Virtual Area Management: Multitasking on Dynamically Partially Reconfigurable Devices

Josef Angermeier[#1], Sándor P. Fekete[*], Tom Kamphans[*2], Nils Schweer[*], Jürgen Teich[#]

[#] *Department of Computer Science 12, University of Erlangen-Nuremberg*
Erlangen, Germany
{angermeier, teich}@cs.fau.de

[*] *Department of Computer Science, Braunschweig University of Technology*
Braunschweig, Germany
{s.fekete, n.schweer}@tu-bs.de, tom@kamphans.de

***Abstract*—Every year the computing resources available on dynamically partially reconfigurable devices increase enormously. In the near future, we expect many applications to run on a single reconfigurable device. In this paper, we present a concept for multitasking on dynamically partially reconfigurable systems called *virtual area management*. We explain its advantages, show its challenges, and discuss possible solutions. Furthermore, we investigate one problem in more detail: Packing modules with time-varying resource requests. This problem from the reconfigurable computing field results in a completely new optimization problem not tackled before. ILP-based and heuristic approaches are compared in an experimental study and the drawbacks and benefits discussed.**

I. Introduction

Reconfigurable devices offer more and more space and functionality over time and will probably continue to do so in the future. Yet, huge hardware applications can already be instantiated on the reconfigurable chips, or even arrays of processors. Furthermore, nowadays reconfigurable chips are mostly used to instantiate a single application with multiple modules, which might not all be necessary at each instant in time. However, as reconfigurable devices evolve further, the resources offered will also be large enough to run completely different applications simultaneously. This development also took place in the software world: In the beginning of the nineties, most personal computers used operating systems such as MS-DOS, which allowed just one single software application to be executed, along with some drivers. Since then, increased computing resources have allowed us to run multiple applications simultaneously.

Similar to the software world, multitasking will lead reconfigurable devices to higher efficiency, but will also raise several challenges that must be solved. For example, in order to provide reliability and security, we must make sure that no application can violate the execution of another one. In software this is solved by running each application in its own virtual address space. Each application knows only the virtual addresses of its own resources and parts of the operating system. The virtual addresses are translated at runtime into physical addresses in the memory. Thus, pages of different applications may be physically adjacent, but only the approved application may access them. Furthermore, the concept of the virtual address space facilitated writing software, as each application did not need to bother about the positions of other applications but can act as if it is the only user of the processor and the memory.

The basic idea of our concept which we called "Virtual Area Management" is to partition the available FPGA area into different regions. Each hardware application obtains one contiguous region, which may grow or shrink depending on the applications resource needs. Each running hardware application can reconfigure hardware modules in its region and free or try to allocate more area. In this process, each application knows only the relative positions of its modules, not the the physical positions. Thus, it cannot reconfigure any region that belongs to a different application and is concerned only with its own resources. The most efficient inter-module communication is obtained between adjacent modules. Thus, the modules that must communicate with each other are preferably placed near to each other. However, we can also handle the communication problem between partial modules and the I/O-periphery, see Section II-C.

Due to lack of space, this is a quite terse presentation. For thorough descriptions we refer the reader to our technical report [1].

A. Related Work

Communication between the partial modules and with the input and output periphery is an important point. We assume that the application modules or the hardware platform provide some means for inter-module and I/O-port communication. There are several possible ways to achieve this goal (see, e.g., [2], [3], [4]).

1) Reconfigurable Computing: Brebner [5], [6] addressed the problems involved in presenting a larger virtual hardware resource that is implemented using smaller physical FPGA hardware to a software-oriented user. Their approach is based on using swappable units, and a prototype operating system is described that demonstrates operational steps. In contrast

[1]Supported by DFG grant TE 163/14-2, 14-3, project "ReCoNodes", as part of the Priority Programme 1148. "Reconfigurable Computing".

[2]Supported by DFG grant FE 407/8-2, 8-3, project "ReCoNodes", as part of the Priority Programme 1148, "Reconfigurable Computing".

978-1-4244-6534-7/10/$26.00 ©2010 IEEE

to that work, this paper does not address the problem of overcoming the physical constraints given by a small FPGA, but tackles the problem, of how to run multiple applications on a single reconfigurable resource.

Operating systems for reconfigurable embedded platforms have been developed, e.g. in [7], [8]. The focus in former works was put more on hardware abstraction, we focus on securely running multiple, dynamic hardware applications.

Many recent works have focused on achieving optimal performance to put multiple tasks onto one FPGA within this context (e.g. in [9], [10]). The different applications are represented by a task graph and the aim is to obtain a total near-optimal execution time by taking reconfiguration and communication times into account. However, this approach does not allow dynamic behavior, such that according to the current state of the resources, the tasks can select on their own which module implementation to reconfigure next and at which nearby position to place it.

2) Packing: It turns out that placing hardware modules with growing and shrinking area resources amounts to strip packing. There is a lot of work on classical strip packing, in the online- as well as in the offline case (see, e.g., [11], [12], [13], [14], [15], [16]). Note that none of the previous works allows stretching the objects in any direction.

II. Virtual Area Management

A. Main Idea

To run different applications on a partially reconfigurable device, we propose to partition the available reconfigurable area into so called *virtual areas* (VAs). For performance reasons, the different VAs should consist of contiguous reconfigurable area units. As the required amount of resources of an application changes over time, the size of the virtual area may change dynamically over the running time of its application. The mapping of virtual area to physical reconfigurable area is done by the control processor. Each hardware application being executed on the reconfigurable device has its own software control thread running on this CPU, see Fig. 1. These threads request the initially required area and transmit changes in the requirements. An operating system service on the software side takes the requests and is in charge of the management of the virtual areas. This secure operating system unit maps the virtual area units to the corresponding physical dynamically reconfigurable area units. The virtual area management unit can be compared to the memory management unit (MMU) in the software world: both handle the translation of virtual to physical memory positions. Furthermore, the corresponding application software threads do not know the actual physical positions of the reconfigured modules, only the relative positions of each reconfigurable module to each other. Each application simply requests to load a specified module to a virtual position in the assigned virtual area. See the example in Fig. 1: The second application with its virtual area VA2 issues a request to load a specified bit-file called "X" to the virtual address (here called: *VSlot*) '2'. It does not know that this virtual address corresponds physically

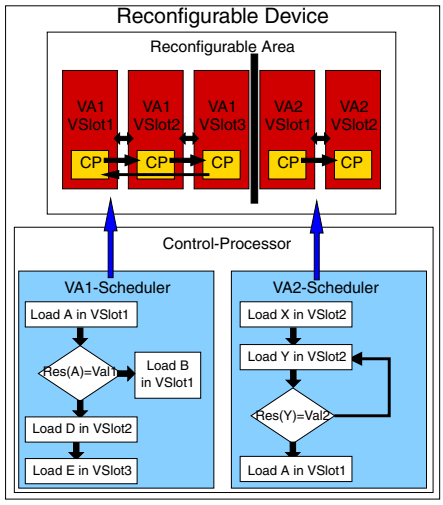

Fig. 1. Two different hardware applications running on a 1D partially dynamically reconfigurable device: The first three slots form the first virtual area (VA1), the remaining the second virtual area (VA2). Modules in one VA can communicate with each other, e.g. by using their communication point (CP) belonging to reconfigurable bus. Neither the first nor the second application knows the exact position of its VA on the reconfigurable device; only the size of the available area and the relative positions of the reconfigurable modules loaded in its virtual area are known.

to the last reconfigurable unit on the reconfigurable device. It knows only that the module loaded there is on the right side of a module loaded to the virtual address (VSlot) '1'. As each application is allowed to specify only a virtual address corresponding to its virtual area in which to place the module, it cannot place a module into an area belonging to a different application.

There are many other approaches to schedule tasks of different applications. In contrast to them, our approach of virtual areas combines a localization strategy, putting tasks which belong to one application together, and highly dynamic applications, which can individually make different module selection and placement decisions based on the current context.

B. Advantages

The idea of virtual area management offers the following advantages when running multiple applications on a single dynamically partially reconfigurable device.

- *Resource accounting*: The concept can be easily used to prevent applications from using too many resources at runtime. One might want to restrict the resources granted to an application for each execution or just in a certain context in order. The goal is, for example, to reduce the power consumption or to accelerate more important applications in a system where certain applications have lower priorities than others. The concept of virtual area management provides a limitation mechanism by granting just a certain resource amount to an application.
- *Resource protection*: Each application controls only the reconfiguration within its virtual area, as it can only specify those virtual positions that are valid in its own VA. The virtual area management unit checks for each

request if the virtual address is valid and translates it into a physical position on the reconfigurable device. In this way, no application can load any module to the area of another application and the applications are protected. Thus, an error in one application cannot harm the execution of all other applications and lead to a complete system breakdown.

- *Support for differing scheduling and placement procedures*: Different applications require different scheduling and placement methods to increase performance. Instead of designing complex scheduling algorithms that try to meet all the requirements (e.g., periodic or aperiodic, with or without deadlines) by introducing various priority classes, each application gets its own virtual area and specifies the best scheduling strategy. This may include a simpler and faster implementation of new hardware applications within an existing system.
- *Area position transparency and programming dynamic applications*: The independence of the executed modules to the absolute positions, in the following called *area position transparency*, offers a new programming model. It allows one to write dynamic applications, which, subject to the current resource context, decide on their own how to proceed. Depending on the assigned area, an application can either use an implementation that offers more performance but needs more area, or another one that takes longer but uses less area. Another new possibility is that the application can decide how to increase and shrink depending on the current area usage context. An application can ask the virtual area management unit, if it can grow to the left or right, or at the bottom, and, depending where some unused area is available, it can decide to put a partial module there and instantiate some appropriate communication module to transfer data there and back. Thus, at each run, an application can operate differently in its amount of resource usage. Before, trade-off decisions were also possible for a single application, but the new idea is to let each application decide in its own control program its next steps depending on the behavior of the other applications.

C. Challenges

A technical challenge lies in the communication of the partial modules to the external periphery (e.g., video, audio) over the input/output pins at the border of the FPGA. Our experimental board has a crossbar device that routes I/O signals dynamically from the periphery to the current position of the partial module [2], [3], [4].

A larger challenge is to fulfill the changing resource requirements of the different applications. Every application has a different resource usage profile that depends on the inputs specified. An application called with a larger problem instance to solve needs also more resources. Additionally, the resource usage profile also depends on the execution context. The application may behave differently depending on how many area resources are available at each position. Furthermore, the resource requirement depending on the inputs of an application may or may not be known in advance (or at least can be estimated, e.g., in a numerical application based on the required precision of the solutions). The former is called the offline case, the latter the online case.

In the online case, a deadlock is possible when two applications currently running on the reconfigurable device can only proceed in their task when an resource increase is granted, see Fig. 2. The blocks represent the occupied area of one application at different points in time. The leftmost application needs two more area units, as does the application second from the left. Furthermore, no application can give up its accumulated resources, as saving and restoring states is widely considered too expensive on FPGAs. Such a deadlock must be prevented. A commonly used solution is to allow hardware task preemption. Another approach is not to wait until a deadlock scenario actually happens, but to check beforehand that it cannot get this far: Assuming that the maximal size of a request is known, an area shared between two applications is granted exclusively to just one application, but not one part of it to the one application, and another part to the other application.

In the following, we consider the offline problem: The resource usage profiles for specific inputs of a series of applications is known a priori, or can be estimated roughly. An example of application resource profiles is given in Fig. 2. Note that in the offline case deadlocks cannot occur: We search for feasible solutions (i.e., schedules where no resource requests overlap) only.

Hardware task preemption can be used to increase the area usage. However, saving and restoring the states of the hardware applications can be very costly in time and memory. An approach that balances reconfiguration costs and efficient resource usage is to allow requests to be delayed by the scheduler. The application keeps its currently occupied area, but remains idle until the request is fulfilled. Compare the two schedules for our example shown in Fig. 2: The schedule shown on the left hand side is a solution for scheduling the application modules without delaying requests. On the right hand side, we allow that requests are postponed. Using this option for the forth request of the forth module, we achieve a better makespan.

III. Packing Application Modules

We consider the problem *FPGATris*: Scheduling modules whose resource requests (i.e., space on an FPGA) varies over time. We assume that a module occupies a certain number of slots on the FPGA, but requests only complete slots. Thus, we model the FPGA as a one-dimensional array. Furthermore, we assume that time is discrete; that is, requests are multiples of a fixed-size time slot. Now, scheduling a sequence of modules with time-varying resources corresponds to strip packing: The width of the strip is the number of slots on the FPGA; the height corresponds to the time axis.

We are allowed to *delay* a request; that is, we may stretch the modules along the time axis; see Fig. 2. Our goal is to

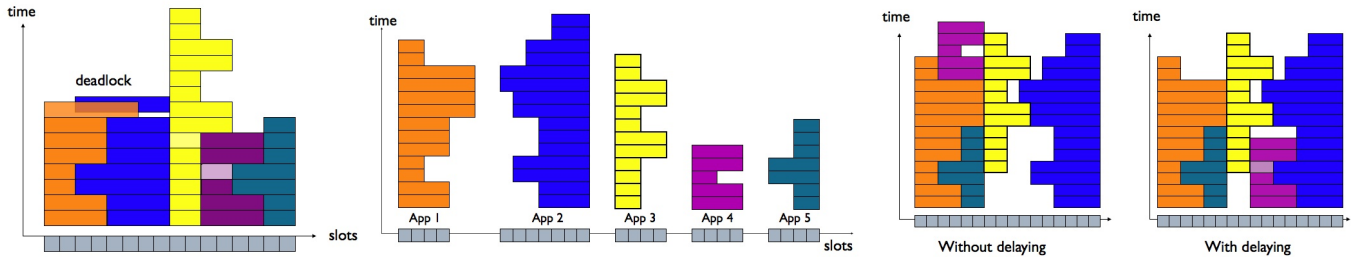

Fig. 2. Left: Both the first and the second application (from the left), request more area resources in order to proceed. A deadlock occurs, because no more resources are available to meet any request. Middle: Five applications modules, each with different resource demands over time for their specific inputs. Right: Schedule of the resource demands on the reconfigurable device under the assumption that running applications cannot wait for a resource grant compared to a schedule with the option to delay resource grants.

minimize the *makespan* (i.e., the time needed to fulfill all requests). Equivalently, we want to minimize the height of the occupied part of the strip.

This problem is optimally solvable with an Integer Linear Program. However, solving instances of this problem takes quite some time, so we also implemented several heuristics: A simple FirstFit with and without delaying requests, and two more elaborated heuristics, BestFit and TabuSearch. For a thorough description of the heuristics as well as the ILP we refer the reader to our technical report [1].

To test our heuristics, we conducted a set of experiments. Indeed, the tabu search yields the best results (in our setting 20% better than Best Fit), but has a higher running time[1].

Choosing the best suited strategy depends on the scenario. For systems that run the same request sequence on and on and that are produced in a large number, it may even pay off to use the ILP. Clearly, balancing computation time and quality, the tabu search is a better choice. Nevertheless, it requires that the requests are known beforehand. If this is not given, BestFit can be used, because it works in an offline scenario as well as in an online setting.

IV. Conclusion

Reconfigurable devices increasingly offer enough computing resources, so that in the near future, multiple applications may be executed on them concurrently, instead of just a single application. However, until now there is a general lack of research on how to successfully achieve secure and flexible execution of multiple applications on dynamically partially reconfigurable devices. We present an approach called *virtual area management*, which is heavily influenced by multitasking and operating system concepts of the software. Advantages of our concept (e.g., support for accounting of resources, resource protection, multiple scheduling and placement strategies, and a new programming model) are explained. Furthermore, challenges posed by this concept and possible solutions are discussed. Afterwards, a specific approach to virtual area management in the offline case is presented in more detail. It is based on the assumption that most applications can handle also a delayed resource grant. The corresponding optimization problem to minimize the total makespan is solved with an ILP and heuristics. Both approaches are compared in an experimental study.

[1] Average running times in our experiment: FF 0.25 s, FF with delays 0.33 s, BF 2.09 s, Tabu Search 1125 06 s.

References

[1] J. Angermeier, S. P. Fekete, T. Kamphans, N. Schweer, and J. Teich, "Maintaining virtual areas on FPGAs using strip packing with delays," Computing Research Repository, Technical Report ArXiv:1001.4493, 2010.

[2] D. Koch, C. Haubelt, and J. Teich, "Efficient reconfigurable on-chip buses for FPGAs," in *Proc. 16th Annu. IEEE Sympos. Field-Programm. Custom Comput. Mach.*, 2008.

[3] J. Angermeier, D. Göhringer, M. Majer, J. Teich, S. P. Fekete, and J. V. der Veen, "The Erlangen Slot Machine — A platform for interdisciplinary research in dynamically reconfigurable computing," *Information Technology*, vol. 49, pp. 143–148, 2007.

[4] C. Bobda, M. Majer, A. Ahmadinia, T. Haller, A. Linarth, J. Teich, S. Fekete, and J. van der Veen, "The Erlangen Slot Machine: A Highly Flexible FPGA-Based Reconfigurable Platform," *IEEE Symp. on FPGAs and Custom Computing Machines (FCCM)*, pp. 319–320, 2005.

[5] G. Brebner, "A virtual hardware operating system for the xilinx XC6200," in *Field-Programmable Logic Smart Applications, New Paradigms and Compilers*, 1996, pp. 327–336.

[6] ——, "The swappable logic unit: a paradigm for virtual hardware," in *Proc. 5th Annu. IEEE Sympos. FPGAs Custom Comput. Machines*, 1997, pp. 77–86.

[7] G. Wigley and D. Kearney, "The development of an operating system for reconfigurable computing," in *Proc. 9th Annu. IEEE Sympos. Field-Programm. Custom Comput. Machines*, 2001, pp. 249–250.

[8] H. Walder and M. Platzner, "Reconfigurable hardware operating systems: From design concepts to realizations," *Proc. 3rd Internat. Conf. Engineering Reconf. Syst. Architect.*, pp. 284–287, 2003.

[9] R. Cordone, F. Redaelli, M. A. Redaelli, M. D. Santambrogio, and D. Sciuto, "Partitioning and scheduling of task graphs on partially dynamically reconfigurable FPGAs," *IEEE Transact. Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, pp. 662–675, 2009.

[10] F. Redaelli, M. D. Santambrogio, and D. Sciuto, "Task scheduling with configuration prefetching and anti-fragmentation techniques on dynamically reconfigurable systems," in *Proc. 11th Conf. on Design, Automation and Test in Europe*, 2008, pp. 519–522.

[11] B. S. Baker, E. G. C. Jr., and R. L. Rivest, "Orthogonal packings in two dimensions," *SIAM J. Comput.*, vol. 9, pp. 846–855, 1980.

[12] B. S. Baker and J. S. Schwarz, "Shelf algorithms for two-dimensional packing problems," *SIAM J. Comput.*, vol. 12, pp. 508–525, 1983.

[13] J. Csirik and G. J. Woeginger, "Shelf algorithms for on-line strip packing," *Inform. Process. Lett.*, vol. 63, pp. 171–175, 1997.

[14] Y. Azar and L. Epstein, "On two dimensional packing," *J. Algorithms*, vol. 25, pp. 290–310, 1997.

[15] E. Coffman, Jr., P. J. Downey, and P. Winkler, "Packing rectangles in a strip," *Acta Inform.*, vol. 38, pp. 673–693, 2002.

[16] S. P. Fekete, T. Kamphans, and N. Schweer, "Online square packing," in *Proc. 20th Algorithms and Data Structure Symposium (WADS)*, 2009, pp. 302–314.