Adaptive Computing and Run-time Reconfiguration

Sandeep Neema, Ted Bapty, Jason Scott
ISIS, Vanderbilt University
neemask@vuse.vanderbilt.edu

Adaptive Computing is a relatively new research area [1]. The main thrust of this research area so far has been on programmable/configurable hardware. Significant successes have been reported. The programmable gate array (FPGA) technology has matured producing very high-density gate arrays (~1 million gates) with lower configuration times. A variety of hardware platforms comprising of microprocessors and FPGA's have been developed. A host of applications have been ported over to these platforms demonstrating significant speedup over the conventional microprocessor based implementation [2].

The promise of adaptive computing however extends far beyond that. True adaptive computing systems will be able to adapt to changes in environment at runtime, without compromising the consistency and real-time properties of the system. To that effect the main challenge that needs to be addressed is run-time reconfiguration. Some researches have demonstrated restricted capability of run-time reconfiguration, however most of these approaches are too application specific and lack adequate formalism. Additionally, the run-time reconfiguration effort in these researches is directed solely towards reconfiguring the programmable hardware. This is clearly inadequate as most high-performance systems involve a mix of hardware and software. We therefore believe that a more comprehensive approach needs to be taken and the problem of run-time reconfiguration needs to be addressed at multiple levels. In this paper we discuss some of the issues involved in run-time reconfiguration and present our approach to address these issues.

Difficult issues arise in supporting run-time reconfiguration in a heterogeneous system consisting of DSP's, RISC processors, FPGA's and ASIC's at multiple levels.

a) Low-level Hardware: A host of problems can occur during reconfiguration that may destroy the consistency of the underlying hardware. Loss of hardware consistency can have many negative effects, ranging from temporary loss of performance to hardware damage and total/permanent system malfunction. Some of the possible scenarios are 1) Port contention may occur when bi-directional ports are improperly initialized, a reconfiguration event is not properly sequenced/synchronized or if an improper/inconsistent design is implemented. If two connected drivers are enabled and if the resistance is sufficiently low, permanent physical damage can occur to the circuits. 2) Token loss or duplication can result from incorrect initialization or a loss of communication integrity. Tokens represent the status of empty or full slots in a communication interface. An extra token on the sender side can cause too much data to be sent, resulting in a FIFO overrun. A lost token can effectively block a communication port, resulting in a system deadlock. 3) Device state maintenance refers to the control of a complex external hardware device, such as an attached processor or storage device. In controlling an external device, the controlling computational component must maintain an accurate representation of the device's state. If a reconfiguration occurs during a state transition within the device, or if the reconfiguration modifies the computational component's representation of the device, there can be a state mismatch. This can result in improper commands being sent to the device, or in a deadlock where both components are waiting on each other for triggering events.

b) Software/OS: Software issues can present a larger challenge to dynamic system reconfiguration. While the hardware built into standard microprocessor devices protects against low-level hardware conflicts, there are many more details that must be managed. The internal state of software must be managed under reconfiguration. Modern operating systems have evolved to support the flexible implementation of multiple tasks, with dynamic addition and removal of tasks on a single processor in the form of time-sharing and/or multitasking, and Real-time kernels allow time critical tasks to be dynamically scheduled on a single processor. These kernels typically do not address the consistency of dynamic reconfiguration for distributed networks of tasks. As a result memory leaks could occur that would adversely affect long-term reliability. Task structure mismanagement can happen resulting in extra tasks executed by the kernel, with a loss in performance. In a message passing system messages in transit can be delivered when the receiving process no longer exists, resulting in mismatched messages and communication errors.

c) Application: The primary concern at this level is state preservation during reconfiguration and maintaining real-time properties of the application. The loss of consistency at the application level could render the system entirely unusable. Some possible scenarios are: 1) The elevation and azimuth controller in a missile requires a continuos update on the position and velocity of the missile. A loss/corruption of this state information could cause the missile to go completely haywire. 2) An external system may require signal output continuity and/or continuous first derivative properties. A reconfigurable system that swaps filters online, the new filter can operate out of sync with the original filter. A rapid switchover will create a discontinuity in both the signal and its first derivative. 3) The system can fail to maintain real-time constraints during reconfiguration. If the reconfiguration cannot be completed in sufficient time, deadlines will be sacrificed. In addition, the time-base can be shifted, resulting in a skew in system output period.

Our approach consists of two core elements. We developed a computation model for dynamically reconfigurable systems, and we implemented a runtime system architecture that executes the computation model while addressing the consistency issues raised above. We represent a dynamically reconfigurable system as a multi-mode system, where the system has well-defined modes of operation, rules for transition between these modes of operation and a computation algorithm for each mode of operation. The algorithm is described as a data-flow graph, where the nodes of the graph represents primitive operations and the arcs between these nodes represent the flow of data between the nodes. Reconfiguration in this model corresponds to the transitions between modes of operations and upon reconfiguration the currently executing data-flow graph is replaced by the next data-flow graph. In this model of computation we also model the inheritance relationship between different data-flow graphs, so that internal state of nodes in the data-flow graphs could be preserved and transitioned. This enables the issue of maintaining Application State. We have implemented design tools that allow designers to graphically represent their reconfigurable system in the formalism described above [3]. We have also implemented generators that can automatically generate configuration specification for the runtime architecture described below from the models.

The runtime architecture comprises of a large-grain dataflow kernel for the DSPs and RISC processors, a virtual HW kernel for the FPGA's and a reconfiguration controller that manages the reconfiguration process (Figure 1). These together implement a large-grain reconfigurable dataflow architecture. The reconfigurable dataflow kernel provides services for dynamic creation/removal of tasks, dynamically modifying the inter-task communication maps. It also provides services/primitives for storing/retrieving internal task state and global System State. The software kernel is responsible for maintaining the software consistency. The virtual HW kernel exists as a concept providing semantically equivalent data-flow architecture on the FPGA's. The reconfiguration controller is responsible for maintaining the overall state of the system, and accomplishing run-time reconfiguration.

References

[1] J. Villasenor, W. H. Mangione-Smith, "Configurable Computing," *Scientific American*, pp.66-71, June 1997.

[2] M. Rencher, B.L.Hutchings, "Automated Target Recognition on SPLASH 2", in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, April 1997.

[3] J. Scott, T. Bapty, S. Neema, J. Sztipanovits, "Model-Integrated Environment for Adaptive Computing," MAPLD'98, Proceedings of the 1998 Military and Aerospace Applications of Programmable Devices and Technologies Conference, Sept 15-16, 1998, Greenbelt, MD

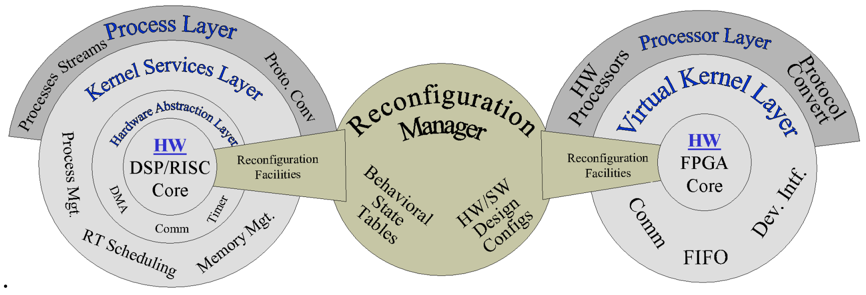

Figure 1: Run-time system architeture