The Future of Reconfigurable Computing

Nicholas J. Macias
nick.macias@worldnet.att.net

Introduction

The field of reconfigurable computing has attracted much interest recently, with research journals and the popular press announcing paradigm shifts, third generation reconfigurable systems and artificial brains. One of the most promising application areas for reconfigurable systems is general purpose massively parallel systems (GPMPS). The reason is that while a GPMPS must contains a large number of general-purpose processing units, the system as a whole is much more than the sum of the programming of these individual units. For example, the interconnection among the processing units is a vital characteristic of the whole system. Likewise, the architecture of each processing unit has a critical effect of the system's overall performance. Traditional fixed-hardware parallel processors only allow modification of the software within each individual processor. To build a truly general purpose massively parallel system, one needs control over the hardware as well. This suggests that a GPMPS might be implemented as a large general purpose reconfigurable system.

Unfortunately, almost all work to date on reconfigurable systems has focused on the model of an externally-controlled reconfigurable device, such as an FPGA, attached as a coprocessor to a general-purpose controller, such as a PC. For example, at the 1998 IEEE Symposium on FPGAs for Custom Computing Machines [CCM], 22 papers were presented which discussed particular reconfigurable systems. Of those, 21 describe coprocessor-style systems (the notable exception is the Plastic Cell Architecture from NTT [Nag], a system with the capacity for self-reconfiguration, but seemingly under external setup and initiation).

This separation of the reconfigurable circuits from the circuits which control them is most unfortunate, for while improvements in technology make it easy to increase the number of reconfigurable circuits within a device, the task of controlling those circuits becomes correspondingly harder. This is a well-acknowledged problem in the literature, and various solutions have been proposed to work around it, such as combining the reconfigurable device (RD) and the controller on the same chip, or implementing higher-bandwidth interfaces between the RD and the controller. Such techniques, however, don't actually eliminate the underlying problem with these systems, rather they lessen its impact.

The Key to a General Purpose Reconfigurable System

What then is the key to a general purpose reconfigurable system? The common theme seems to be the need for internal control. As long as reconfigurable devices depend on external agents to perform their reconfigurations, reconfiguration will remain a fundamental bottleneck in the system.

This is not unlike the state of affairs in the 1940's with machines such as the ENIAC. While these early machines processed data electronically, their programming was done mechanically by setting switches and connecting physical cables. Thus, though these systems were programmable, they were certainly not self-programmable, as they required an external agent (the operator) to perform programming changes. While the system could produce electrical outputs to direct that external agent, the system itself could not change its own programming.

When John von Neumann modified the design of the ENIAC to make it electrically-programmable, he introduced the capacity for self-programmability. Key to this design was a fundamental duality in the electrical pulses stored within the system's memory. They could represent either data or code, depending on how the system interpreted them. Von Neumann's design for the EDVAC was later used by others to design the EDSAC, arguably the first instance of what we now call a von Neumann machine [Neu].

The analogy to reconfigurable hardware is straightforward. Current reconfigurable devices are not self-reconfigurable. They require assistance from an external agent to modify their configuration. While the reconfigurable device can generate outputs which dictate the behavior of that external agent, the device can not autonomously reconfigure. If reconfigurable devices are to be used to implement massively parallel systems, particularly those whose hardware is to change at runtime, they must be autonomously reconfigurable.

The Processing Integrated Grid

The Processing Integrated Grid or PIG [US Patent #5,886,537, issuing March 23, 1999] is a new type of reconfigurable device. It is fine-grained, extremely scalable, and completely capable of autonomous self reconfiguration. It consists of an arbitrarily-large regular array of simple processing elements called cells. Each cell is connected only to its immediately adjacent neighbors. Neighboring cells exchange a single bit with each other, and that bit can represent either data or code, depending on how the receiving cell interprets it. This is not dissimilar to the data/code duality in von Neumann's design for the EDVAC. This duality is in fact one of the most important features of the PIG.

If a cell is in D (DATA)-mode, meaning it is interpreting its incoming bits as data, it uses an internally-stored program to map its inputs to outputs, which it transmits to its neighboring cells. A collection of properly configured D-mode cells can thus be used to implement things such as wires, memories and state machines. As such, the PIG can be used like any other reconfigurable hardware device.

If, however, a cell is in C (CODE)-mode, it interprets incoming bits as new program information, and modifies its internal program based on the received bits. This means each cell of the PIG can be locally reconfigured, purely under the control of nearby circuits.

The current mode of a cell is controlled by its neighboring cells, and therefore any cell can affect how its neighbors interpret incoming bits. This means that any cell has the ability to read and modify the configuration of any neighboring cell.

This is the key to the PIG's power. It does not need any assistance from an external agent in order to reconfigure its circuits. As the number of cells in the system increases, so does the number of potential reconfiguration controllers. As the parallelism of the system increases, so does the parallelism of the reconfigurator. Circuits which need to be monitored and potentially modified based on local conditions need not interact with a shared, external reconfigurator through a narrow channel. The reconfiguration circuitry itself can be configured in close proximity to the circuits being monitored, and the controller can be dedicated to controlling a single subsystem. Hence the PIG achieves **massively parallel reconfiguration**.

Sample PIG Circuits and Applications

A number of circuits have been developed for the PIG, and a brief overview of these circuits may help illustrate the PIG's unique nature compared to other reconfigurable devices such as FPGAs. In all the examples which follow, the entire circuit is implemented on a array of PIG cells, without the need for any external hardware. All specialization of the behavior of the system is achieved through software reprogramming of the cells.

Some of these circuits developed so far include:

- Hardware libraries contained within the PIG itself, which contain templates for various cell configurations. Circuitry implemented on the PIG allows any template to be selected and used to guide the programming of a target cell.
- Wire builders, which dynamically construct circuits to allow access to remote (non-neighboring) cells. In this circuit, the wire itself is used to build more wire, to extend the reach of the circuit.
- Arithmetic circuits which detect an impending overflow, and respond by synthesizing hardware to extend the wordsize of the circuit. The entire arithmetic unit itself is implemented in PIG cells, as is the circuitry which monitors the unit's behavior, detects the upcoming overflow, and reconfigures additional cells to extent the unit's hardware. The monitor and control circuitry is located physically near the arithmetic unit itself, and is dedicated to

monitoring that unit alone. Hence, if multiple units are operating in parallel, they are also monitored and potentially reconfigured in parallel, without any shared-resource bottlenecks.

- Space-filling circuits, which spread outward through the PIG to allow placement of remote circuits.
- An entirely self-contained self-replicating circuit.
- Circuits which authenticate reconfiguration attempts to allow controlled reconfiguration of certain cells, while preventing uncontrolled reconfiguration of those same cells.

These circuits tend to be foundational in nature, that is, they are building blocks for more general applications. Such foundational work is where most development efforts on the PIG have occurred. Still, a few higher-level applications have been developed, including:

- A fully parallel genetic algorithm for evolving digital circuits [Mac]. This algorithm features a spatially-distributed population of circuits, each with the ability to evaluate and mate in parallel with all other members of the population. Hence the time for a single evolutionary cycle is independent of the population size. Additionally, since the circuits being evolved are implemented on PIG cells, which are capable of self-reconfiguration, this setup can be used to evolve self-modifying hardware.
- A system for storing multiple configurations of a subgrid, with the ability to rapidly reconfigure the entire subgrid from one configuration to another. In this system, the reconfiguration time is independent of the number of cells in the subgrid. This is achieved by having each cell locally reconfigured in parallel with all other cells. Commercial manufactures have begun redesigning FPGA masks to incorporate this feature into their devices**. On the PIG, this ability is introduced through simple programming**. This is quite characteristic of PIG applications.

In all these example, the underlying system is always the same. It is a regular array of PIG cells with a uniform interconnection scheme. The cells are always identical to each other, except for their configuration information. All specialization is done via programming of the cells.

Conclusions

The PIG is a general purpose self-reconfigurable system. It's capacity for self-reconfiguration makes it an ideal building block for a reconfigurable general purpose massively parallel processor. The ability of the system to locally monitor and reconfigure nearby circuits allows for massively parallel reconfiguration, an ability which is critically missing from today's reconfigurable systems.

A very small PIG has been built in cMOS and successfully tested. The system featured an inherent fault tolerance, owing to its extremely regular internal structure. Fabrication errors in one area of the chip generally had no effect on cells in other areas of the chip. This suggests the possibility of wafer-scale integration for manufacturing a large PIG.

The PIG is completely scalable. Since cells only connect to adjacent neighbors, two PIGs can be attached side-by-side, with their border cells connected to each other, and the result is a larger PIG. There are no address buses which grow wider as the grid size increases.

This uniformity of the PIG’s structure, and its flexibility in interconnection topology, make it an ideal candidate for newly-emerging technologies, such as nanotechnology and molecular computing. As these technologies are probably best suited for building large regular arrays of a simple circuit, they are well suited for building a large PIG.

Finally, something must be said about the types of applications which the PIG is best suited to. Certainly, reconfigurable hardware can’t match the speed of an ASIC. This is especially true with something as fine-grained as the PIG, in which even simple wires are implemented by a collection of cells. The PIG is therefore not intended as a replacement for the von Neumann processor. The tasks it is best suited to are those which require both massive parallelism and massively parallel reconfiguration. In such tasks, the lower speed of the individual processors and their interconnection delays is eventually outweighed by the speedup from the parallelism.

In 1951, Maurice V. Wilkes described the ability of the EDSAC to modify its own operations, stating “This facility of being able to modify the orders in the program by performing arithmetical operations on the numbers representing them is of great importance, and it is perhaps the feature most characteristic of program design for machines like the EDSAC” [Wil]. While this feature clearly seemed important to Wilkes, particular applications were probably difficult to imagine in 1951; the only example given by Wilkes was for reducing the amount of code needed in a loop to add a set of numbers. Today, we recognize this self-programming feature as the key to concepts such as object libraries, multitasking, virtual memory, and most other aspects of modern operating systems.

Similarly, the ability of a circuit to analyze and modify its own configuration is likely the feature most characteristic of applications well suited to implementation on the PIG. This is so different from the conventional computing models of today that such applications are difficult to imagine. We are only now beginning to work with systems which feature any reconfigurable hardware at all. The leap to self-reconfigurable applications is a large one, but represents a fundamentally new and powerful way to solve problems. As our experience and understanding of the PIG grows, we hope to be able to better utilize this unique ability.

References

[CCM]

[Nag]

[Neu]

[Mac]

[Wil]