

Early Performance-Cost Estimation of Application-Specific Data Path Pipelining

Jelena Trajkovic

Computer Science Department
École Polytechnique de Montréal,
Canada

Email: jelena.trajkovic@polymtl.ca

Daniel D. Gajski

Center for Embedded Computer
Systems
University of California, Irvine, USA

Email: gajski@uci.edu

Introduction

- Application-specific processors (ASPs) are used for optimized implementation of embedded systems
- Problem: determine the optimal pipeline configuration for a data path
 - Automatically estimating the application execution time
 - ⇒ Estimated clock cycle length * estimated number of cycles
- Compute the cost of each pipelined design
- Our estimation enables **fast, accurate and early analysis** of different pipeline configurations
 - Each design is characterized by performance and cost
 - No need to create prototype or execute cycle-accurate model for each design
- High fidelity of performance estimation

Related Work

- Application Specific Instruction-set (IS) Processor and IS extension processors (Xtensa and Stretch processor)
 - Designer configures processor, including its pipeline
 - Based on SW profiling, not pipeline-specific estimation

- C-to-RTL tools (Catapult Synthesis)
 - Create data path and insert pipeline stages ‘on the fly’
 - Limited to small code size

- Our technique
 - Automatically produces performance and cost for early trade-off analysis
 - Can handle any size of C code (10K lines of C code)
 - Complementary to use of custom IS extensions and HW accelerators

Estimating the Number of Pipeline Stages

- Pipelining: partially overlap execution of multiple instructions
 - Increases throughput, not instruction execution time
 - Reduces clock cycle length
 - Ideal case: *speedup* factor = # pipeline stages

- Practical constraints:
 1. Can not divide a data path into equal parts
 2. Pipeline registers are required \Rightarrow overhead to cycle time (T_{clk})
 3. Dependences between adjacent instructions prevent partial overlapping

- #1 and #2 \rightarrow Impact on clock cycle time (T_{clk})
- #3 \rightarrow Influence the number of execution cycles

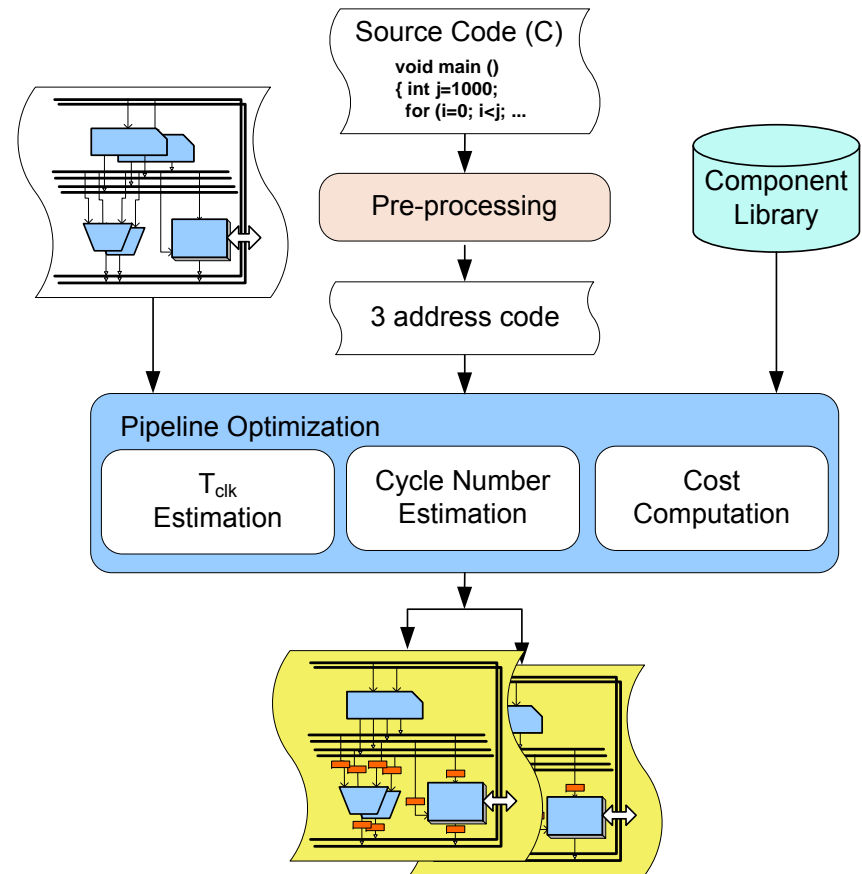
Pipelining - Tool Flow

Inputs:

- (Optimized) non-pipelined data path
- Application code
- Component Library
 - Xilinx, but may be any other

Estimate:

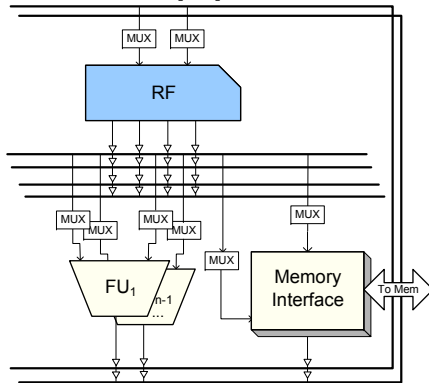
- Execution time T_{exe}
 - Cycle length T_{clk}
 - Number of cycles
- Cost
 - BRAM + Slices



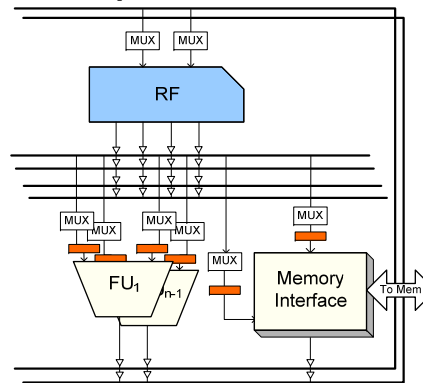
Candidate Pipeline Configurations

Uniform pipelining

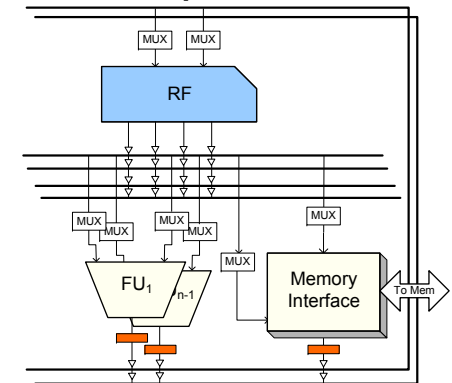
Non-pipeline



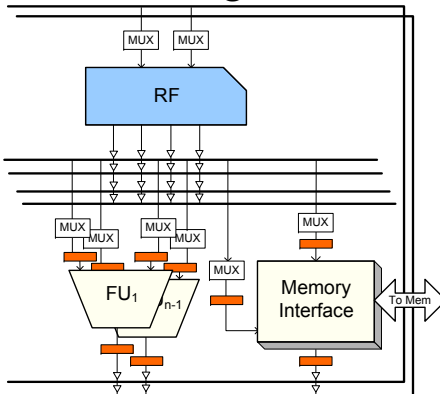
Input



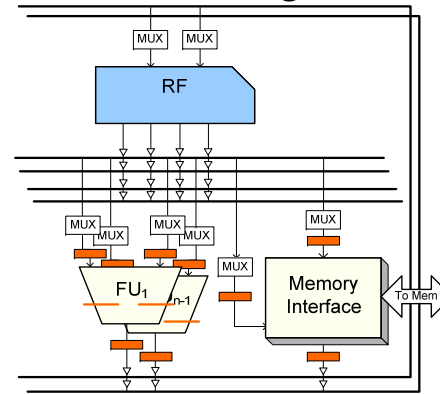
Output



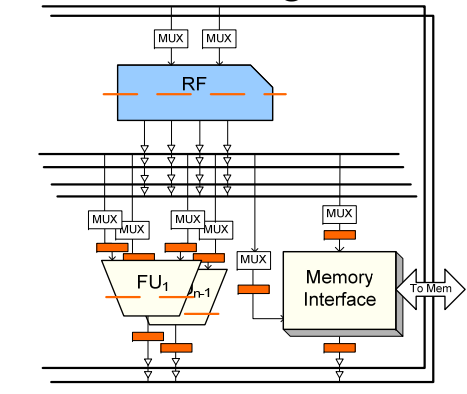
Two stage



Three stage



Four stage



Cycle Time T_{clk} Estimation

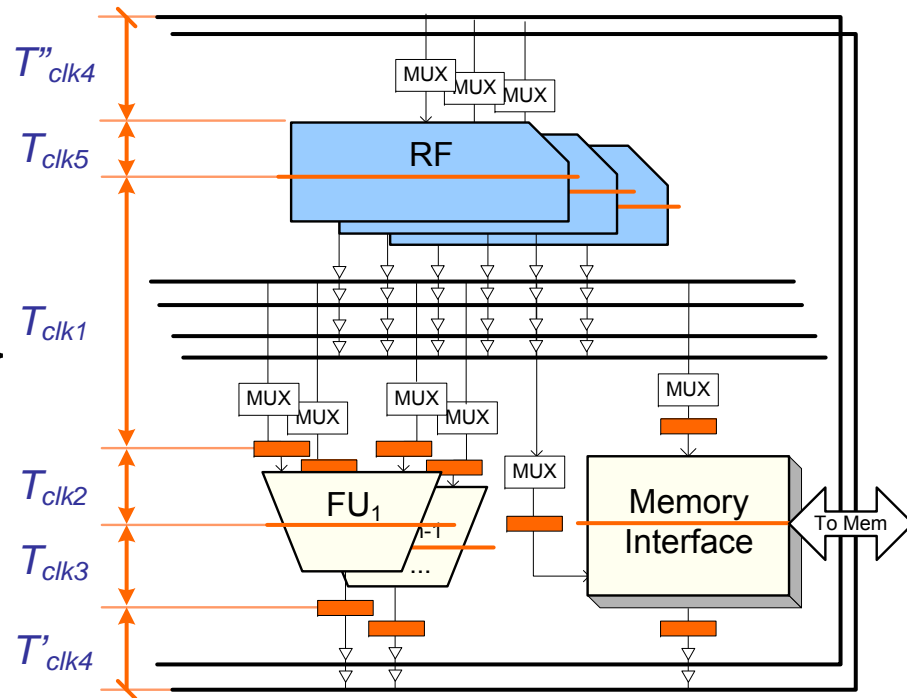
- Component is annotated with
 - Propagation delay
 - Setup time

- Eg: four stage pipelining

$$T_{clk} = \max \{ T_{clk1}, T_{clk2}, T_{clk3}, T_{clk4}, T_{clk5} \}$$

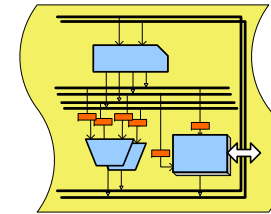
$$T_{clk1} = \max (\text{RF.prop delay}) + \max (\text{interconnect.src})$$

$$+ \max (\text{pipe reg}(T1). \text{setup time})$$



Number of Cycles Estimation

- Estimating code schedule
- Example:
 - Data path with Input pipelining
 - Memory access – 2 cycles
 - Add – 1 cycle



3-address code

```
state 0:  
t483 = *(t482);  
t77 = (t467 * t499);  
state 1:  
t79 = (t77 + t483);  
state 2:  
...
```

- Memory access time \Rightarrow *stall* cycle

```
cycle 0:  
addr_reg = [t482];  
pipe_reg_1 = [t467];  
pipe_reg_2 = t499;
```

```
cycle 1:  
t483 = Mem(addr_reg);  
t77 = pipe_reg_1 + pipe_reg_2;
```

```
cycle 2:  
stall
```

```
cycle 3:  
pipe_reg_1 = [t77];  
pipe_reg_2 = [t483];
```

```
cycle 4:  
t79 = pipe_reg_1 + pipe_reg_2;
```

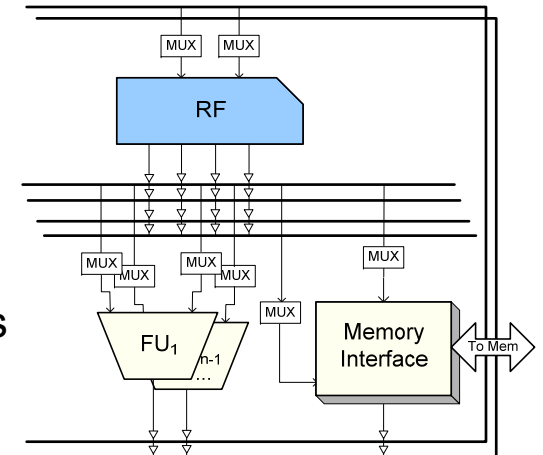
```
cycle 5: ...
```

- \Rightarrow Estimated number of **extra** cycles compared to the non-pipelined

Experimental Results

Application	LoC	Description
Sort	33	Bubble sort algorithm implementation
Bdist2	61	Part of motion estimation function from MPEG2 encoder
inv/forw 4x4	95	Integer transformation functions from video H.264 codecs
dct32	1006	DCT transformation implementation from Mp3
Mp3	13898	Reference Mp3 <i>decoder</i> implementation

- Mp3 – initial data path:
 - register file: 3 input/6 output ports, with 128 registers
 - 2 ALUs, 1 multiplier, 1 comparator and 1 divider
- Target architecture: NISC
 - No Instructions \Rightarrow no decoding step \Rightarrow compiler creates set of control signals (control words)
 - Our technique is applicable to any ASP design



Experimental Results – Mp3

Estimation error:

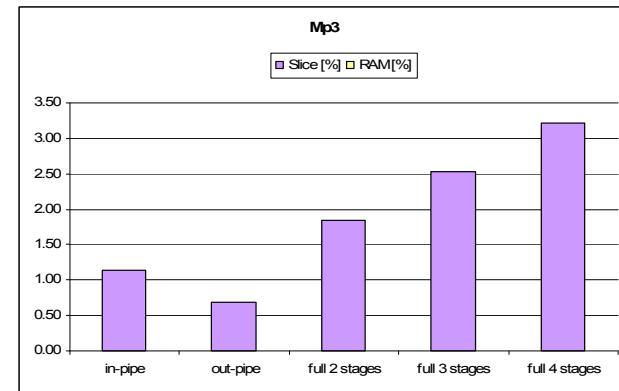
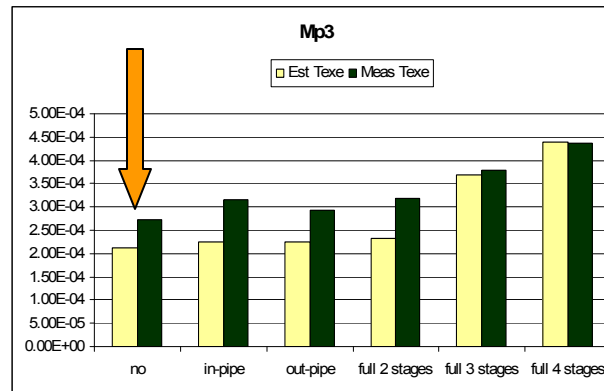
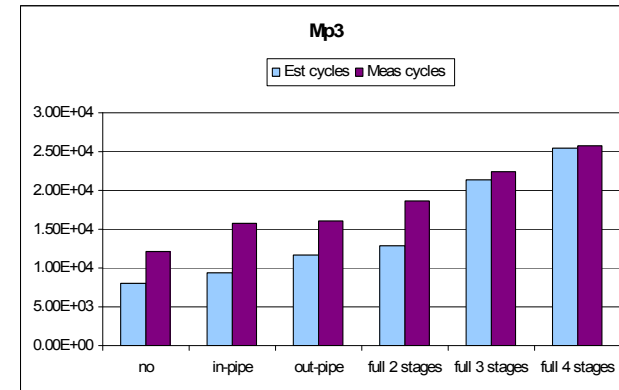
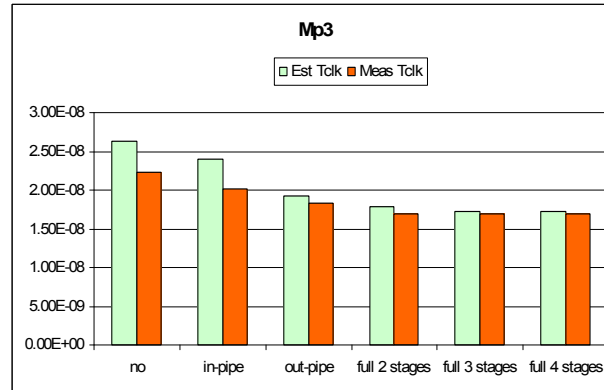
- Avg: 20%
- Max: 33%

Fidelity

- 94%
(71 out of 75)
- 5 benchmarks, 6 configs each

Evaluation metric: Fidelity

- Estimated (c_A) > Estimated (c_B) \Rightarrow Measured (c_A) > Measured (c_B)



Conclusions:

- Contributions:
 - Developed automatic method for estimating application execution time and cost for different data path configurations from C code
 - Implemented tool for automated early performance/cost estimation

- Benefits
 - Scalable to any size of C code
 - Evaluates pipeline configurations in fraction of time compared to manual design, simulation and synthesis
 - Significantly reduces the # of configurations to be tested

- Future work
 - Support for forwarding and non-uniform pipeline designs

Acknowledgements

- Prof. Gabriela Nicolescu (École Polytechnique de Montréal)
 - For constructive feedback on this paper

- Center for Embedded Computer Systems
 - For supporting this research project

Thank You

□ Questions