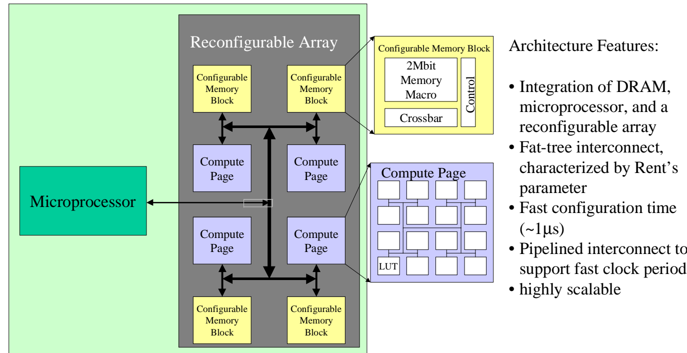
Scheduling on a Reconfigurable Processor with Virtual Compute Page

Eylon Caspi, Michael Chu, and Randy Huang
http://www.cs.berkeley.edu/~eylon/cs262/

Problem: **How to reduce programmer effort in managing large computations on arbitrary reconfigurable resources?**
Solution: **Virtual hardware pages with OS support**

What's a Reconfigurable Processor?



What's a virtual compute page?

A Compute Page (CP) is a hierarchically-connected array of programmable computational elements called LookUpTables (LUTs). E.g. A 4-LUT can compute any binary function of 4 inputs.

A Virtual Compute Page is a logical entity which must be scheduled to run in a physical compute page, analogous to virtual memory pages and physical frames. Operating system support automatically manages the virtual mapping and page loading/eviction. Automated virtualization frees application writers from physical size constraints and enables compatibility across different size arrays.

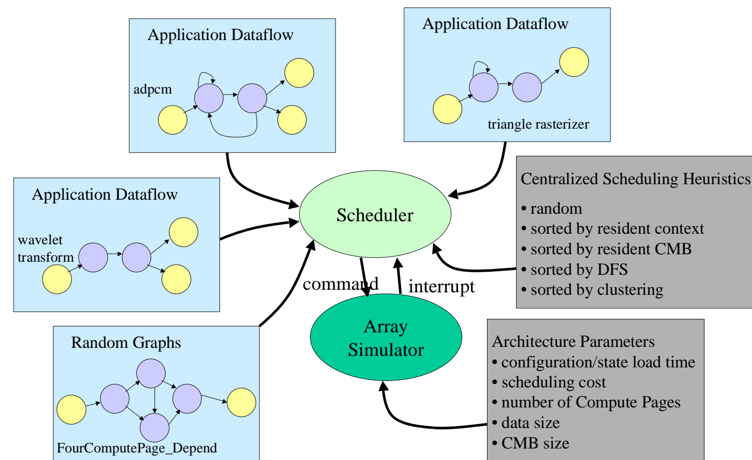
Methodology

1. Define preliminary details of Stream Computing On a Reconfigurable Environment (SCORE) programming model
2. Select and map applications in this programming model
3. Build an event-driven array simulator
4. Build a centralized scheduler
5. Try different scheduling heuristics/algorithms
6. Understand architectural tradeoffs

BRASS Home Page: http://www.cs.berkeley.edu/projects/brass/

How to Program Using SCORE

```
import score.*;

public class FourComputePage_Depend extends Design {

  public CMBSeqSrc src;
  public SingleComputePage_page_2outs buf0;
  public SingleComputePage_page_2outs buf1;
  public SingleComputePage_page_2ins buf2;
  public SingleComputePage_page_2ins buf3;
  public CMBSeqSink sink;

  public FourComputePage_Depend() {
    char memSrc[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 99};
    char memSink[] = new char[memSrc.length];

    src = new CMBSeqSrc(memSrc.length, memSrc, 8, 1);
    src.setName("src");
    buf0 = new SingleComputePage_page_2outs();
    buf0.setName("buf0");
    buf1 = new SingleComputePage_page_2outs();
    buf1.setName("buf1");
    buf2 = new SingleComputePage_page_2ins();
    buf2.setName("buf2");
    buf3 = new SingleComputePage_page_2ins();
    buf3.setName("buf3");
    sink = new CMBSeqSink(memSink.length, memSink, 8, 1);
    sink.setName("sink");

    connect(src.dataOut, buf0.input);
    connect(buf0.output0, buf1.input);
    connect(buf0.output1, buf2.input0);
    connect(buf1.output0, buf2.input1);
    connect(buf1.output1, buf3.input0);
    connect(buf2.output, buf3.input1);
    connect(buf3.output, sink.dataIn);
  }

  public static void main(String args[]) {
    FourComputePage_Depend design;
    Simulator simulator;
    Scheduler scheduler;

    simulator = new Simulator(16, 100000);
    scheduler = new Scheduler();
    design = new FourComputePage_Depend();

    scheduler.addDesign(design);

    System.err.println("SIMULATOR EXIT CODE: " + simulator.run());;

    System.err.println("Simulator time: " + simulator.getTime());
  }

}
```



Programming Model

- Token flow among compute-page operators

Scheduling Considerations

- I/O between resident and non-resident compute-pages is retimed in on-chip DRAM (configurable memory blocks, CMBs)
- On array, streaming communication is cheap, context swaps are expensive (opposite of MP)
- combine list-based resource-constrained scheduling and heuristic gang scheduling
- time-sliced operation

Scheduling Algorithm

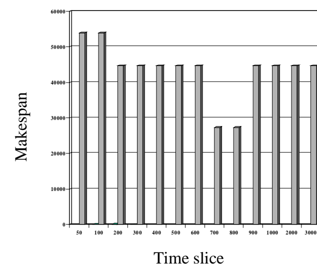
•*buildRQ*( ): determine which virtual compute pages are ready to be scheduled

•*buildPriorityRQ*( ): given readyQ and page priority heuristic, forms a *prioritizedReadyQ*

•*jobSelection*( ): given *prioritizedReadyQ* and cost model, selects which VPCs to evict/schedule on array (maximize locality, minimize page configuration+state swaps)

•*issueCommands*(): parallelizes and issues command groups to the simulator

Preliminary Results



Lessons Learned

- Random graph generator is a very useful for debugging
- Sorting the prioritizedReadyQ by resident CMB can lead to deadlock
- Path-based priority algorithm seems to work best
- Equal number of CMBs and CPs might not lead to optimal utilization of area

Status

- 5500 lines of Java code written (scheduler, simulator)
- Mapped three benchmark into SCORE programming model
- Still collecting data

Future Work

•Improve array cost model

•Memory management for CMBs

•Cluster pages (feedback)

•Cross-process Fairness (lottery scheduling)