PipeRench: A Coprocessor for Streaming Multimedia Acceleration

Seth Goldstein, Herman Schmit et al.
Carnegie Mellon University

Introduction

- Multimedia workloads increasingly emphasize relatively simple calculations on massive quantities of mixed-width data
- Underutilizes the processing strengths of conventional processors in two important respects:
  1. Size of data elements underutilizes the processorís wide datapath
     - Multimedia extensions and SIMD instructions attempt to address this defficiency (see SIMD later)
  2. Instruction bandwidth is much higher than is needed for regular dataflow-dominated computations on large datasets
     - Renewed interest in vector processing (see IRAM later)

Reconfigurable Computing

- A fundamentally different way is to configure connections between programmable logic elements and registers to construct an efficient, highly parallel implementation of the processing kernel
- The interconnected network of processing elements is called a *reconfigurable fabric*
- The dataset used to program the interconnect and processing elements is a *configuration*
- The advantages of this approach, known as *reconfigurable computing* is that no further instruction download is required after a configuration is loaded and the right combination of simple processing elements can be combined to match the requirements of the computations

Reconfigurable Computing Challenges

- Picking the right *logic granularity*
- Living with *hard constraints*
- Minimising *configuration overheads*
- Finding *appropriate mappings*
- Excessive *compilation times*
- Providing *forward compatibility*

PipeRench solves some of these problems

- PipeRench uses a technique called *pipeline reconfiguration* to solve the problems of compilability, reconfiguration time, and forward compatability
- Architectural parameters, including logic block granularity, have been chosen to optimize performance for a suite of multimedia kernels
- PipeRench is claimed to balance the needs of the compiler against the design realities of deep sub-micron process technology

Attributes of Target Kernels

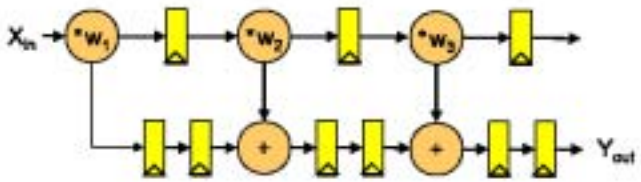
- Reconfigurable fabrics can provide significant benefits for functions with one or more of the following features:
  1. The function operates on bit-widths that differ from processorís native word size
  2. Data dependencies in the function allow multiple function units to operate in parallel
  3. The function is composed of a series of basic operations that can be combined into a single specialized operation
  4. The function can be pipelined
  5. Constant propagation can be performed, reducing the complexity of operations
  6. The input values are reused many times within the computation

Two broad categories emergeÖ

- *Stream-based functions* process a large data input stream and produce a large output stream
- *Custom instructions* take a few inputs and produce a few outputs

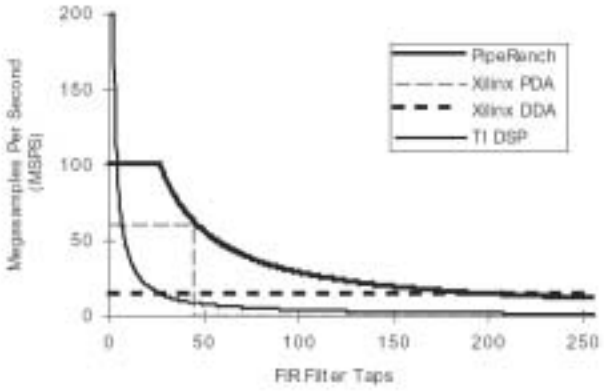
Stream-based example

```
for (int i=0; i<maxInput; i++) {
   y[i] = 0;
   for (int j=0; j<Taps; j++)
      y[i] = y[i] + x[i+j]*w[j];
}
```



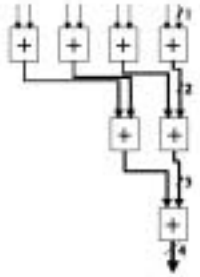
Code for a FIR filter and a pipelined version for a three-tap filter

Performance on 8-bit FIR filters



Custom instruction example

```
int
popCount(unsigned n) {
  int sum=0, i;
  for (i=0; i<8; i++)
    sum += (n >> i) & 1;
  return sum;
}
```

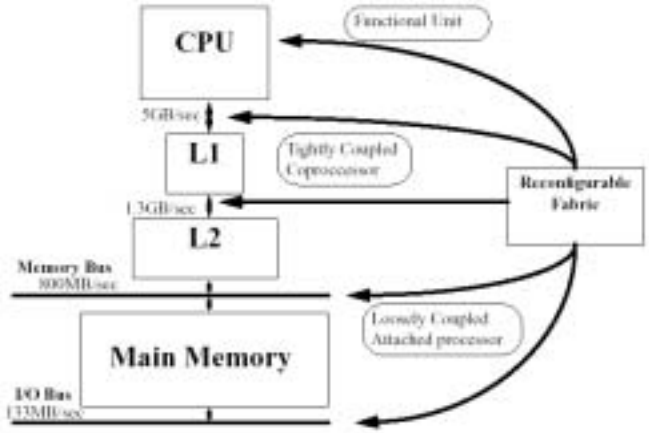


- The reconfigurable computing solution replaces the *O(n)* loop with an adder tree of height *O(log n)*

Configuration time & Communication latency

- If the previous *popCount* function is called just once it may not be worth configuring the fabric because the time needed to configure the function exceeds the benefit obtained from executing the function on the fabric
- If the function is used outside of a loop, and its results are to be used immediately, the fabric needs direct access to the processorís registers
- On the other hand if the function is used in a loop with no immediate dependencies on the results, performance can be improved by providing the fabric with direct access to memory

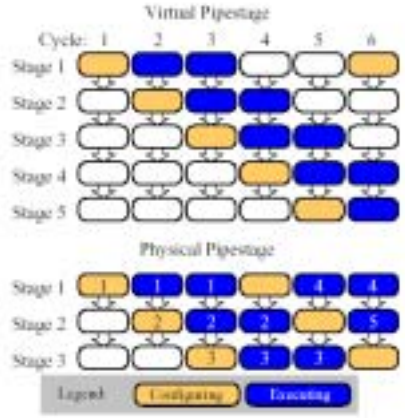
Possible placements for reconfigurable fabrics



Pipelined Reconfigurable Architectures

- We have seen how application-specific configurations can be used to accelerate applications
- The static nature of these configurations causes problems if
  1. The computation requires more hardware than is available, and
  2. The configuration doesnít exploit the additional resources that will ineviatbly become available in future process generations
- *Pipeline reconfiguration* allows a large logical design to be implemented on a small piece of hardware through rapid reconfiguration of that hardware

Pipeline Reconfiguration



- This diagram illustrates the process of virtualizing a five-stage pipeline on a three-stage device

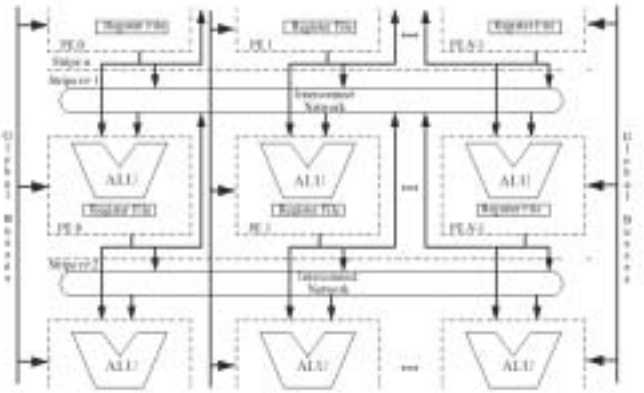
Benefits of pipeline reconfiguration

- Pipeline reconfiguration breaks the single static configuration into pieces that correspond to pipeline stages ñ these are then loaded, one per cycle, into the fabric. Computation proceeds even though the whole configuration is never present at one time
- With this technique, the compiler is no longer responsible for satisfying fixed hardware constraints
- In addition, the performance of the design improves in proportion to the amount of hardware allocated to that design; as future process technology makes more transistors available, the same hardware designs achieve higher levels of performance
- The configuration cost is hidden

Challenges of pipeline reconfiguration

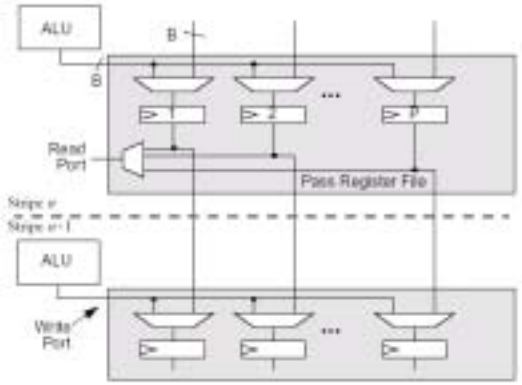
- For virtualization to work, cyclic dependencies must fit within one stage of the pipeline
  - Interconnections to previous or future stages other than the immediate successor are not allowed
  - Fortunately, this is not a severe restriction on multimedia computations, and the architecture provides pass registers to support forwarding
- The primary challenge is configuring a computationally significant pipeline stage in one cycle
  - Wide on-chip configuration buffers must be used
- Before swapping virtual stages, the state of the resident stage, if any, must be stored. This state needs to be restored when loading this stage once more

The PipeRench architectural class



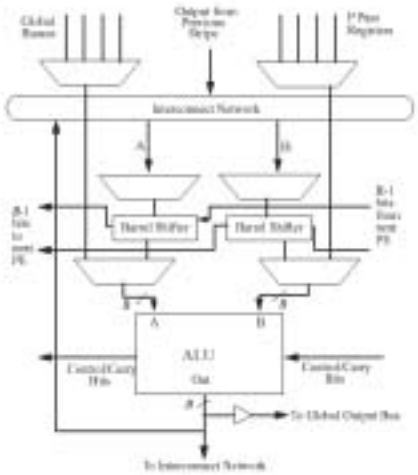
- ALUs are LUTs; PEs have access to global I/O bus; PEs can access operands from registered outputs of previous as well as current stage; no interconnect to previous stage

Pass Register File



- Provides efficient (registered) interstage connections; ALU output can write to any of *P* registers otherwise register is loaded from previous stage

Interconnection network

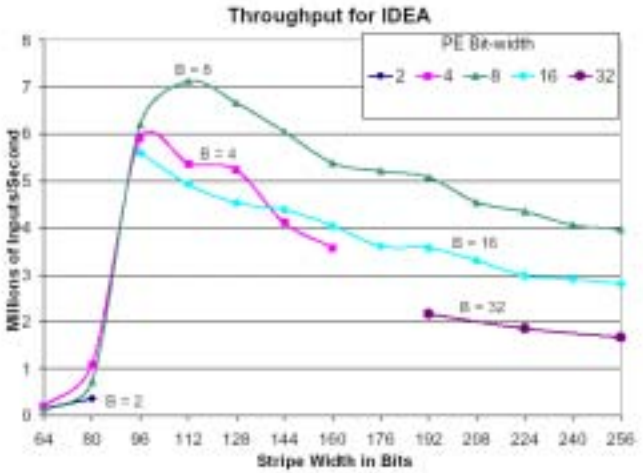


- Full *B*-bit *N*x*N* crossbar; barrel shifter for word-based arithmetic

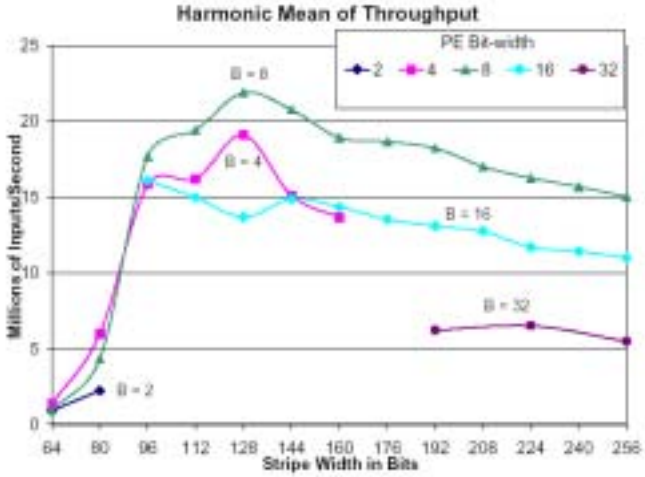
Evaluation

- Three architectural parameters:
  - *N* (number of PEs per stage)
  - *B* (bit-width of ALU and registers), and
  - *P* (number of pass registers)
- Evaluate performance as parameters varied using several kernels:
  - ATR
  - Cordic
  - DCT
  - FIR
  - IDEA
  - Nqueens
  - Over (Porter-Duff operator for joining two images based on a mask of transparency values for each pixel), and
  - popCount

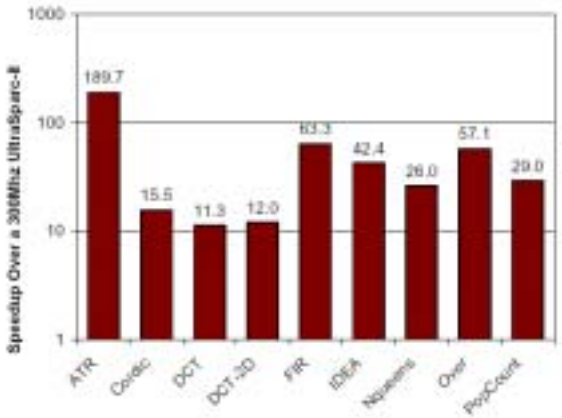
Representative results on up to 8 registers



Over all kernels



Speedup



- **Speedup for 8-bit Pes, 8 registers/PE, 128-bit wide stripes (stages)**