

Dynamic Adaptation of Networked Reconfigurable Systems

Ram Kumar¹ Soheil Ghiasi² Mani Srivastava¹

¹Networked and Embedded Systems Lab (NESL), Electrical Engineering Department, UCLA

²Embedded and Reconfigurable Systems Lab, Computer Science Department, UCLA

¹{ram, mbs}@ee.ucla.edu, ²soheil@cs.ucla.edu

Abstract- Networked Reconfigurable System comprises of a heterogeneous ad-hoc network of reconfigurable devices interconnected in a wired or wireless manner. This system exploits the reconfigurable computing resource available over the network to improve application performance and increase the robustness of the system to failures. The resource constraints of an individual device can be overcome by utilizing the idle resources in the network. There is a need for providing run-time software support to the system to enable the dynamic adaptation. Resource monitoring, scheduling and task allocation at a network level become very critical issues in ensuring the proper functioning of the system. This paper proposes a framework for dynamic adaptation of the Networked Reconfigurable Systems. A Network Resource Manager (NRM) is developed that orchestrates the reconfiguration process at every device in the network. A Local Reconfiguration Manager (LRM) at every device abstracts the underlying hardware and provides a common interface and set of services to the Network Resource Manager.

I. INTRODUCTION

In recent years, the specifications for embedded systems have become increasingly demanding, requiring higher performance and reliability at lower power and cost. At the same time, these systems are increasingly proliferating into the environment [1] and being employed for a limitless range of applications. Such systems are exposed to a wide range of run-time events and tasks that cannot be predicted at design time. These observations promote the design and development of dynamically adaptive computing systems.

A. Networked Reconfigurable System

Run-time adaptation in embedded systems was traditionally done in software. However, the introduction of new high performance, high density field programmable gate arrays [2] and hybrid architectures embedding a processor core and a FPGA fabric on the same die [3][4] have enabled hardware reconfiguration for dynamic adaptability. Research efforts have been made in providing run-time support for this class of reconfigurable systems [5]. The run-time reconfiguration capability permits *temporal adaptation* of systems

wherein multiple configurations required to execute an application can be scheduled over time.

The capability of these systems can be extended manifold by interconnecting these devices to realize an ad-hoc network of reconfigurable embedded devices. A heterogeneous network can be envisioned that comprises of processors, FPGA and other reconfigurable resources communicating with each other through wireless or wired communication channels. The entire network can be viewed as one monolithic adaptive system, called the *Networked Reconfigurable System* that can be adapted in both *spatial* and *temporal* domain. Different parts of the system can be reconfigured independently in response to the external inputs.

Networked reconfiguration *enhances the performance of applications*. Sub-tasks of the application can be executed in parallel at various locations within the network. Often, a single device may be resource constrained to do a task. In such situations, the idle resources in the network can be utilized. The framework of networked reconfiguration *improves resource utilization*. Also if a part of a device gets damaged, it can avail the resources shared through the network. This *increases the robustness of the overall system to failures*.

B. Contribution

In this paper, we present a framework for the dynamic adaptation of networked reconfigurable systems. We propose a centralized entity named the Network Resource Manager (NRM) that is responsible for managing the reconfiguration mechanism of the devices belonging to the network.

A networked reconfigurable system can be modeled as a pool of computing resources, which are utilized by the tasks whenever they are initiated and are freed upon completion. The tasks can arrive at random times at arbitrary locations. This can result in an uneven demand for computation resources throughout the network. The NRM ensures an even distribution of the computation load throughout the networked reconfigurable system.

To support the NRM, there is a local entity on each device called the Local Resource Manager (LRM) that abstracts the underlying hardware and provides a uniform interface throughout the network. In our framework, the NRM and the LRM communicate with each other over the network to co-ordinate the reconfiguration processes on the devices.

This paper is based in part on research funded by DARPA NEST Program under Contract No. F33615-01-C-1906. The views expressed in this paper are the authors', and do not necessarily reflect those of the funding agency.

We proceed to give a detailed description of NETCAMS (NETwork of CAMeraS), a networked reconfigurable system. The Computing Model assumed to develop the framework is discussed in section 3. NRM and LRM are described in Section 4 and section 5 respectively. The paper ends with the current progress and conclusion.

II. NETCAMS: SYSTEM DESCRIPTION

NETCAMS is a network of IQinVision IQEYE3 [6] cameras deployed for intrusion detection and target tracking in an area of interest.

A. IQEYE3 Camera Architecture

The IQeye3 camera from IQinVision [6] is a network-enabled camera that can be configured to act as a web-server delivering live images to a web-browser. The main components of the architecture of the camera are shown in the block diagram in figure 1.

The IBM PowerPC processor is the central processing unit of the system. The image datapath is implemented in a Xilinx VirtexE FPGA with the capacity of 200K system gates. The CMOS image sensor captures the scene images and transfers the raw stream to the image datapath, which resides inside the FPGA, using the IEEE 656 standard bus. The image datapath initially synchronizes the raw stream to the system clock. The synchronized stream is downsampled, windowed and packetized. The packetized stream is stored in the SDRAM through a DMA transfer on the PCI bus. The stream is available on the network through the Ethernet. The PCI bus connecting the components is clocked at 33 MHz. The bus arbiter resides in the PowerPC.

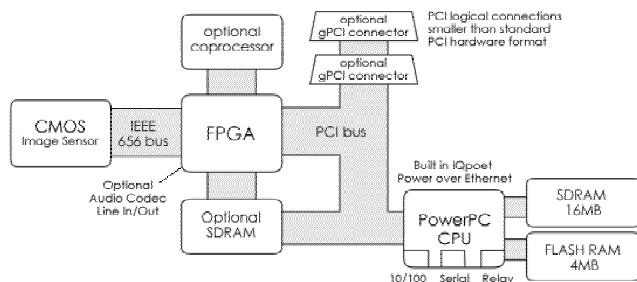


Fig. 1: IQeye3 camera architecture

The software architecture on the PowerPC provides an API that can dynamically reconfigure the FPGA at run-time. This feature enables developing applications on the PowerPC that can use the FPGA as a computing resource by scheduling different configurations over time. The Ethernet controller enables the camera to be controlled over the network. Thus, the camera with its FPGA can serve as a computing resource for the other devices in the network.

B. Networked System Architecture

The system comprises of three IQEye3 cameras and a laptop computer networked via Ethernet. The cameras are monitoring three separate rooms as shown in the figure. The system has been deployed for intrusion detection and target tracking in the rooms. The detection of the intruders is done by executing the *Background Subtraction* algorithm in the cameras. This algorithm continually subtracts the currently captured frame with the previous frame. The difference between the two frames is the measure of activity taking place in the scene. If the difference exceeds a threshold, an intrusion is signaled and the image that captures the intrusion is stored.

The intruder upon detection is then compared with a database of targets that need to be tracked. The comparison is done using *Template Matching* algorithm. The templates of the targets of interest are stored in a database. The image of the intruder is co-related with the target template and if the degree of co-relation exceeds a threshold, a match is signaled. The target of interest is tracked using the *Feature Selection and Tracking* algorithm. Thus, the complete operation of the system requires the execution of three algorithms viz. Background Subtraction, Template Matching and Feature Selection and Tracking.

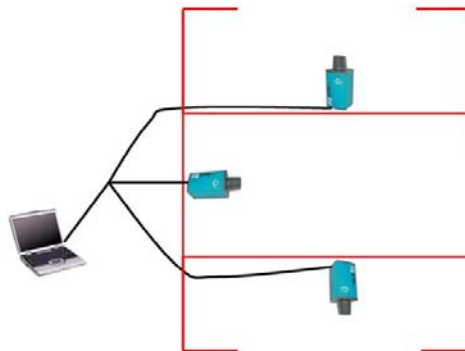


Fig 2: Networked System Architecture.

The three algorithms cannot be executed by the same camera due to the following reasons. Firstly, the Background Subtraction algorithm continually needs to be running after the intruder is detected in order to determine its location. Secondly, the *resource constraints* on the camera do not permit the simultaneous execution of Background Subtraction and Template Matching. Therefore, the Template Matching algorithm is instantiated in the other idle cameras in the network. Template Matching is an involved step as it requires co-relating the input image with multiple entries in the target database. However this task can be partitioned by dividing the database entries that need to be examined.

Significant improvements in the performance can be achieved by reconfiguring multiple cameras in the network to perform different parts of the template matching task. Finally, a hit in the database by any camera in the network will trigger a reconfiguration in the original camera to perform target tracking. Thus, the framework of dynamic networked reconfiguration enables the distributed execution of the application by utilizing the resources available in the network.

III. COMPUTATION MODEL

The dynamic networked reconfiguration framework enables triggered instantiation of operations across devices in the network. Multiple applications can be executed simultaneously on the network.

A. Task Graph Representation

The applications are represented using task graphs. The task graph is a directed acyclic graph where each node of the graph represents an operation and each edge of the graph represents the input for an operation and the events that initiate it.

The intruder detection and tracking application discussed in the previous section can be represented by the task graph shown in figure 3. Each operation in the task graph is initiated only upon the generation of an event from its preceding operation. For example, the Template Matching operation is started only after the Background Subtraction operation signals the detection of an intruder.

The task graph representation captures the data-dependencies amongst the operations. The data communication between the operations takes place through the network. The deadline for the application execution is annotated to the task graph.

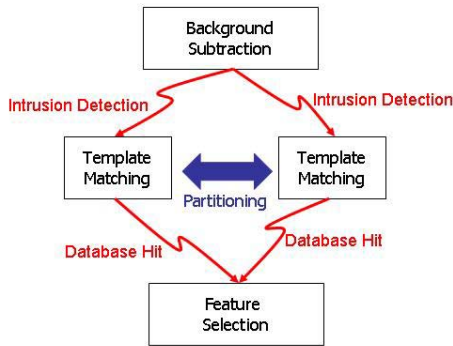


Fig 3. Task Graph representing the intrusion detection and target tracking application. The edges represent the triggers initiating the reconfigurations.

B. Application Execution

The application execution proceeds in a distributed manner over the networked system. Each operation in the application task graph generates an event upon completion. NRM receives the events and schedules the next set of operations and allocates them to the reconfigurable devices in the network. Reconfiguration commands are sent to the devices over the network. Each device in the network maintains a configuration database of the possible operations that can be mapped onto it.

In the NETCAMS system, every camera contains FPGA images of the three vision algorithms. Also every device in the network provides an API for the run-time reconfiguration of the FPGA. On receiving the reconfiguration commands from the NRM, the device LRM performs the dynamic reconfiguration using the API.

IV. NETWORK RESOURCE MANAGER

The Network Resource Manager (NRM) is the kernel of the adaptive configurable system framework. It is a centralized entity that is responsible for managing the reconfiguration across the network. The network resource manager constitutes of sub-modules viz. *Network Interface, Resource Monitor, Scheduler and Allocator*.

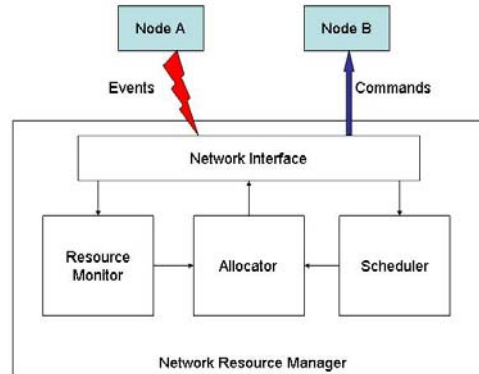


Fig. 4: Block diagram of the Network Resource Manager

A. Network Interface

The network interface module enables the communication of the Network Resource Manager with the devices in the network. The network interface implements a daemon that listens and captures the events generated by the individual devices. It parses the events to determine the source ID of the generator device and also the attribute of the event. The parsed information is sent to the Scheduler module. The network interface module also transmits the reconfiguration commands issued by the Allocator module to the appropriate destination devices in the network.

B. Resource Monitor

The networked reconfigurable system can be modeled as a pool of adaptive computing resource. The primary function of the resource monitor is to maintain a record of the utilization of the computing resources by all the devices in the network. Each device in the network can be addressed uniquely. The record for each device is indexed by its address. The record contains an identifier for the configuration currently mapped to the reconfigurable resource, the resource availability and the time when this reconfiguration was done on the device. For example, a record for one of the cameras in the NETCAMS system will be as follows:

IP Address	Configuration Mapped	Resource Status	Instantiation Time
Camera 1 128.97.93.56	Template Matching	Busy	500 time units

Fig 5. An example of a resource utilization record.

The resource monitor records are updated by the individual devices that convey their individual resource availability status whenever they are reconfigured or when a task execution terminates.

C. Scheduler

The scheduler determines the set of operations in the application task graph that are ready for execution and their deadlines. The scheduler initially computes a delay budget for all the operations in the task graph. The delay budget is the maximal permissible time for any operation to complete its execution and IO. The computation of the delay budget is done by assigning the worst-case execution time for every operation in the task graph. The critical path through the task graph is then identified. It is the path from the source operation to the destination operation that encounters the maximum delay. The operations on the critical path have zero slack. The slack for all the other operations is calculated and is added to their worst-case execution times to compute their delay budget.

A task graph is shown in figure 6 that comprises of seven operations and their worst-case execution times. The critical path is identified to be A, B, D, F, G. All these operations have zero slack and therefore a delay budget equal to their worst-case execution time. Operations C and E do not lie on the critical path and hence they have a positive slack as indicated in the figure. The delay budget for the operations C and E will be 35 and 30 respectively.

Each operation in the system generates an event upon completion. The events are picked up by the network interface and sent to the scheduler. On receiving the events, the scheduler decides the next set of operations to be scheduled from the task graph. Based on the delay

budgets of the operations and the current execution time of the application, the scheduler assigns a deadline for the completion of each operation that is to be scheduled next.

For example in the task graph shown in figure 6, when operation A signals completion, the operations B and C are scheduled. If operation A is completed in 5 time units, then the remaining 5 units of its delay budget are added to the delay budget of operations B and C. Therefore, operation B will have a deadline of 20 time units and operation C will have a deadline of 40 time units.

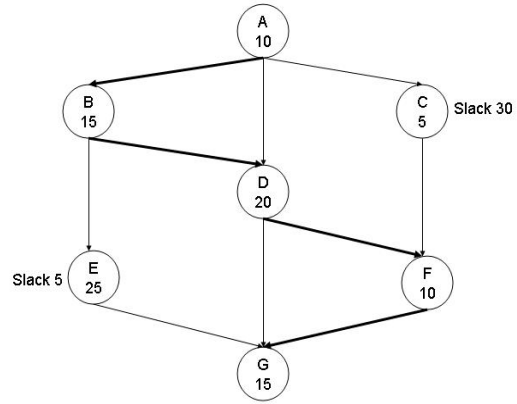


Fig 6. Task graph with execution times of operations

D. Allocator

The allocator module receives the operations to be scheduled with their deadlines and allocates the devices to perform those operations. It also initiates the transfer of the output data from the parent operation in the task graph to the device allocated for the current operation. The allocator maintains an estimate of the execution time of an operation on each one of the devices in the system. The resource monitor is queried by the allocator to determine the availability of the devices. The allocator assigns the operation to the available device, which has an estimated execution time that is less than the deadline.

In case no device is available, the allocator computes the time after which the first device will be free. This can be done by reading the “Instantiation Time” field in the record of that device in the resource monitor. The time at which the device would be free can be computed by summing the estimated execution time of the *operation currently allocated* to that device and the “Instantiation Time”.

For example from figure 5, if the estimated execution time of Template Matching on camera 1 is 20 time units, then the device would be free at 520 time units. If the current time is 510 time units, then the allocator can determine that the camera 1 will be free in 10 time units. The estimated execution time of the *new operation* to be

allocated on that device is added to the time after which the device would be free. For instance, if the new operation to be instantiated is Feature Selection and Tracking and it has an execution time of 30 units on camera 1, then it would terminate after 40 time units due to the initial delay of 10 time units. If this time is less than the deadline, the operation is allocated to the device. This process is repeated till the first feasible device is found. If there is no device found, the operation is allocated to the device that has the minimum termination time.

V. LOCAL RECONFIGURATION MANAGER

The Networked Reconfigurable System will comprise of heterogeneous devices with different architectures and implementation technology. The Local Reconfiguration Manager (LRM) acts as a hardware abstraction layer presenting a common interface to the NRM. The LRM also provides a basic minimal set of services that would enable the device to be reconfigured dynamically over the network. The LRM comprises of a Network Interface, Main Thread, Reconfiguration Queue, Loader Thread and a Configuration Database. The main components of the LRM are shown in the figure.

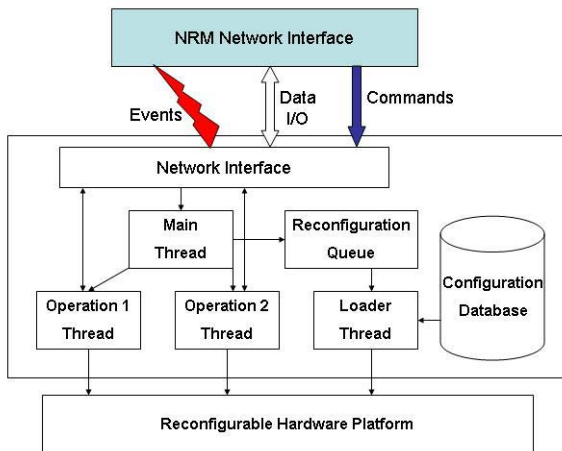


Fig 6. Block diagram of the Local Reconfiguration Manager

The network interface transmits the events generated by the device to the NRM. The allocation commands issued by the NRM are parsed by the network interface and sent to the main thread. The main thread coordinates the functioning of all the operations that are allocated to the device. A new thread is spawned for every operation that is allocated to the device. The operation is added to the reconfiguration queue. The top element of the reconfiguration queue is called the *current operation*. The loader thread is started by the current operation thread. The configuration corresponding to the current

operation is fetched from the Configuration Database. The loader thread programs the hardware using the runtime reconfiguration API. The current operation thread signals its instantiation to the NRM. Only the current operation thread interacts with the hardware. The data input for the current operation thread is received through the network interface. Upon termination of execution, the current operation thread signals to the NRM and removes itself from the reconfiguration queue. The thread stays alive till all its output is routed to the appropriate destination using the network interface, after which it dies.

VI. CURRENT WORK IN PROGRESS

Currently, most of the work in progress involves implementation of the algorithms discussed in Section 2. The algorithm is described using Behavioral VHDL. The design flow consists of behavioral synthesis of the design using the Synplify Pro from Synplcity [8]. The generated net-list is then input to the Xilinx ISE [9] that performs the mapping, place and route and bitfile generation operations. The bitfile is then built into a flash image using the proprietary tools from IQinVision. Upon the completion of the implementation phase, experiments would be conducted to observe the latency and overall resource utilization.

VII. CONCLUSIONS

In this paper, we have described a framework for dynamic adaptation of Networked Reconfigurable Systems. We first motivate the benefits of harnessing the computing capability available in the network in the form of idle reconfigurable hardware devices. Networked reconfiguration helps overcome resource constraints, improves application performance and increases robustness of system to failures.

We have identified the key components necessary to enable networked reconfiguration: Network Resource Manager and Local Reconfiguration Manager. There is a detailed description of the sub-modules that constitute the Network Resource Manager viz. Network Interface, Resource Monitor, Task Partition Module and Scheduler. The working of the Local Reconfiguration Manager has also been explained.

In future, the dynamic reconfiguration framework can be extended to devices that can support partial reconfigurations [7]. In such devices, multiple operations can be instantiated simultaneously which can greatly improve the granularity of the resource utilization and also the application performance. The use of hardware byte codes to transfer configurations across the network is also intriguing.

REREFENCES

- [1] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks.", ACM *Mobicom '99*, pp. 263-270, August 1999.
- [2] Xilinx Platform FPGAs, <http://www.xilinx.com/products/platform>
- [3] Atmel FPSLIC. <http://www.atmel.com/atmel/products/prod39.htm>
- [4] J. R. Hauser and J. Wawrzynek, "Garp: A MIPS processor with a re-configurable co-processor", in *Proc. IEEE Symp. FPGAs for Custom Computing Machines*, Napa, CA, Apr. 1997, pp. 12-21
- [5] J.S.N. Jean, K. Tomko, V. Yavagal, J. Shah, and R. Cook, "Dynamic Reconfiguration to Support Concurrent Applications," in *IEEE Transactions on Computers*, Special Issue on Configurable Computing, Vol. 48, No. 6, pp. 591--602, June 1999
- [6] IQinVision IQeye3 Smart Cameras, <http://www.iqinvision.com/prd/IQe3.htm>
- [7] Edson L. Horta, John W. Lockwood, David E. Taylor, David Parlour, "Dynamic Hardware Plugins in an FPGA with Partial Run-time Reconfiguration", Design Automation Conference (DAC), New Orleans, LA, June 10-14, 2002.
- [8] Synplicity Products: Synplify Pro, <http://www.synplicity.com/products/synplifypro>
- [9] Xilinx ISE Design Tools Center, http://www.xilinx.com/ise/design_tools