UTILIZING RECONFIGURABLE HARDWARE TO OPTIMIZE WORKFLOWS IN NETWORKED NODES

Deploying partial reconfiguration to optimize load and dependability in constrained networks

Dominik Murr, Felix Mühlbauer, Christophe Bobda
Self-Organizing Embedded Systems Group, Department of Computer Science
Kaiserslautern University of Technology
Dominik@Murr.com, {Muehlbauer,Bobda}@informatik.uni-kl.de

Falko Dressler
Computer Networks and Communication Systems, Department of Computer Science
Friedrich-Alexander University Erlangen-Nuremberg
Dressler@informatik.uni-erlangen.de

Abstract This work investigates the use of reconfigurable devices as computing platform for self-organizing embedded systems. Those usually consist of a set of distributed, autonomous nodes interacting with each other in order to solve a given problem. Several aspects of hardware-software co-design as well as partial reconfiguration are presented in order to enforce adaptivity of a node. One targeted application field for this kind of system are sensor networks in which reconfigurable devices, in this case FPGAs, can be used as computation nodes to provide services that require more computation power. To manage the available hardware resources as a whole we suggest a market-economy-like system of supply and demand. Requests, built up out of several tasks, can be posed to the collective.

The goal is to gain a system able to perform simple tasks as well as very complex computations, while keeping the overall energy consumption low. This will be achieved by deploying highly specialized hardware accelerators and a reasonable resource management. First results show the viability of the methods in the distributed management of available resources.

Keywords: FPGA, Partial Reconfiguration, Distributed Computing, Self-Organization

Introduction

The ongoing progress in chip technology provides us with steadily increasing computation power whereas the accompanying miniaturization shrinks the device sizes to unprecedented measures. Meanwhile electronic circuits are produced at rock-bottom prices. This trend has been around for so long that some are sure to have found a law in it. The new thing is that small, efficient, yet fairly powerful devices, are built that are able to communicate. This puts developers in the position of creating large and complex systems of interacting nodes that are small, powerfull and energy efficient. With the computation power arises the ability of deploying highlevel-algorithms to implement some kind of intelligence into the networked units. This can be used to realize mechanisms of labor division and collective cooperation. Clearly this approach works if every specialized node behaves as specified by the designer at compile time. However, the system has no way to react to deviations due to failure-prone devices or changing operational and environemental conditions.

In order to enforce flexibility in such systems and allow single nodes to adapt their behavior to disturbances, we use reconfigurable units, in this case FPGAs. Those devices pose a decent trade off between computation power, energy consumption and flexibility. They can be configured and reconfigured to provide hardware acceleration for highly specialized services. Furthermore, they can be partially reconfigured while keeping the rest of the system working. Some are even capable of initiating their reconfiguration themselves keeping the part of their logical circuits that hosts the local operating system up and running while only a small partition that hosts specialized accelerators is being reconfigured. Despite those advantages a powerful and flexible management of available resources is required in order to optimize parameters like the overall power consumption in the system or a reasonable distribution of work among the nodes. In this work we present our developed approach for tackling this problem. A framework is built up that enables nodes to (re)distribute tasks in a marketplace-like manner that we called **LMGS** - *Local Marketplaces, Global Symbiosis* - which delivers the basis technology to elevate implementations of collective task completion to a new level. Here a node that recognizes the need for a certain task to be done formulates it as a query to itself and its neighbors. Every node that offers the execution of a task replies to a query with its cost for fulfilling the job. The inquirer is now able to choose whether it is more appropriate to maybe reconfigure and do the task by itself or to delegate.

An application field for this novel approach of a community of self-configuring nodes are sensor networks. Nowadays they are a prime example for a set of connected nodes that are bound to several restrictions: nodes are only powerful to a certain extend, energy is limited since simple sensors are battery driven and communication is costly. Still the data volumes to be processed in those networks are escalating not only when thinking of optical surveillance with recognition of biometrical features. We will show how to enhance the capabilities of such a network by deploying self-reconfigurable nodes as participants.

Next we point out some of the work that yields the foundation for our concept (part 1). Following we will go into the technological basis we've constructed in order to allow self-reconfigurable nodes (2), explain the ideas behind **LMGS** (3) and demonstrate how these two essentials can be combined to drive powerful sensor networks (4). Finally we summarize this article (5).

1. Related Work

On the hardware side we want to concentrate on the application field of sensor networks. Here a variety of solutions has been constructed like the Mica2Dot Mote [6]. They particularly cause a low power consumption but in our case these systems are too weak concerning computation power. Their clock rate is as low as 10MHz compared to the racy Virtex-4 FPGAs that run at up to 500 MHz. Also their processors cannot be deployed as flexible as reconfigurable hardware since they are static and hard wired. Here FPGAs represent a much better compromise for our needs.

As a limitlessly flexible management system for resources we use Linux as operating system for the nodes. We built our own kernel and distribution on the basis of the PowerPC port of the Linux kernel and MontaVista Linux.The partial reconfiguration is described in the main features in the Xilinx documents [9, 11].The marketplaces idea took advantage of previous work like auction methods presented by Gerkey et. al [1], but we wanted to keep the algorithm much more simple to be able to deploy it on weak nodes with less computation power as well. The Open Agent Architecture [2] might be a viable approach for a very stable network without too much fluctuation and a central server that is very unlikely to stop working. We think the application field is much wider if we don't constrain our system to using only one specialized node. We moreover favor decentralized management and easy replacement of failing units. The aim is a system small enough to run on the weakest units but also extremely expandable to allow sophisticated decision making when executed on powerful nodes. To estimate the energy consumption of transporting a certain amount of data over a connection energy aware routing protocols will be useful [4, 5]. The energy demand when operating a hardware module can be determined with [3, 8].

2. Basic Technology Available

The essential resources that must be available on each node in a distributed cooperative system are a minimal local computation power, a resource management, the ability to communicate and a system to distribute or gather jobs in the network. The own computation power allows nodes to locally solve tasks or to run more or less simple programs to decide whether jobs should be solved remotely. To be aware of the own capabilities and to easily deploy them there's necessity for a resource management that is flexible and generally applicable to run on most nodes in the network. From rudimentally equipped to all-in-one high-potential units. Communication allows cooperation to happen in order to spread parts of a bigger assignment that possibly cannot be solved by only one node to several others. Finally our framework **LMGS** provides a simple way to exploit the available resources, locally and remote, the most efficient way and to distribute large tasks among the contributors. It therefore realizes a variant of self-organization.

2.1 Computation Power in FPGAs

To develop and test our concepts we use a Virtex-II Pro FPGA by Xilinx sitting on the XUP development board as well as the Xilinx ML403 board equipped with a Virtex-4 FX12 FPGA [10]. The FGPAs provide one resp. two embedded PowerPC hard-core RISC processors that are contained in the chip fabric. Besides the configurable logic cells, that allow custom hardware accelerators, the Virtex-II Pro and Virtex-4 both supply a certain amount of basic, hard-wired circuits (primitives) to extend the device's speed and effectiveness. These are e.g. multipliers, block RAM and especially a module named ICAP: Internal Configuration Access Port. As depicted in figure 1 this module is the key ability for a node to change its own reconfigurable logic. Additionally Xilinx FPGAs since the Virtex-II series are capable of partial reconfiguration. That means that single hardware modules can be exchanged while vital parts of the cell like the memory controller or the network interface controller keep on operating uninterruptedly.

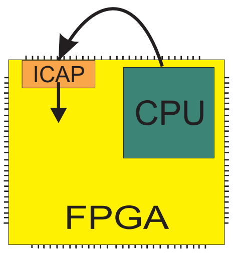

Figure 1. FPGA with ICAP-module

2.2 Local Resource Management

In order to cope with the increasing complexity that arises when managing a very large number of individual nodes we deploy a general purpose operating system. Linux serves our needs in several ways. It encapsulates the difficulties with accessing hardware and thus facilitates resource deployment through its abstract driver interfaces. For example the ICAP is incorporating into the Linux system with John Williams' device driver [7]. Especially the fully developed networking abilities are a convenient way to realize communication. The innumerable quantity of applications for any kind of need and furthermore adaption and development of software in high-level programming languages pose a big plus. The use of a standard Linux kernel and applications also speeds up development for another reason: the large community that exists and the vast database of solutions to standard problems.

2.3 Communication

As mentioned above our implementation enables standardized communication facilities. They provide us with a TCP/IP-stack to start with. Of course a certain application field might afford adapting the communication: A network that consists of battery driven nodes is more likely to yield better results when utilizing an energy aware routing protocol. The development boards we use provide RJ45- respectively USB-jacks, so besides the already implemented ethernet connection wireless LAN, bluetooth or ZigBee are feasible. For now remote nodes can be accessed and controlled via *telnet*. Datacan be transferred via HTTP with *wget*.

2.4 Adaptive Processing

Using partially reconfigurable hardware devices we are in the position to equip our nodes with a virtually adaptive behavior. To keep development easy at the beginning our nodes provide only one region of fixed size that can be reconfigured separately from the rest of the system as shown in figure 3 on page 12. With this, we are able to realize a hardware-software co-design for optimized performance. Tasks that can efficiently be computed in hardware may be executed in a specialized hardware module. Other services that are not available as a hardware module or are more efficiently solved in software are then worked off on the local CPU. We use the ICAP to allow the CPU to reconfigure the node itself to utilize free reconfigurable space or to displace unused hardware accelerators to save unnecessary electrical circuits and thus energy. The individual situational behavior of a node is determined by the controlling application which decides whether to replace a local hardware accelerator or to employ a neighboring node according to a set of parameters stored locally in simple files.

An example of such a parameter is the willingness of a node to reconfigure the local hardware to serve a remote nodes's request. Generally these parameters will be stored as relative quantities to system wide base values to keep them comparable. The parameter files can be written (adapted) to a changed environment, either by the control application itself or by another node. Since we are using a full Linux system it is also possible to impose access control to prevent crucial data to be undeliberately or maliciously altered.

2.5 Self-Organization

The nodes, that are now able to change their own behavior to a certain extend, need to organize themselves in some way. We therefore developed a simple concept to deploy our new features: **LMGS** - *Local Marketplaces, Global Symbiosis*. The work is distributed according to principles of supply and demand within the network. As we touched on previously the controlling and resource management is left to the CPU with the Linux operating system and applications running on top of it. The *real* work to do in the network will also be done in software and, where possible, in specialized hardware. We present **LMGS** in more detail in the next section.

3. Concept of LMGS

The simple idea is that every node does exactly what it deems to be the *best* according to its stored parameters. The minimal software to actively 'take part in the game' consists of two elements. A **customer** which issues requests to the network to have a certain job done and a **purveyor** that answers requests for jobs with the costs it charged if it would be commissioned. The effort a node makes to create the answer can vary widely. According to its computation power, knowledge and storage capacity this can reach from a simple return of standard values to a complex measuring where the load, the utilization of the node's components or the probability for the effectiveness of anticipated reconfiguration might be taken into account. In sensor networks for example communication is the most expensive action so a distribution of work should be chosen that minimizes overall traffic, maybe by executing a lot of tasks locally through reconfiguring the node.

3.1 Requests

Requests issued by the customer component of a node consist of a tuple as follows:

$$Request = \{Source,\ Target,\ Requester,\ Data\_Volume,\ Task,\ Max\_Hops\}$$

Where *Source* contains the data source for the task, *Target* the data sink and *Requester* the device which posted the inquiry. *Data_Volume* holds the quantum of data to be processed with *Task*. *Max_Hops* specifies the maximum number of times a request may be relayed by a node's neighbors.

Source, *Target* and *Requester* may be partly or completly identical. Given they are not, we included mechanisms to distribute jobs within a channel between data source and target. The values can be one-to-one identifiers of nodes or might as well contain wildcards to address groups of nodes so that, for example, modifications that have to be applied to a set of devices can be launched by a single command.

The *Data_Volume* will be taken into account when an offer is generated for the request. It usually influences the decision whether it is more appropriate to solve a task in software or in hardware and if the data may be processed remotely or if the communication costs for that would be too high.

In order to be able to specify a *Task* or a whole group of tasks, we suggest an hierarchical nomenclature which comprises every single service that can be rendered in the network under one root node.

As $Task$ in the request structure is not limited to one atomic job it may be a list of tasks. These lists may contain jobs that are to be processed sequentially or in parallel reaching from a single data-source to a single target, to complex data flows with multiple sources and targets.

Finally the restriction to *Max_Hops* ensures that inquiries are not simply flooded through the net but stay local working off jobs at a kind of in situ marketplace when sensible. This of course only if the local neighbors provide appropriate solutions to tasks at a reasonable price. The composition of this 'price' will be presented in section 3.2. The idea behind that is obvious: since we are able to reconfigure nodes to serve virtually any need, even small localized groups are highly adaptable and will be able to cope with most challenges with optimal efficiency. Thus data will in general be kept in a spatially narrow cloud and communication is minimized. Later in this section and in part 3.2 we will explain how this is accordable with the claim of super-regional cooperation.

To publish and find services to fill in a valid request basicly three mechanisms can be deployed. First one central directory server knows which node offers which services and has to be prompted for every job to work off. If it fails the whole network is paralyzed. New services have to be entered, causing additional traffic. Second, searches for certain services are flooded through the whole network. This is very flexible, but rather inefficient.

In this work, we developed a third approach, a hybrid solution between the two previously mentioned ones. Here data is flooded only within the closest neighborhood. Only if the answers from them are unsufficient, say because none of the next nodes wants to execute the requested task, the job is advertised again, this time with a higher number of maximum relays. Additionally nodes keep a more or less extensive list of other, remote hosts, that satisfy a certain request. In this manner not only local offers will be taken into account but also more distant ones which possibly better suit the current situation. The maintenance of this list is closely related to the structure of offers replied to a request which will be covered in section 3.2.

Generally, requests will be posed to the direct neighbors and to the issuer itself. A neighbor that finds the *Max_Hops* greater than one reduces that counter and sends the request to all it's neighbors. Especially the answer from the node itself is interesting in this regard: with the possibility to reconfigure, scenarios can be managed where communication cost is very high and nodes have to cut back on transporting lots of data through the net.

3.2 Offers

The response to a concrete query contains two elements: the cost-vector that the replying node is estimating for supplying the service and the local list of known providers that are also capable of satisfying this particular request: *Offer* = {*CostVector, List_of_known_providers*}

Costs mean figures given as multipliers of a base cost. For example the transmission of one byte of data over a wireless bluetooth connection will be a lot more expensive than over a wired ethernet link.

We identified three dimensions of costly actions: time consuming, energy consuming and space consuming. Thus a node's purveyor calculates the cost vector for a task A according to

$$C_A = \begin{pmatrix} Z_K & E_K & P_K \end{pmatrix}^T \cdot W$$

where Z_K denotes the cost for the local effort concerning time, E_K for energy and P_K or required space. W contains the willingness of the node to spend part of the specific resource to locally execute task A as a diagonal matrix. A battery driven node might for example want to lower its willingness to accept very energy consuming jobs as it runs low on battery power. The final cost-vector C_A is passed to the issuer as part of the offer.

The list of additional service providers is being built up through logging of messages indicating the commission of a node for a particular task or through deliberate writing. On the one hand devices that relay an acceptation message (basicly a request with a specially formulated task) to a node store the target node and task together with an expiration time. On the other hand nodes can advertise their capabilities by giving out a request to every other participant of the net to amend its local list of providers. If it intends to stay in the community for a longer period the expiration time may be set accordingly, attracting all sorts of requesters, locally and remote.

3.3 Negotiation Example

The flow of a negotiation bases on sending requests, retrieving responses, determining the most appropriate solution and commissioning a perveyor (figure 2). When evaluating the replies, the contained lists of alternative, maybe remote, providers may be taken into account and selected ones may be prompted for a bid. This enables the system to incorporate both: decentral, distributed computing as well as central services like the storage of gathered and processed data.

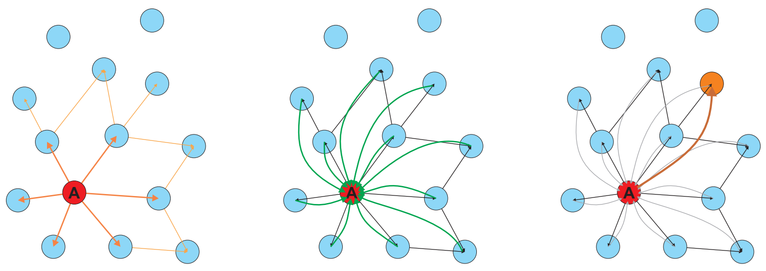

Figure 2. **LMGS**: issue a request (left picture), retrieve answers (middle) and commission job (right)

When enough answers came back in or after an amount of time the *customer* determines the optimal partner to commission. The weighted cost including communication is therefore calculated according to

$$A : CC_A = \vec{C_A} \cdot \vec{G_A} = \sum_i C_{A_i} \cdot G_{A_i}$$

Manipulating the components of G_A allows the node to emphasize a rather fast, energy-efficient or space preserving execution of a task. Depending on the computation-power and -willingness the node might accept the earliest offer or may run a multi-goal optimization on the plenty of data retrieved to find the best trade off.

4. Adaption to Sensor Networks

Sensor networks used to consist out of many very small sensor nodes to collect data and at least one data sink that gathers all the information and offers an interface to other nets. In scenarios with more complex data aggregations beacons are introduced which provide more computation power, collect data and transmit preprocessed information to the destination.

Our targeted application is a sensor network that is able to track an object optically over a large area where several cameras have to cooperate to keep the target within sight. A sensor node is now equipped with a video camera and has a certain computation power to extract features of object recognition from the taken pictures. Information about what the cameras detect will be send to the user's terminal. For this we built up a sample application for the XUP development board that captures the video signal from a camera, digitizes it and applies a filter to the stream to provide it for further processing. To start with we have implemented a mean-value filter to suppress noise and a set of sobel filters for edge detection in x- and y-direction.

The whole task is solved in hardware on the FPGA with the filter being partially reconfigurable at runtime. Figure 3 on page 12 shows the schematic (left hand side) and the physical layout (right) of our sample system. There the small boxed area is reconfigurable, whereas the bigger region encloses fixed logic like the ethernet controller or the SD-RAM controller. Both are connected via fixed interfaces, so called BusMacros. The lighter lines on the right picture are wire connections between logic blocks.

We evolved our designs to be partially reconfigurable using Xilinx's Early Access Partial Reconfiguration design flow [9] and made use of tools like PlanAhead, FPGA Editor and EDK [10].

Surely much of the video processing might be solved in hard core graphic processing units as well. Sure the programming of these is easier and more standardized than partial reconfigurable module implementation for FPGAs but flexibility and energy efficiency are the unbeatable advantages of our approach. FPGA nodes not only support hardware acceleration for graphic processing but also for virtually any task that is realizable in hardware. Hardware modules can be implemented using a high-level hardware description language like VHDL and thus are relatively easy to develop. Additionally, unused hard core units still waste space and cause stray current whereas unused FPGA modules can be physically disconnected from the current-carrying system, replaced by an other needed component or erased completely. This reduces the overall power consumption for a complete system with certain abilities that are on the one hand constructed out of many specialized hard core hardware components, on the other hand using one extremely flexible FPGA.

Using Linux several power saving modes like suspend to memory or to disk are feasible. This helps reducing the power consumption of the CPU when it's processing capabilities are not demanded. The node can even be set to sleep mode with only few circuits one to listen to a wake-up message.

LMGS is designed to be deployable on devices from both ends of the spectrum: very powerful or small, energy preserving nodes. In a sensor network the minimal equipment with **customer** and **purveyor** can even be further tapered: a sensor node generally doesn't have to issue requests thus running a **purveyor** module on it is enough to be able to obtain its measurements. The node simply answers to corresponding requests and will only be charged with a minimal amount of computations.

Since our system is flexible and expandable the loss of camera or computation nodes (beacons) can be compensated to a certain degree in terms of surveilled area as well as computation power for picture analysis. This is achieved by distributing the work the failing node contributed to other units. The data needed to configure a possibly missing hardware module into a neighboring node can be obtained from a central storage or, better when thinking of reliability, from a node that still has it in his cache. Using such a cache for hardware module definition files also serves another need: data that is used more often can be retrieved from a lot of nodes maybe even one in the vicinity of the own location thus the time to get and deploy such a module may be reduced.

Also new camera nodes or beacons can be added as the task of the network and therefore the demand for computation power changes.

5. Conclusion

This excerpt of our work presents our approach on how to deploy (partially) reconfigurable hardware in a network of semi-intelligent nodes to optimize overall resource demand and increase stability. Less resource usage because we keep tasks local were appropriate and spare expensive communication and more stability because services that have been offered by nodes that just dropped out will be distributed among the neighboring nodes. They will be prompted to reconfigure themselves and are then able to compensate the failing node as a community. We

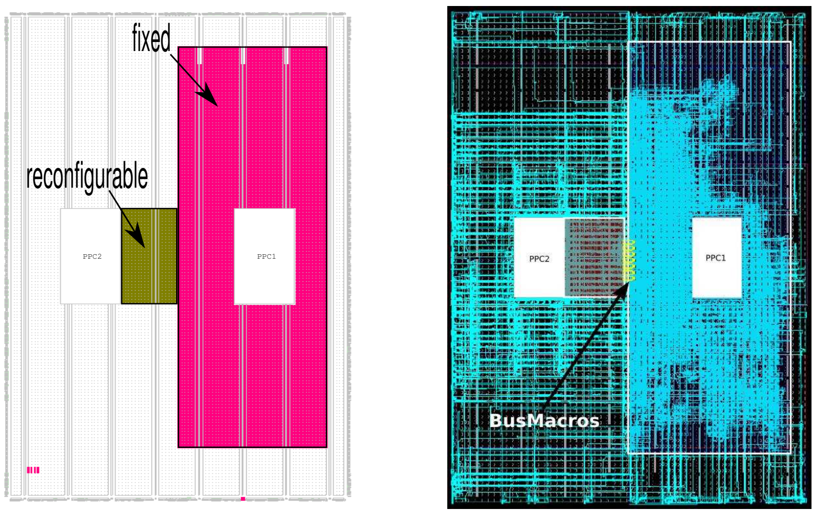

Figure 3. Layout of our sample system: the small emphasized area is reconfigurable.

created a working sample system for the Xilinx Virtex-II Pro XUP development board that allows us to initiate partial self-reconfiguration, send and receive files, controll distant nodes, being controlled remotely and to deploy standard applications. The local resource management is assigned to a Linux operating system, thus governed in software. Also high-level applications like load balancing of the computations and the network congestion or determining the node which suits the needs of our task the best are designed in software. The real work to do in the network, besides the administrative clutter, can now be accomplished using a hardware-software co-design. The filtering of a video stream or the detection of keywords in an audio file can now be split up in parallel and sequential parts and we can have the parallelizable parts run in one or more hardware modules. Still some tasks will be done in software but the increase in performance is significant.

References

[1] Brian P. Gerkey and Maja J Matarić. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, October 2002.

[2] D. Martin, A. Cheyer, and D. Moran. The Open Agent Architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1/2):91–128, 1999.

[3] Jingzhao Ou and Viktor K. Prasanna. Rapid energy estimation of computations on fpga based soft processors. In *IEEE International SoC Conference (SOCC)*,

2004.

[4] R. Shah and J. Rabaey. Energy aware routing for low energy ad hoc sensor networks, 2002.

[5] Suresh Singh, Mike Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Mobile Computing and Networking*, pages 181–190, 1998.

[6] Crossbow Technology. The mica2dot mote. http://xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2DOT_Datasheet.pdf.

[7] John W. Williams and Neil Bergmann. Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip. In Toomas P. Plaks, editor, *ERSA04: Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, pages 163–169, Las Vegas, Nevada, USA, June 2004. CSREA Press.

[8] Xilinx. Xpower. http://www.xilinx.com/products/design_tools/logic_design/verification/xpower.htm.

[9] Xilinx Inc. Early access partial reconfiguration user guide. Xilinx User Guide UG208, Version 1.1, March 6, 2006.

[10] Xilinx Inc. Homepage. http://www.xilinx.com/.

[11] Xilinx Inc. Two flows for partial reconfiguration: Module based or difference based. Xilinx Application Note XAPP290, Version 1.1, 2003.