-dimensional one.

In summary, today's practical challenges from space missions can be handled by combining modern reconfigurable hardware with cutting-edge optimization techniques. This requires interdisciplinary collaborations in order to make use of current knowledge from several fields. We are optimistic that this will continue to bring about important and relevant progress.

Acknowledgements

This work is part of the DFG Research Unit "Controlling Concurrent Change", funding number FOR 1800. In particular, funding for the Department of Computer Science was provided under grant number FE 407/17-1, while funding for the Department of Electrical Engineering was provided under grant number MI 1172/4-1.

References

[1] A. Ahmadinia, J. Angermeier, S. P. Fekete, D. Göhringer, T. Kamphans, D. Koch, M. Majer, N. Schweer, J. Teich, C. Tessars, and J. C. van der Veen. Reconodes – optimization methods for module scheduling and placement on reconfigurable hardware devices. In *Dynamically Reconfigurable Systems: Architectures, Design Methods and Applications*, pp. 199–221. Springer Netherlands, 2010.

[2] J. Angermeier, S. P. Fekete, T. Kamphans, N. Schweer, and J. Teich. Virtual area management: Multitasking on dynamically reconfigurable devices. In *Proceedings of the 17th International Reconfigurable Architectures Workshop*, 2010.

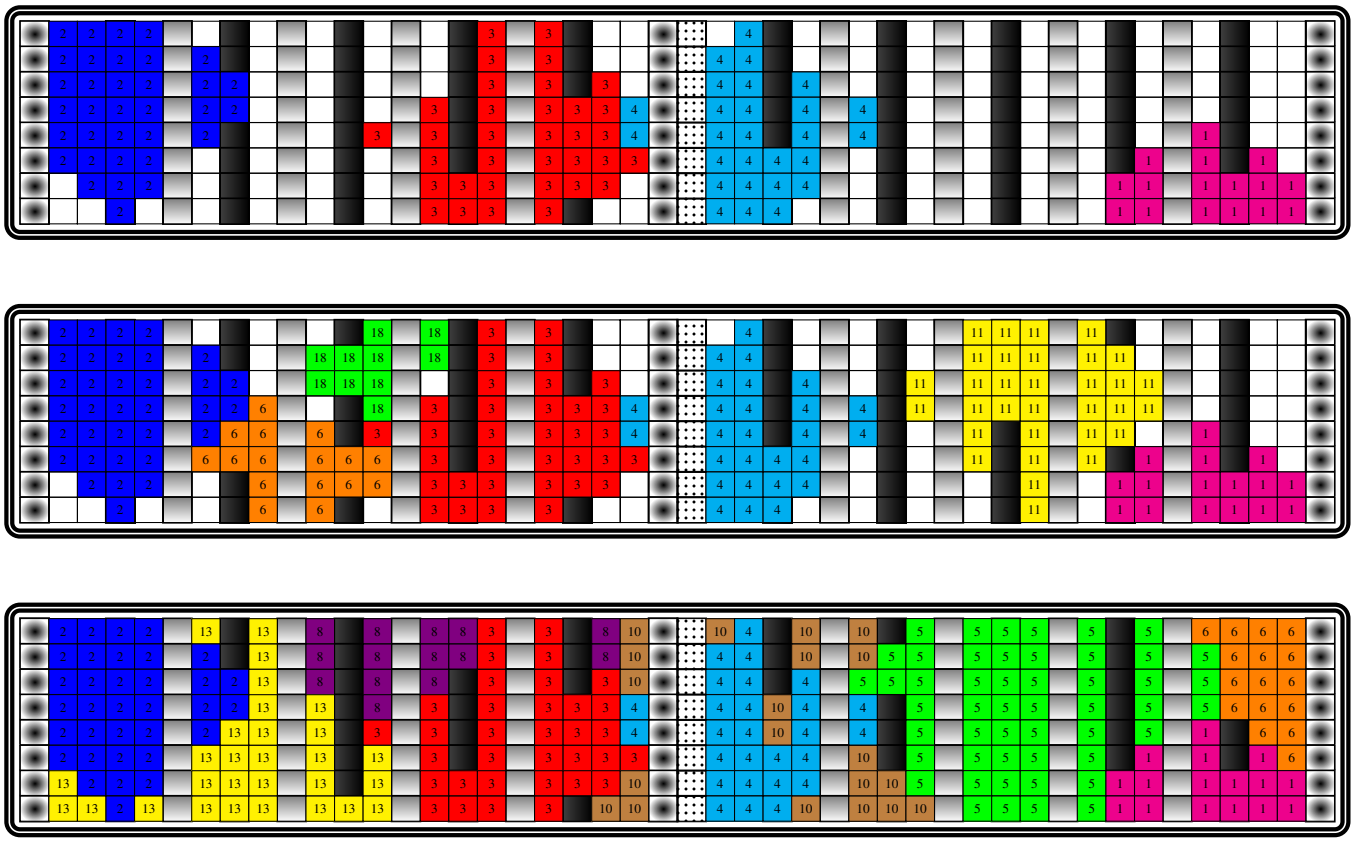
[3] C. Bender, M. Bender, E. Demaine, and S. P. Fekete. What is the optimal shape of a city? *J. of Physics A: Mathematical and General*, 37(1), 2004.

[4] M. A. Bender, D. P. Bunde, E. D. Demaine, S. P. Fekete, V. J. Leung, H. Meijer, and C. A. Phillips. Communication-aware processor allocation for supercomputers: Finding point sets of small average distance. *Algorithmica*, 50(2):279–298, 2008.

Module	Task(s)	Priority	Clock frequency	Configuration block allocation			Diameter
				#SLCs	#BRAMs	#DSPs	
1. Memory access	All	High	High	12	4	-	10
2. SDRAM controller	All	High	High	24	10	-	13
3. Network communication switch	All	High	Low	28	6	-	56
4. Processing control	All	High	Low	24	3	-	52
5. Wavelet-based data compression	WDC	Low	High	40	10	-	17
6. Addition/Subtraction of frames	DF,FF,CC,PC	Low	Low	12	4	-	40
7. Division of frames	FF	Low	Low	16	4	-	44
8. Scalar multiplication	PD,PC	Low	Low	12	2	-	40
9. Scalar division	PD	Low	Low	12	2	-	40
10. Comparator	SD,MD	Low	Low	16	4	-	44
11. Binning processing	B	Low	Low	24	8	-	52
12. FFT processing	FC (2×),PSF (2×)	Low	Low	24	7	3	52
13. Filtering processing	H/D	Low	Low	24	5	-	52
14. Linear fitting processing	CC	Low	Low	24	9	3	52
15. Mean values processing	CC	Low	Low	16	4	-	44
16. Matrix operations processing	PD	Low	Low	40	5	4	68
17. Image shifting	IA	Low	Low	16	8	2	44
18. Integer to FP conversion processing	CRTE	Low	Low	8	2	-	36
FPGA resource availability (Xilinx Virtex 4)				**192**	**80**	**64**	

TABLE I: Description of the modules along with a task mapping of our PHI data processing application. Here, Task FC requires for 2 FFT processing modules while Task PD for 3 different modules.

Fig. 3: Example of three allocations for the PHI data processing application by using our IP. The first-stage IP produced a total of five configurations. The top figure shows the allocation of the high-priority modules (step **(2a)**, which was carried out with additional diameter minimization. These are completed to full allocations (step **(2b)**, two of which are shown in the middle and at the bottom. Note that the bottom configuration uses all SLC resources; also note that most of the low-priority modules in this configuration are also low-frequency modules, so their diameter constraints are looser.



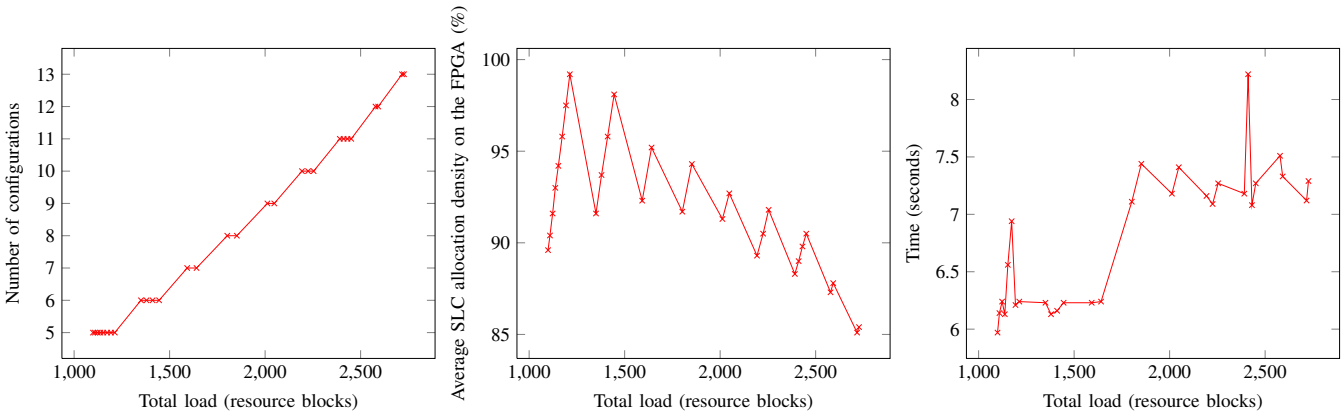


Fig. 4: Computational results for the PHI data processing application (14 tasks including 2 HH modules, 2 HL modules, 1 LH module and 21 LL modules), and a demonstration of scalability of our approach. The first figure shows the number of configurations needed in each experiment while the second one the average SLC allocation density in each experiment. Finally, the third one shows the running times in each experiment. The first point of each curve corresponds to the original PHI data processing application, which can be solved to optimality within a few seconds; the experiments show that even expanding the instance size by a factor of 2.5 does not have a serious affect on the computation times.

[5] C. Chekuri and S. Khanna. On multi-dimensional packing problems. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 185–194. SIAM, 1999.

[6] E. D. Demaine, S. P. Fekete, G. Rote, N. Schweer, D. Schymura, and M. Zelke. Integer point sets minimizing average pairwise L1 distance: What is the optimal shape of a town? *Computational Geometry: Theory and Applications*, 44(2):82–94, 2011.

[7] S. Fekete, E. Köhler, and J. Teich. Optimal FPGA module placement with temporal precedence constraints. In *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, pp. 658–665. IEEE, 2001.

[8] S. P. Fekete, E. Köhler, and J. Teich. Higher-dimensional packing with order constraints. *SIAM J. Discrete Math.*, 20(4):1056–1078, 2006.

[9] S. P. Fekete and J. Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical Programming*, 91(1):11–31, 2001.

[10] S. P. Fekete, J. Schepers, and J. C. van der Veen. An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, 55(3):569–587, 2007.

[11] S. P. Fekete, J. C. van der Veen, A. Ahmadinia, D. Göhringer, F. Hurtado, and J. Teich. Offline and online aspects of defragmenting the module layout of a partially reconfigurable device. *IEEE Trans. VLSI Systems*, 16(9):1210–1219, 2008.

[12] S. P. Fekete, J. C. van der Veen, J. Angermeier, D. Göhringer, M. Majer, and J. Teich. Scheduling and communication-aware mapping of HW/SW modules for dynamically and partially reconfigurable SoC architectures. In *Proc. 20th Internat. Conf. Archit. Comput. Syst.*, pp. 151–160. VDE-Verlag, Berlin, 2007.

[13] B. Fiethe, F. Bubenhagen, T. Lange, H. Michalik, H. Michel, J. Woch, and J. Hirzberger. Adaptive hardware by dynamic reconfiguration for the Solar Orbiter PHI instrument. In *Proc. 2012 NASA/ESA Conf. on Adaptive Hardware and Systems (AHS)*, pp. 31 –37, 2012.

[14] B. Fiethe, H. Michalik, C. Dierker, B. Osterloh, and G. Zhou. Reconfigurable system-on-chip data processing units for space imaging instruments. In *Proceedings of the conference on Design, automation and test in Europe*, pp. 977–982. EDA Consortium, 2007.

[15] L. Fossati and J. Ilstad. The future of embedded systems at ESA: Towards adaptability and reconfigurability. In *Proc. 2011 NASA/ESA Conf. on Adaptive Hardware and Systems (AHS)*, pp. 113-120, 2011.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[17] J. Hirzberger. Phi instrument user manual. Manual SOL-PHI-MPS-OP3000-MA-1, Max Planck Institute for Solar System Research, December 2013.

[18] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[19] M. Koester, W. Luk, J. Hagemeyer, M. Porrmann, and U. Ruckert. Design optimizations for tiled partially reconfigurable systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(6):1048–1061, 2011.

[20] M. Koester, M. Porrmann, and H. Kalte. Task placement for heterogeneous reconfigurable architectures. In *Proc. IEEE International Conference on Field-Programmable Technology.*, pp. 43–50, Dec 2005.

[21] N. Marques, H. Rabah, S. Jovanovic, E. Dabellani, and S. Weber. Methodology and reconfigurable architecture for effective placement of variable-size hardware tasks. In *Adaptive Hardware and Systems (AHS), 2013 NASA/ESA Conf. on*, pp. 156–163, June 2013.

[22] H. Michalik, T. Fichna, B. Fiethe, F. Bubenhagen, H. Michel, and B. Osterloh. Adaptive Computers in Space - In-flight HW-reconfigurable processors using high density FPGAs. In H. Michalik, T. Fichna, B. Fiethe, F. Bubenhagen, H. Michel, and B. Osterloh, editors, *Proceedings of the 8th Symposium on Small Satellites for Earth Observation*, Germany, April 2011. Deutsches Zentrum für Luft- und Raumfahrt e.V., International Academy of Astronautics.

[23] D. Orozco Surez and J. C. del Toro Iniesta. The usefulness of analytic response functions. *Astronomy&Astrophysics*, 462(3):1137–1145, February 2007.

[24] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder. Heuristics for vector bin packing. *research. microsoft. com*, 2011.

[25] H. Sierks, H. Keller, R. Jaumann, H. Michalik, T. Behnke, F. Bubenhagen, I. Büttner, U. Carsenty, U. Christensen, R. Enge, B. Fiethe, P. Gutiérrez Marqués, H. Hartwig, H. Krüger, W. Kühne, T. Maue, S. Mottola, A. Nathues, K.-U. Reiche, M. Richards, T. Roatsch, S. Schröder, I. Szemerey, and M. Tschentscher. The Dawn Framing Camera. *Space Science Reviews*, 163:263–327, 2011.

[26] J. Teich, S. P. Fekete, and J. Schepers. Optimization of dynamic hardware reconfigurations. *J. Supercomput.*, 19(1):57–75, 2001.

[27] J. Woch. Polarimetric and Helioseismic Imager for Solar Orbiter Experiment Interface Document - Part B. Technical Note SOL-PHI-MPS-SY1000-IF-1, Issue 2, Rev. 4, Max Planck Institute for Solar System Research, August 2011.

[28] Xilinx. Virtex-4 FPGA configuration user guide. Technical Report UG071 (v1.11), Xilinx, June 2009.

[29] Xilinx. Virtex-5 FPGA configuration user guide. Technical Report UG191 (v3.11), Xilinx, October 2012.