

Scribe notes

I. Summary

General purpose processors don't always succeed in meeting specialized application's demanding needs. Reconfigurable computing is a very good approach to meeting these particular needs. This type of computing will change the way computing systems are designed and used. *Piperech* is a reconfigurable fabric, which combines the flexibility of a general purpose computer, with the efficiency of custom hardware, to achieve extreme performance speedup. It is a reconfigurable pipelined architecture composed of configurable logic and storage elements and a rich interconnect network.

Reconfigurable computing was until recently based on FPGAs and failed to meet the requirements of the marketplace. Some of the problems with FPGAs are:

- *Logic granularity*
- *Configuration time*
- *Forward compatibility*
- *Hard constraints*
- *Compilation time*

Piperech overcomes these problems and maintains outstanding performance by virtualizing the hardware. A system using *Piperech* can:

- *Increase flexibility*
- *Decrease design time*
- *Extend system life*
- *Reduce part count*
- *Reduce development and maintenance costs*
- *Reduce chip fabrication costs through defect tolerance*

A reconfigurable fabric outperforms a general-purpose processor for computations that:

- *Operate on bit widths that differ from the processor's basic word size*
- *Have data dependencies that let multiple function units operate in parallel*
- *Contain basic operations that combine into a single specialized operation*
- *Can be pipelined*
- *Enable constant propagation to reduce operation complexity*
- *Reuse the input values many times*

A reconfigurable fabric can be used as a functional on the main CPU, as a tightly coupled coprocessor, or a loosely coupled attached processor on I/O or memory bus. To overcome configuration's static nature the pipeline reconfiguration concept is adopted. With this technique, a large logical design can be implemented on a small piece of hardware through rapid reconfiguration of the hardware. The compiler is no longer responsible for satisfying fixed hardware constraints. It is a method of virtualizing pipelined hardware application designs by breaking the single static configuration into pieces that correspond to pipeline stages in the application. Thus it becomes possible to perform the computation, even though the whole configuration is never present in the fabric at one time.

Piperech devices are composed of a set of physical pipeline stages, or stripes. Each stripe is composed of interconnect and processing elements which contain register's and ALUs. An ALU

is composed of look-up tables and extra circuitry for carry-chains, zero detection etc. Through the interconnect, the processing elements have access to outputs of previous stripes or outputs of elements on the same stripe. Any feedback must be contained within a stripe and cannot go to a previous stripe.

A good compiler is necessary for creating an efficient system, as good as the hardware fabrics. The compiler trades off configuration size for compilation speed. Thus fitting a design into a limited physical space is not a primary concern. The compiler uses as a source language a *dataflow intermediate language*, a single-assignment language with C operators that doesn't require programmers to specify the bit width, thus being able to generate configurations for any pipelined reconfigurable architecture. The to the compiler's speed is the *place-and-route* algorithm, which is a deterministic, linear-time, greedy algorithm, which runs between 2-3 orders of magnitude faster than commercial tools. However some efficiency is sacrificed for compilation speed.

The performance of the *Piperench* is evaluated by the use of nine important kernels:

- ***Automatic Target Recognition (ATR)***
- ***Cordic (Honeywell timing benchmark for Cordic vector rotations)***
- ***1D Discrete Cosine Transform (DCT)***
- ***2D Discrete Cosine Transform (DCT-2D)***
- ***Finite Impulse Response (FIR) Filter***
- ***International Data Encryption Algorithm (IDEA)***
- ***Nqueens (N-queens problem)***
- ***Over (Porter Duff Over operator)***
- ***PopCount (Population Count Instruction)***

CAD tools were used to synthesize a chip in a quarter micron technology. Custom layout was used for the interconnect and the number of strips was calculated to fit a $5 \times 10 \text{mm}$ area. The *Piperench's* performance is compared to the performance of an Ultrasparc II running at 300MHz and found to be better. Also the *Piperench IDEA* implementation is compared to other implementations and is found to outperform all of them and especially the *IDEACrypt Kernel* from *Ascom Systec Ltd.*

The *Piperench* reconfigurable architecture yields tremendous performance improvements. These reconfigurable devices can serve many purposes, and are likely to be used as *off-the-shelf* parts with reduced cost. This kind of computing has the potential to become a vital component in the next generation of computation devices, since the future processors will no longer benefit from *Moore's Law*.

II. Discussion

Piperench is a reconfigurable architecture that is joining the steady stream of alternate computing architectures that emphasizes processing with highly flexible computing fabrics. The principal difference when compared to using ordinary microprocessors is the ability to make substantial changes to the data path itself in addition to the control flow. *Piperench* is not different from other reconfigurable devices. One of the earliest issues raised was the advantages of *Piperench* over FPGAs. FPGAs are programmed after fabrication to solve virtually any computational task, as long as the task fits in the device's finite state and operational resources. Though it shares the same qualities of flexibility of use with less number of devices as compared to regular processors, *Piperench* has qualities that allow it to become a better force for mainstream computing. It has the ability to reconfigure faster than FPGAs due to the ability to pipeline the configuration. This

also allows the user not to be stuck with hardware constraints that limit the kernel that can be run on the FPGAs. Other advantages such as faster compilation time, capability of PIPERENCH to easily integrate and benefit from future chip technology were also listed but were not part of the discussion.

After talking about the important qualities of PIPERENCH, discussion moved to looking at some of the applications that would be suitable for this architecture. Some of the points raised pointed out that in a hybrid system where a reconfigurable processor is complementing a standard processor, it might be reasonable for the reconfigurable processor to implement the computationally intensive parts in a pipeline. Even though the reasoning was not brought up in class, the main advantage to using such an approach would be to embed the computation on a fabric that is particularly configured to handle it. This would allow us to save instruction band-width and control the overhead by moving the intensive computation to a fabric that is dedicated to it by realizing all possible realms of parallelism. Applications where the reconfigurable process might be suited for would be ones where the processing bit size is different from the main processor's bit size (running a 64-bit algorithm on a 32-bit system), where there is a lot of parallelism, and where there is a lot of feedback computation.

As noted above and in the paper, it was pointed out that the reconfigurable part can be located at three different places relative to the main processor. The reconfigurable fabric can be an attached processor on the memory bus where there is a connection to the main processor only through the bus/wire. This makes it easy for augmenting standard processors with a new fabric; however, this setup is limited by the bandwidth caused by the wire/bus. This forces us to use applications where the fabric rarely talks to the main processor.

Another possible setup is the fabric acting a co-processor which makes it closer to the main processor allowing us to have bigger bandwidth. This correspondingly allows applications that need a bit more communication with the main processor. Finally, this journey towards getting close to the processor goes one more step in the third possible setup for the fabric which puts it as a functional unit in the processor. This allows faster computations due to access to the register, which introduces ambiguities of where the state can be saved and if the fabric is able to access memory directly.

Since the reconfigurable architecture is not directly related to nanotechnology, a lot of time was spent discussion on why such architecture would be suitable in the nano world. This applicability is even more in doubt since the main goal of PIPERENCH is to get rid of constraints that are introduced due to limited hardware devices using virtualizing of the hardware through pipelines. On the other hand, the one sure aspect of any nano approach is that we can depend on having tremendous amount of hardware devices. Even the fault and defect tolerance mechanisms depend highly on having such vast hardware resources. The class quickly realized that it not the pipelining that is of interest, but its ability to quickly reconfigure itself. There are two ways that PIPERENCH can be attractive to a nanotechnologist and they both deal with tolerance. Nanodevices deal with defects by first identifying the defects and then looking for other places in the chip/substrate area that can be used to do correct computation, again relying on the fact that there are lots of other defect-free hardware devices.

We can use a *Piperench* like architecture to make sure that anytime we detect a defect in our device, we can quickly reconfigure it to another area. This makes it possible to drastically lessen the amount of chip testing necessary. Because as long as we are kind of sure that the chip has some good parts, we do not have to worry about checking every aspect of the chip. If we find a defect at run time (not manufacture time), we can just reconfigure and move on. This could also translate to using nano devices that may have a large defect rate. The large defect rate might be able to be compensated by the large number of devices possible AND a pretty fast reconfiguration architecture like *Piperench*.

Piperench does not have to be restricted to defect tolerance but can be extended to fault tolerance as well. In the nano implementation, the fabric would not be a co-processor or an attached processor but a functional unit in the main processor. Along with the vast hardware devices possible, we may have a lot of fabrics available for reconfiguration. This allows us to use the DIVA* approach where transient faults are detected by running an application or a program on two separate but identical processors/units. Detection of a difference between the two outputs will let us know if there has been a transient fault in any of the processing units, fault detection. This type of computation is well suited to *Piperench* which is a good source of computational tool when there is not a lot of communication needed between fabrics. Fault tolerance on a *Piperench* architecture would work by configuring two separate fabrics for the same program and letting the program run on the configured fabrics. This would obviously need a few adjustments to be able to detect differences in the outputs. But since we have a lot of devices around, we can just configure another device to serve as the difference detector.

If we continue this line of thinking, it might be possible to not only detect faults, but also correct them by using a redundancy approach like TMR where the majority gate can be configured with another fabric and three other similarly configured fabrics can act as the triple redundancy that is required. This would obviously require a moderate change to the current architecture, but the spirit still remains the same.

Another interesting point is that TMR and other redundancy correction methods require a large number of devices which even nanotechnology might not be able to provide. So a very far sighted, but feasible approach would be to use the idea of a virtual hardware through data-path pipelining that would get rid of hardware constraints that are introduced with these redundancy models.

In summary with our eye on nanotechnology, *Piperench* can be used as a guide to do fast reconfiguration for defect testing on the run as well as fault detection by using similar fabrics to run similar programs. An ambitious approach would use redundancy schemes to do fault correction using *Piperench* fabrics and then fight the hardware constraint of these redundancy schemes using hardware pipelining.

* Todd Austin, "[DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design](#)," ACM/IEEE 32nd Annual Symposium on Microarchitecture (MICRO-32), November 1999.