

Review of:

Exploiting Operation Level Parallelism through Dynamically Reconfigurable Datapaths

- Paper by:
 - Zhining Huang and Sharad Malik (Princeton)
- Presented at:
 - Design Automation Conference (DAC)
 - June 2002
- Original Paper online at:
 - <http://campuscgi.princeton.edu/~zhuang/mescal/ppt/41.pdf>
- Additional material from:
 - DAC'02 slides
- Survey by:
 - Chris Neely

The Challenge

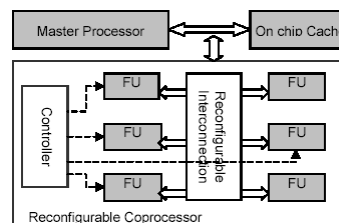
- ASICs are increasingly expensive for medium volume projects
 - ↑NRE costs
 - Signal Integrity issues
- “So-called” programmable solutions are:
 - FPGAs
 - Word-level ASIP
 - General purpose processors (in some cases)
- Authors introduce a dynamically-reconfigurable co-processor with fixed HW blocks and programmable interconnect
 - Present algorithm for designing co-processor
 - Method of mapping kernel loops to dynamic reconfigurable datapath
- Programmable logic has not been popular for multimedia accelerators, but perhaps multimedia can benefit through dynamic operation level parallelism (similar to VLIW).

Style of the Paper

- A Survey of existing work
 - Broad, covers fine to course grain dynamically reconfigurable computing projects with brief history and limitations
 - "Early FPGAs failed to become mainstream multimedia accelerators" (e.g. PipeRench)
 - Fine grained FPGA processors (e.g. GARP, RaPiD)
 - Large reconfiguration bit size
 - Slow clock speeds
 - Limited resources
 - Course grained FPGA (e.g. PipeRench, Pleiades)
 - Use reconfigurable computing logic blocks
 - Faster clock speeds
 - Ability to implement larger applications
 - Authors appear to like PipeRench, but it is still too fine grained
 - New programmable co-processors
 - Chameleon has 32 embedded processors with dynamically reconfigurable fabric
 - Present new work
 - Discuss their co-processor design and present their datapath algorithm for mapping kernel loops

Architecture and Methodology

- Master processor and co-processor have same access to memory.
- Co-processor has ASIC like function units (FU) and reconfigurable datapaths
- "For each such kernel loop a custom datapath is designed to maximize parallelism that can be exploited"
- Rapid context switching for small loops, where entire context can be stored locally.



Mapping Kernel Loops to Datapath

- IMPACT compiler does loop detection
 - Used a meta-assembly as intermediate representation
- Extract the loop details:
 - Live-in set
 - Live-out set
 - Data dependences within iterations
 - Data dependences between iterations

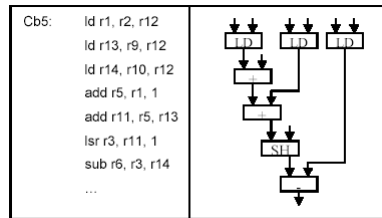


Figure 2. Direct mapping from IR to hardware

Mapping Kernel Loops to Datapath (cont.)

Example execution steps:

1. Application runs on the master processor.
2. Application hits loop.
3. Control switches execution to co-processor.
4. Live-in sets are copied to internal registers.
5. Co-processor runs loop (operations run in parallel—if possible)
6. Loop finishes. Live-out sets are written from registers back to memory.
7. Control switches to master processor.

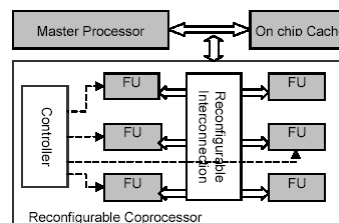


Figure 1. Architectural Model

Mapping Kernel Loops to Datapath (cont.)

- Use a one-to-one mapping between software instructions and hardware (function unit) blocks
- Form data path to connect blocks (as illustrated in Figure 2).

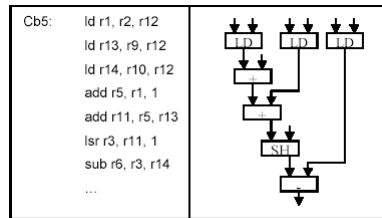


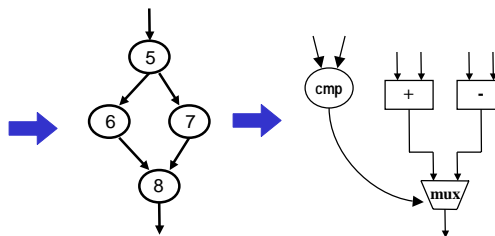
Figure 2. Direct mapping from IR to hardware

Direct Mapping (contd.)

- Branch condition transforms

```

Cb5: ...
    blt r6, 0, cb7
Cb6: add r19, r19, r6
    jump Cb8
Cb7: sub r19, r19, r6
Cb8: ...
    
```



DAC'02 Slide, Huang and Malik

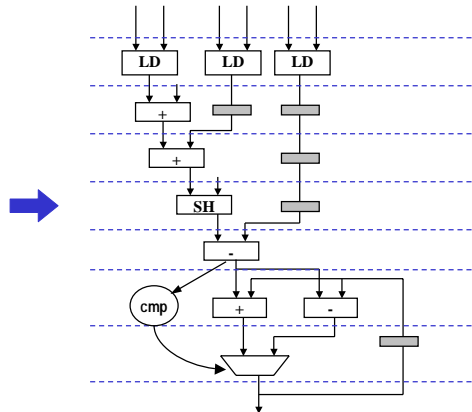
Intra-iteration Scheduling

- Schedule FUs into different pipe stages

```

Cb5: ld r1, r2, r12
     ld r13, r9, r12
     ld r14, r10, r12
     add r5, r1, 1
     add r11, r5, r13
     lsr r3, r11, 1
     sub r6, r3, r14
     blt r6, 0, cb7
Cb6: add r19, r19, r6
     jump Cb8
Cb7: sub r19, r19, r6
Cb8: ...
    
```

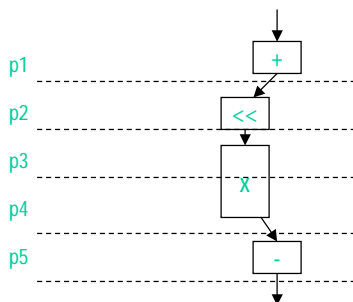
Kernel loop code from GSM



DAC'02 Slide, Huang and Malik

Inter-iteration Scheduling

- Pipelining the execution of loop iterations
- Determine the Initial Interval (II) of a loop datapath

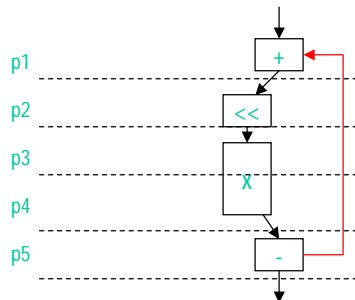


- if no data dependence
 - II = 1 (single copy datapath)
 - II = 0 (multiple copies of datapaths)

DAC'02 Slide, Huang and Malik

Inter-iteration Scheduling (contd.)

- Data dependence from FU i to FU j across loop iterations
 - Feedback connection
 - $II = \text{PipeStage}(i) - \text{PipeStage}(j) + \text{FU_Delay}(j)$, if $II > 0$

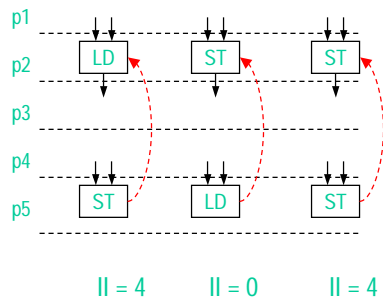


- $II = 5 - 1 + 1 = 5$
- Fetch new loop iteration every 5 cycles

DAC'02 Slide, Huang and Malik

Inter-iteration Scheduling (contd.)

- Data dependence on memory access
 - No feedback connections needed
 - $II = \lceil [\text{PipeStage}(i) - \text{PipeStage}(j) + 1] / k \rceil$
 - K : distance of dependent iterations, from data dependence analysis

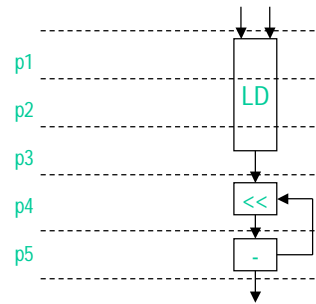


DAC'02 Slide, Huang and Malik

Execution Time Estimation

$$T = [S + II \times (N-1)] + O + W \text{ (cycles)}$$

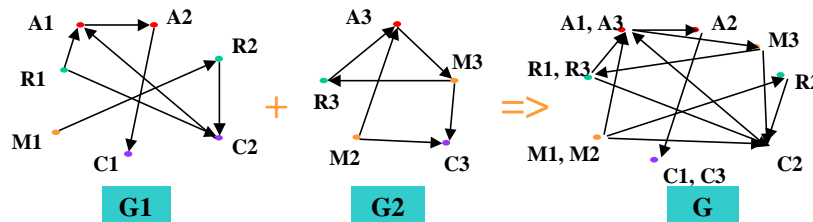
- S: total # of pipeline stages of the datapath
- II: initial interval between the fetch of 2 consecutive iterations
- N: loop iteration number
- O: configuration overhead
- W: system write back
- Example: $T = 5 + 2 \times (32-1) + 4 = 71$



DAC'02 Slide, Huang and Malik

Reconfigurable Datapath Design

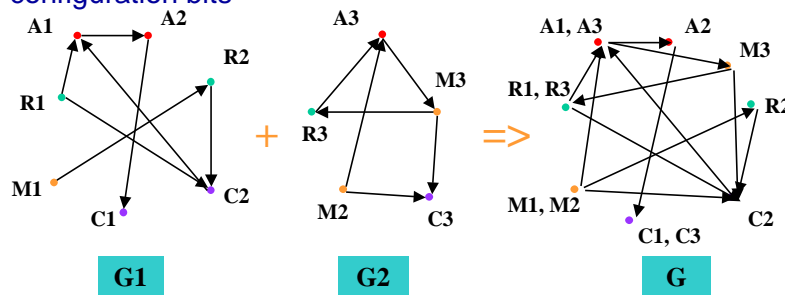
- Embed individual datapaths into a single datapath.
 - Datapath graph G_i
 - Vertices are hardware resources (memories, registers, function units)
 - Edges are connections between them
 - Construct a single graph G such that each $G_i \subseteq G$ and G has the fewest edges and vertices
 - Bipartite matching based algorithm [Huang+ 2001]



DAC'02 Slide, Huang and Malik

Reconfigurable Datapath

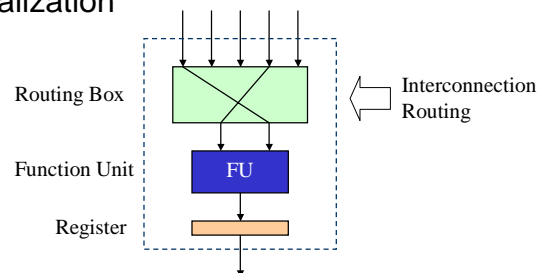
- Merged graph G to reconfigurable datapath
 - Vertices to function units
 - Edges to reconfigurable interconnects
- By selecting subset of interconnections, any selected datapath can be generated and executed on reconfigurable datapath
- Appropriate interconnects in merged datapath are enabled using configuration bits



DAC'02 Slide, Huang and Malik

FU Routing Boxes

- Useful interconnections are selected
- Routing box to select between multiple connections
- Configuration contexts
 - Configuration bits for routing box
 - Control bits for some FU
 - Static registers initialization
- Delay is one gate.



Critical Path and Clock Speed

- Critical interconnect path determines clock speed of co-processor.
- FUs have variable delay depending on complexity of the instruction
- This section doesn't provide much useful information.
- They mention that the co-processor will probably use a slower clock than master processor, but they do not discuss synchronization.

Reconfiguration Overhead

- Usually only 8-16 loops (total?) are selected from kernel and then contexts are stored in distributed cache of the co-processor.
- Context switches involve swapping the live-out and live-in sets (typically 10-20 registers).
- The register file is the bottleneck, with 4 register read/write ports this adds 5 cycles of overhead.

MPEG2 Encoder Calculated Performance

Table 1. Top 10 kernel loops from MPEG

	Software Time	Hardware Time	# of Memory Ports	Speed Up
Loop 1	820	132.3	2	6.2
Loop 2	85.0	9.57	5	8.88
Loop 3	33.5	4.79	3	6.99
Loop 4	33.1	4.73	3	7.0
Loop 5	35.7	10.8	2	3.31
Loop 6	20.2	0.64	3	31.6
Loop 7	31.4	9.7	2	3.24
Loop 8	11.5	1.82	2	6.32
Loop 9	10.6	1.68	2	6.31
Loop 10	7.3	0.50	2	14.6
Rest	178.2	-	-	1
Overall	1266.3	354.73	-	3.57

(Number in Million cycles)

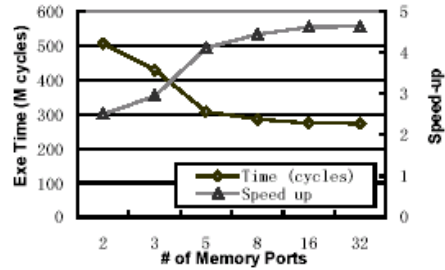


Figure 4. Speed-up vs. Memory Bandwidth for MPEG

(Reconfiguration context is 500 bits per loop.)

They assume speed of co-processor is running at the same speed as master processor (however, they just stated that this is unlikely)—not mentioned until conclusion.

MPEG2 Encoder Calculated Performance

- Performance Comparison based on Number of Instructions issued in parallel

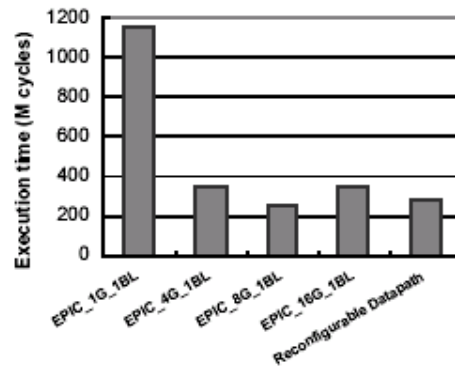


Figure 5. VLIW vs. Reconfigurable Datapath for MPEG

GSM Compared Performance to MPEG2

Table 2. Top 10 kernel loops from GSM

	Software Time	Hardware Time	# of Memory Ports	Speed Up
Loop 1	61.0	9.47	3	6.44
Loop 2	33.3	13.8	2	2.41
Loop 3	5.76	0.203	10	28.4
Loop 4	4.21	0.151	4	27.9
Loop 5	2.31	0.157	1	14.7
Loop 6	2.07	0.192	1	10.8
Loop 7	2.07	0.184	3	11.2
Loop 8	2.06	0.149	2	13.8
Loop 9	1.40	0.118	2	11.9
Loop	1.26	0.149	2	8.46
Rest	26.45	-	-	1
Overall	141.9	51.0	-	2.78

(Number in million cycles)

GSM

Table 1. Top 10 kernel loops from MPEG

	Software Time	Hardware Time	# of Memory Ports	Speed Up
Loop 1	820	132.3	2	6.2
Loop 2	85.0	9.57	5	8.88
Loop 3	33.5	4.79	3	6.99
Loop 4	33.1	4.73	3	7.0
Loop 5	35.7	10.8	2	3.31
Loop 6	20.2	0.64	3	31.6
Loop 7	31.4	9.7	2	3.24
Loop 8	11.5	1.82	2	6.32
Loop 9	10.6	1.68	2	6.31
Loop 10	7.3	0.50	2	14.6
Rest	178.2	-	-	1
Overall	1266.3	354.73	-	3.57

(Number in Million cycles)

MPEG2

Conclusions

- Presented a methodology for the design of dynamically reconfigurable datapath
- Co-processor does similar work as VLIW
 - VLIW is another technology that exploits operation level parallelism
 - VLIW is compiler intensive, whereas the method discussed in this paper is straightforward
 - They predict that their co-processor could more cost effective
- Tensilica has related technology that could be used to exploit localized sections of operation level concurrency

Questions and Discussion

- What are some of the things that were done well in this paper?
- How could this paper have been better?
- Was it clear how detailed their analysis was?