NANYANG TECHNOLOGICAL UNIVERSITY

On Data Forwarding in Deeply Pipelined Soft Processors

H. Y. Cheah, S. A. Fahmy, N. Kapre

School of Computer Engineering
Nanyang Technological University

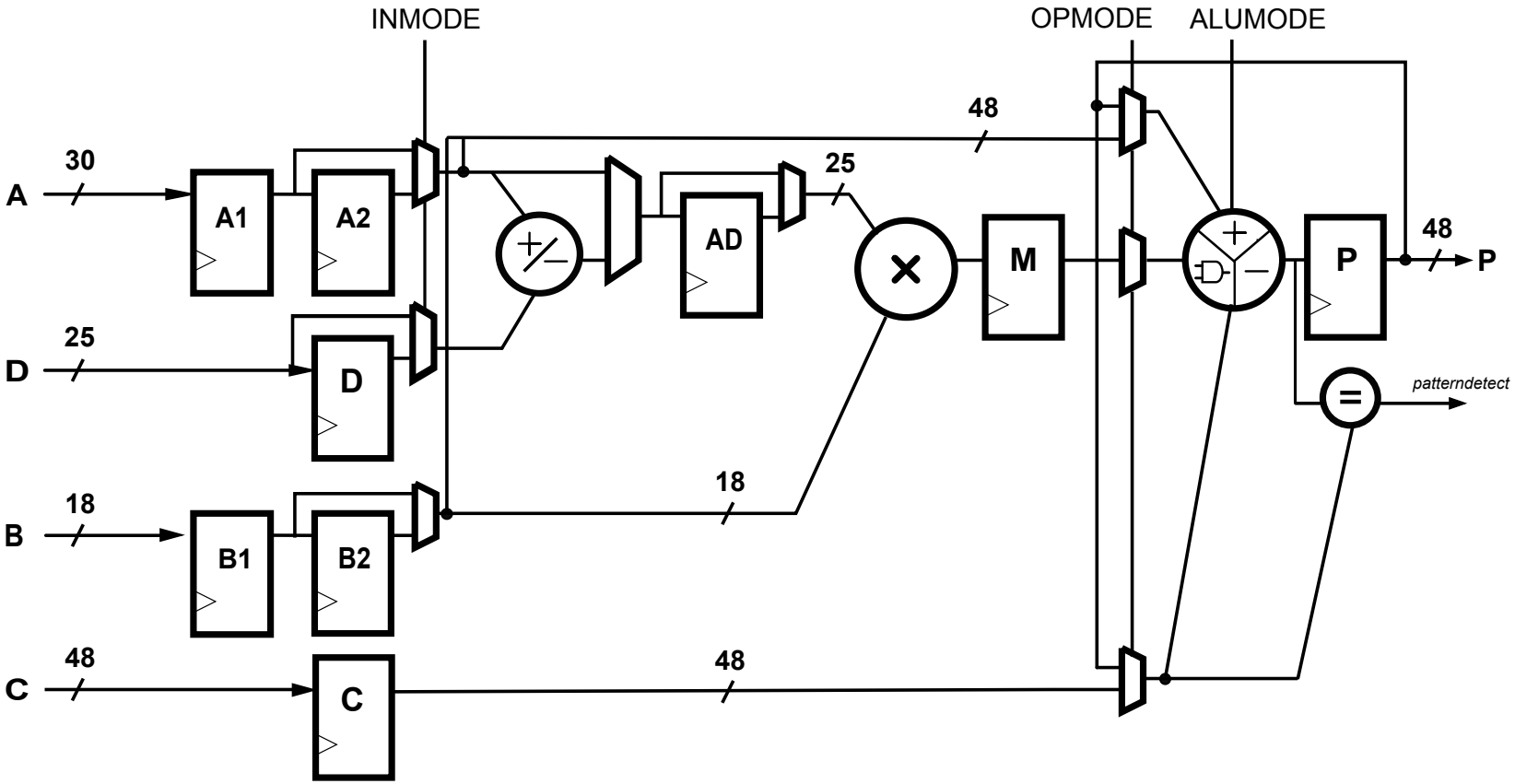
FPGA 2015

Introduction

- Hard macros allow us to build high frequency soft processors (Octavo, iDEA, etc.)
- To achieve high frequency, deep pipelining is required
- Results in significant idle cycles to resolve dependencies
- Full data forwarding is too complex for a lean processor
- Exploit loopback path in DSP block to provide forwarding in iDEA soft processor

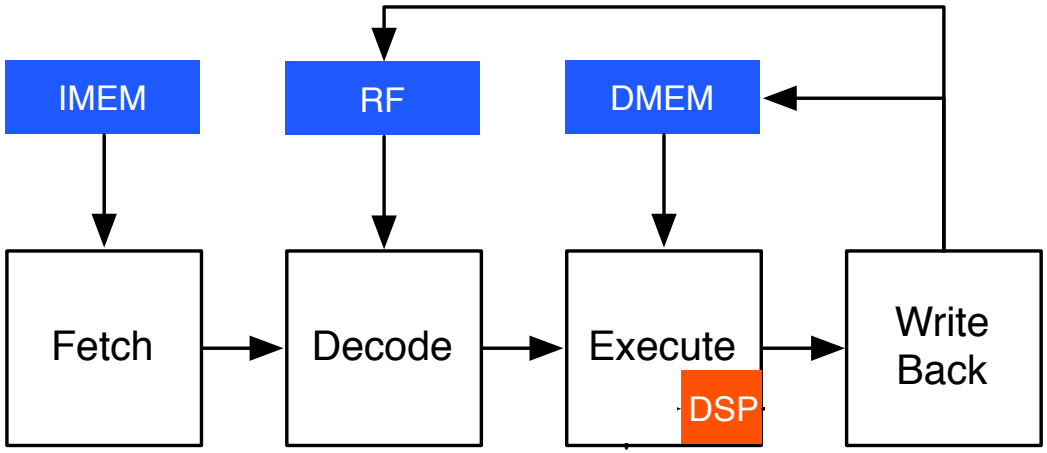
iDEA Soft Processor

- Fundamental novelty is exploiting DSP block dynamic control signals



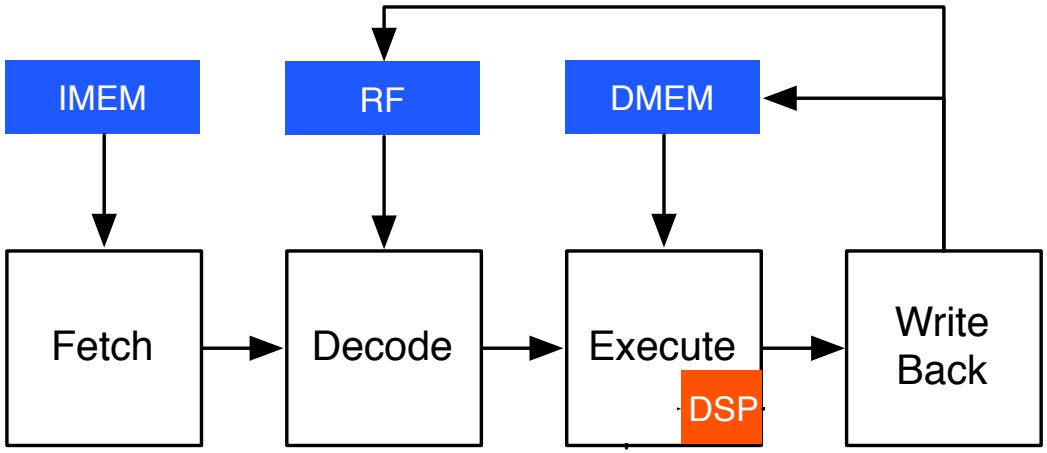
iDEA Soft Processor

- A DSP-block based soft processor
  - Maximises use of the DSP48E1 block
  - 450MHz+ achievable frequency
  - Architecture described in detail in FPT 2012 and TRETS 2014



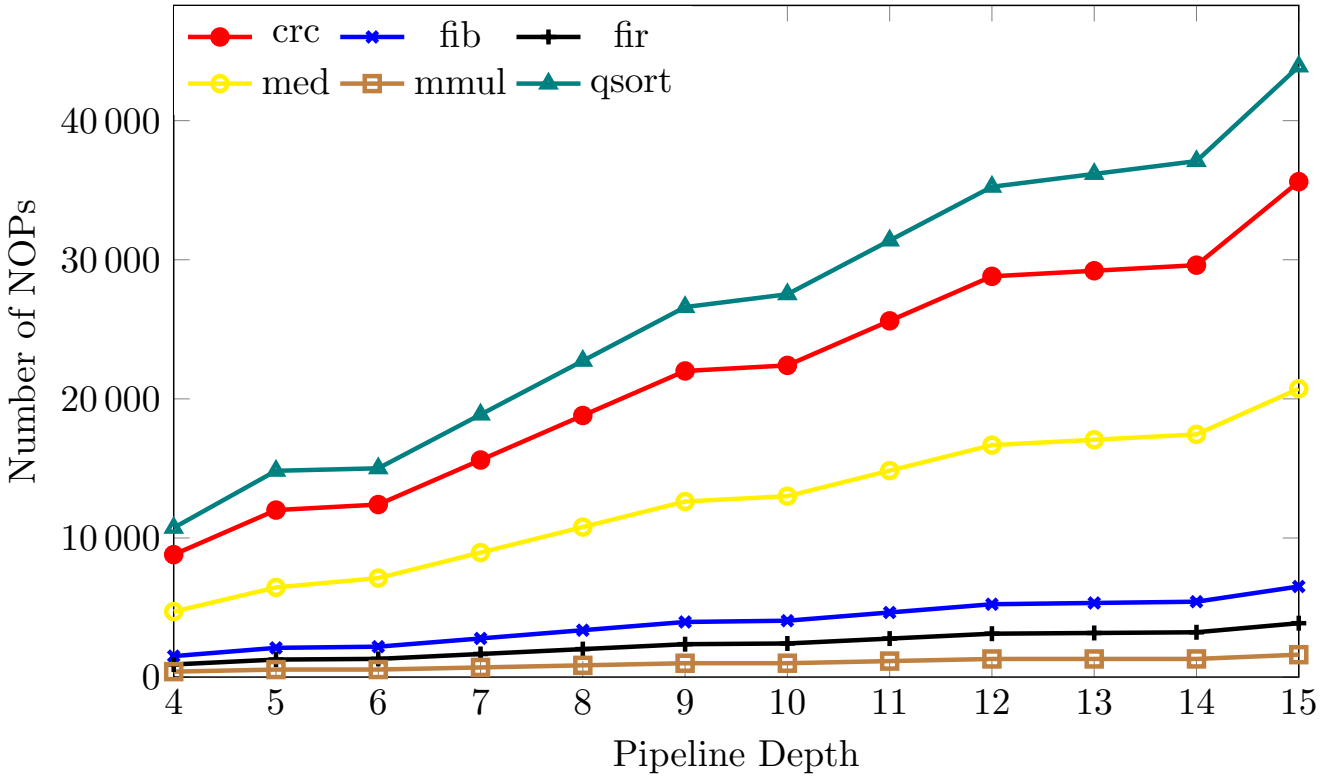
iDEA Soft Processor

- We deeply pipeline iDEA by adding extra cycles to the different processor stages
- DSP48E1 primitive requires 3 cycles to achieve maximum speed
- Multiple ways of distributing extra cycles among stages



Deep Pipelining and NOPs

- High number of empty cycles required to pad dependent instructions, increasing with depth

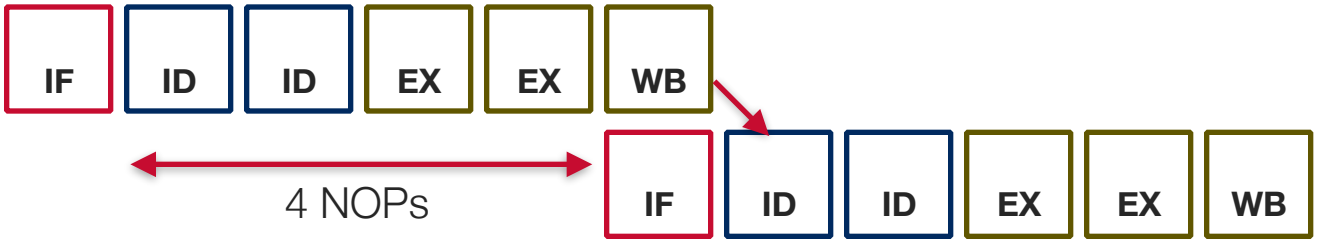


Padding for Dependencies

- Instruction cannot enter decode stage until its input operand has been written back
- Insert empty instructions (NOPs) between dependent instructions

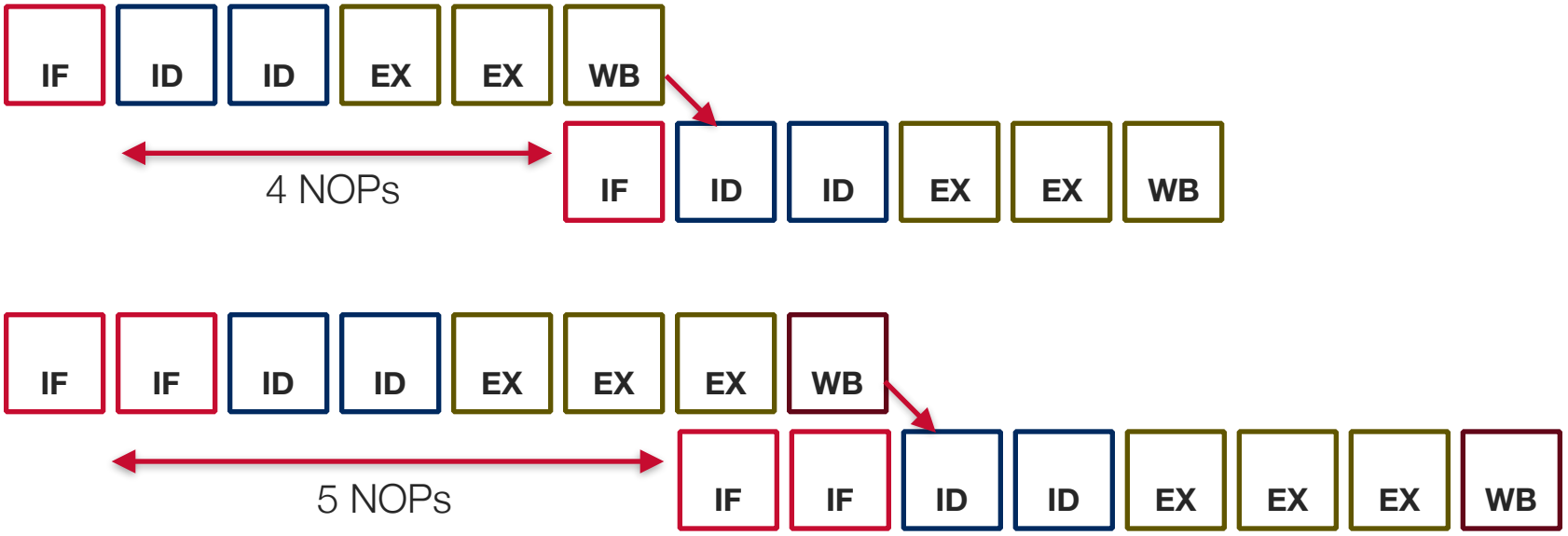
Padding for Dependencies

- Instruction cannot enter decode stage until its input operand has been written back
- Insert empty instructions (NOPs) between dependent instructions



Padding for Dependencies

- Instruction cannot enter decode stage until its input operand has been written back
- Insert empty instructions (NOPs) between dependent instructions



Padding for Dependencies

- A few routes to overcoming this issue:
  - Dynamic in hardware – cost
  - Static in software
    - External Loopback – add a forwarding path around the execute stage
    - Internal Loopback – proposed approach using DSP block feature
    - Hardware support through modified opcodes

Assembly with Loopback

- Original Assembly

```
li $1, x
li $2, a
li $3, b
li $4, c
mul $5, $1, $2
nop
nop
mul $6, $5, $1
mul $5, $1, $3
nop
nop
add $7, $5, $6
nop
nop
add $8, $7, $4
nop
nop
sw $8, 0($y)
```

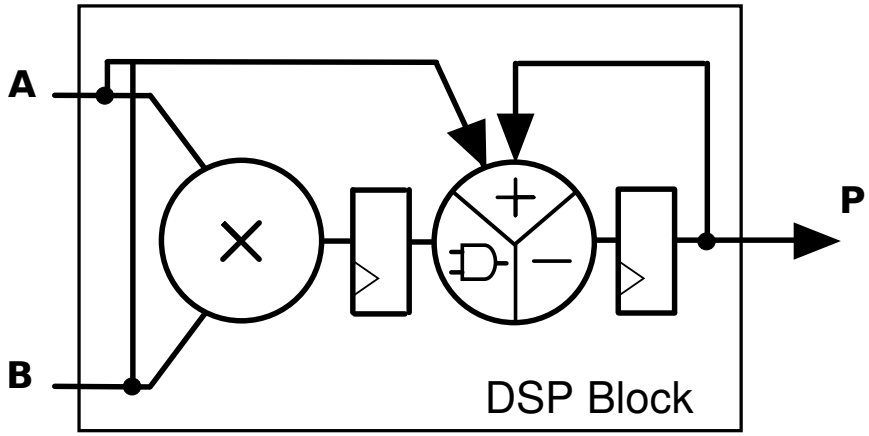
- With loopback instructions

```
li $1, x
li $2, a
li $3, b
li $4, c
mul loop0, $1, $2
mul $6, loop0, $1
mul loop1, $1, $3
add loop2, loop1, $6
add $8, loop2, $4
nop
nop
sw $8, 0($y)
```

→ save 6 instructions

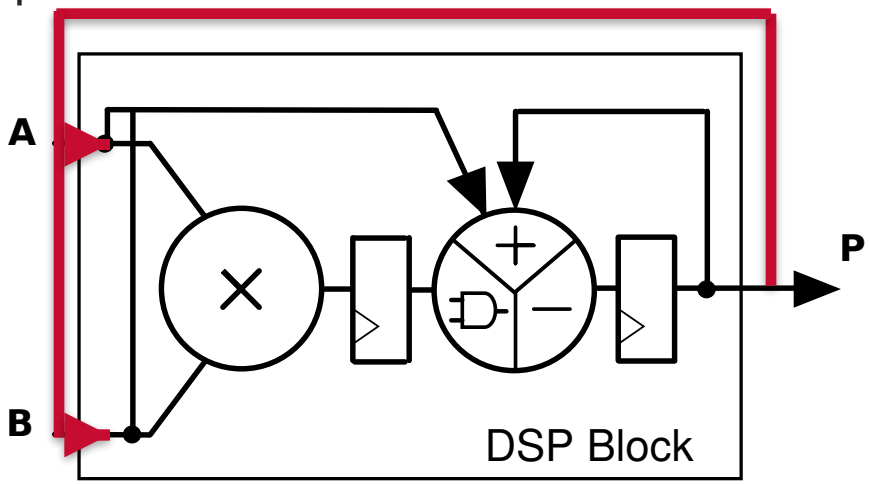
External Forwarding

- Add a path around the execute stage
  - Allows a dependent instruction to start its execute stage once the previous one has completed



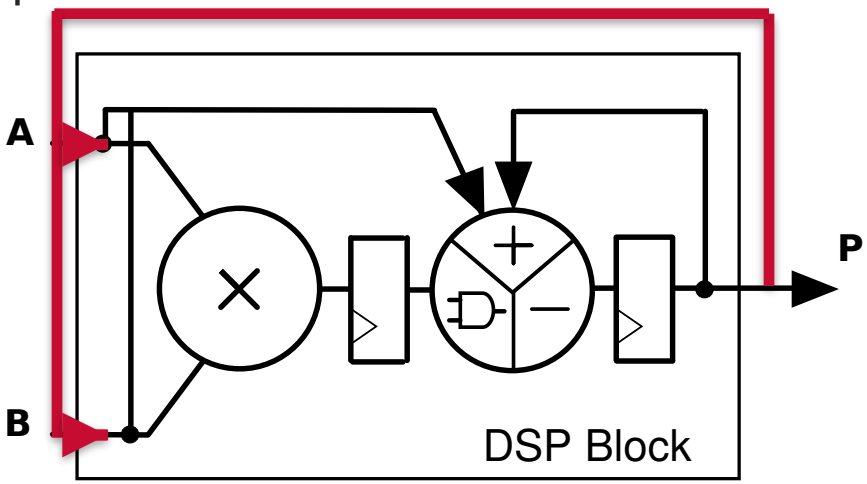
External Forwarding

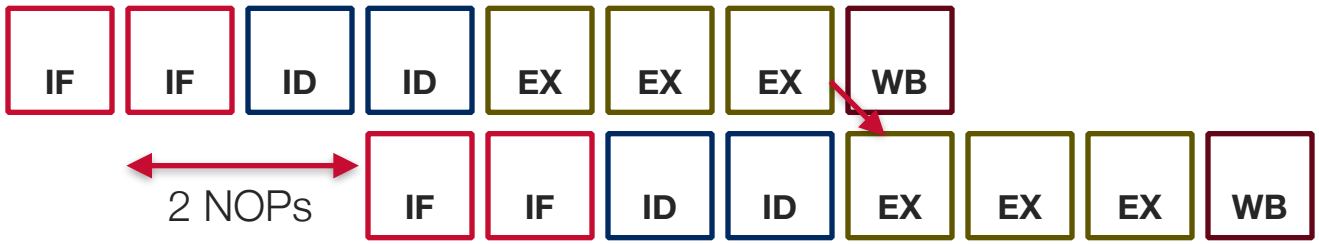
- Add a path around the execute stage
  - Allows a dependent instruction to start its execute stage once the previous one has completed



External Forwarding

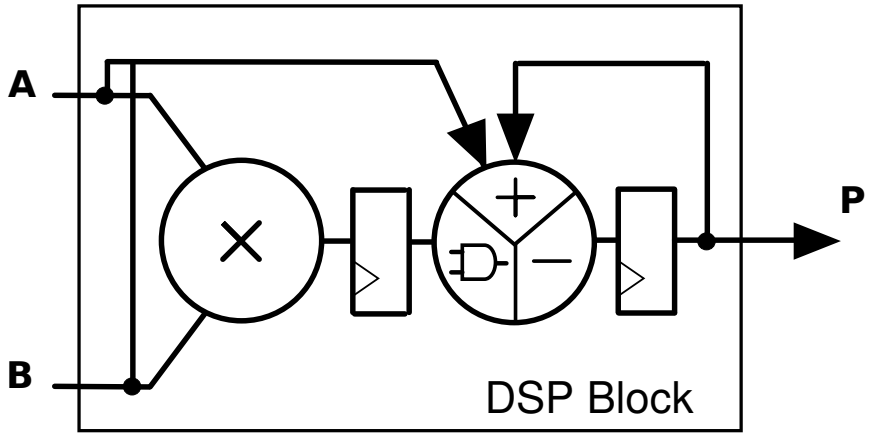
- Add a path around the execute stage
  - Allows a dependent instruction to start its execute stage once the previous one has completed





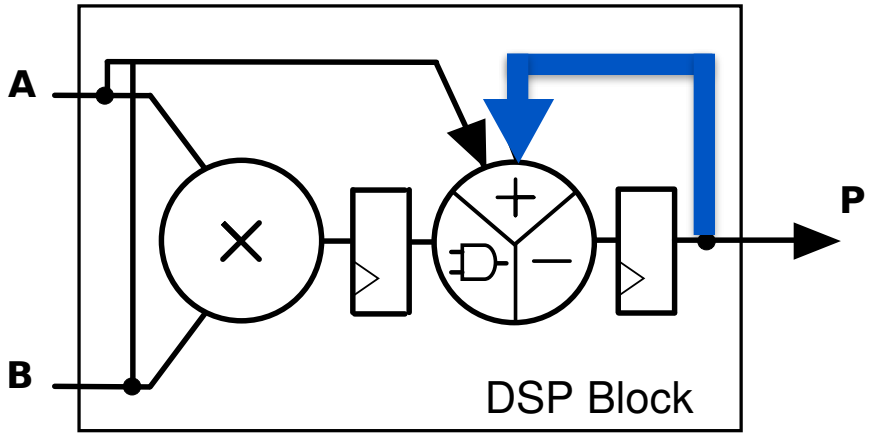
Internal Forwarding

- DSP block primarily used in filtering
- Multiply-accumulate is functional primitive
- Loopback path allows ALU output to be used for accumulation
- Dynamic control signal determines whether this path is used



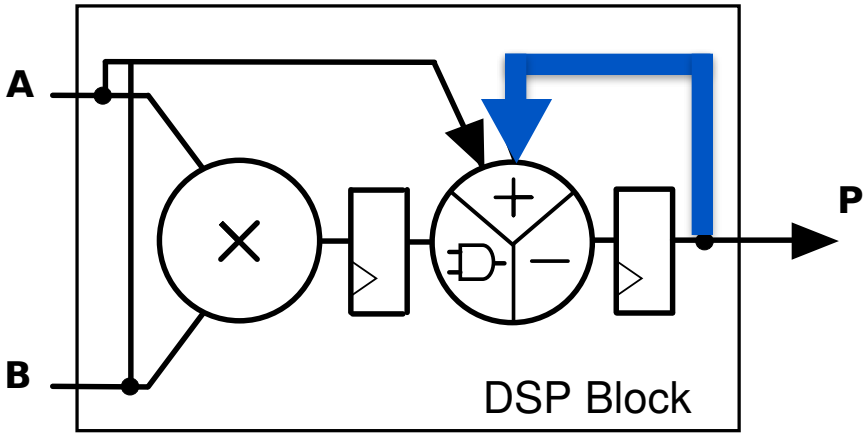
Internal Forwarding

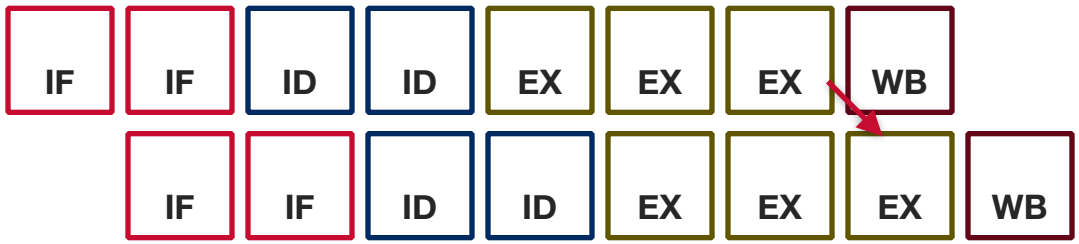
- DSP block primarily used in filtering
- Multiply-accumulate is functional primitive
- Loopback path allows ALU output to be used for accumulation
- Dynamic control signal determines whether this path is used



Internal Forwarding

- DSP block primarily used in filtering
- Multiply-accumulate is functional primitive
- Loopback path allows ALU output to be used for accumulation
- Dynamic control signal determines whether this path is used

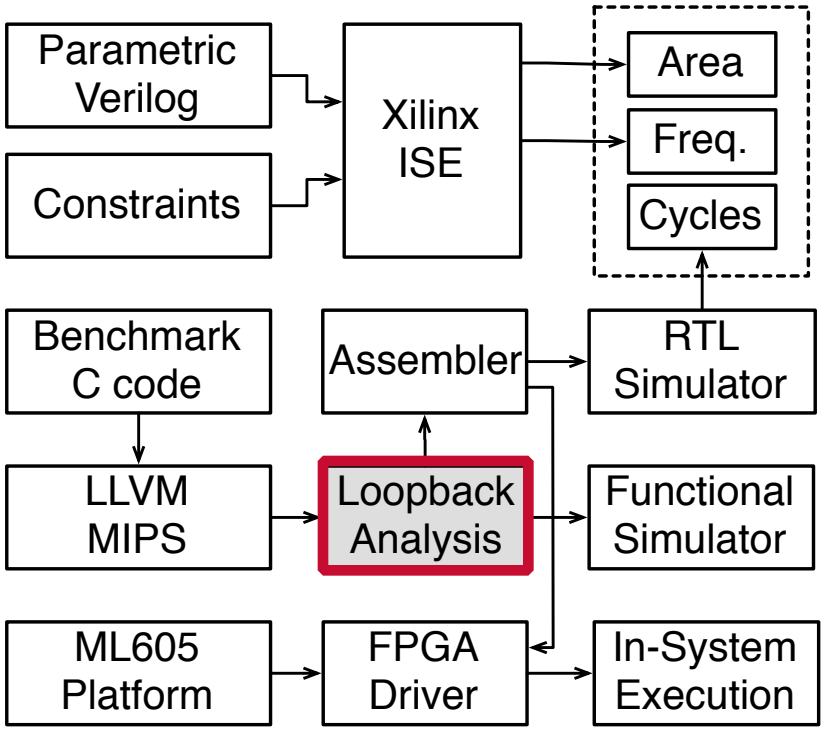




Loopback Analysis

- Identify loopback opportunities in assembly
- Loopback instruction — a subsequent dependent arithmetic operation
- For external loopback: sufficient NOPs to pad the execute stage
- For internal loopback: no NOPs required
- NOPs are inserted for dependencies that cannot be resolved by this forwarding path

Loopback Analysis

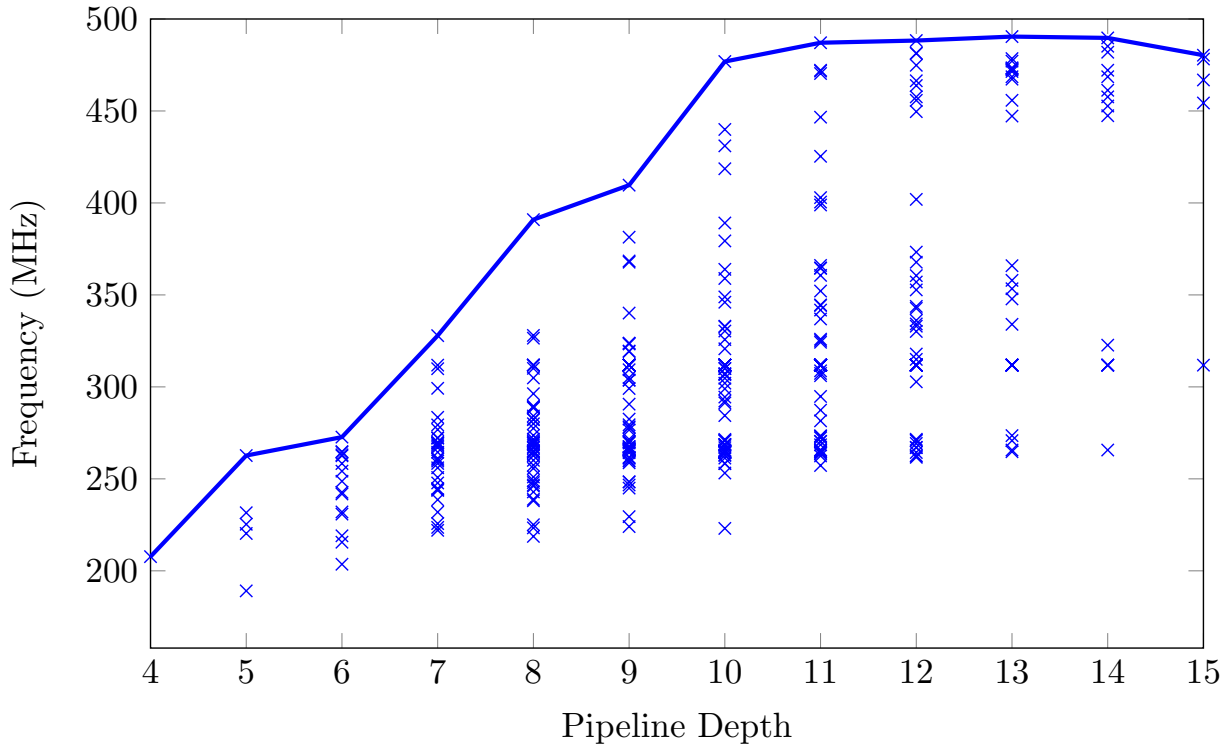


Processor Design Space

- First explore the impact of pipeline depth on area and frequency
- Parameterised register stages in each processor stage (1–5 cycles for Fetch, Decode, Execute)
- Overall pipeline depths of 4–15 cycles
- Multiple possible combinations for each overall depth

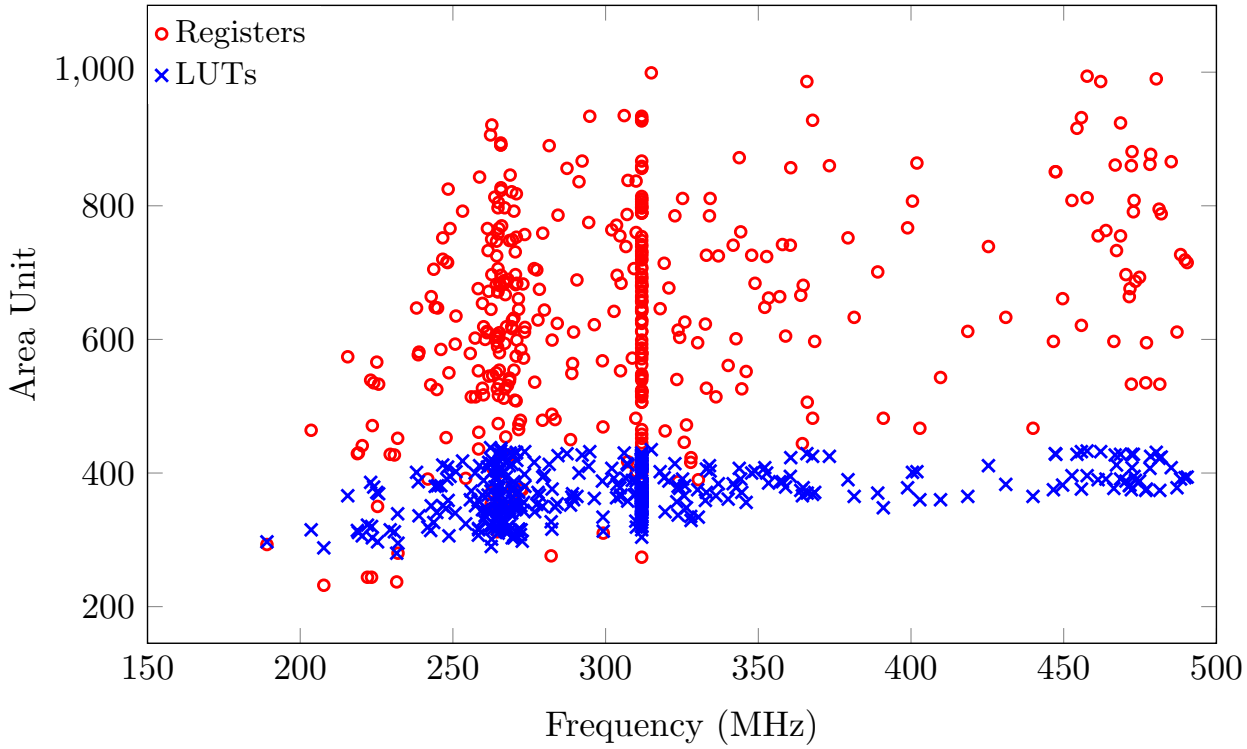
Processor Design Space

- Consider all combinations of stage depths for each overall processor depth
- Reach close to 500MHz from 10 stages onwards



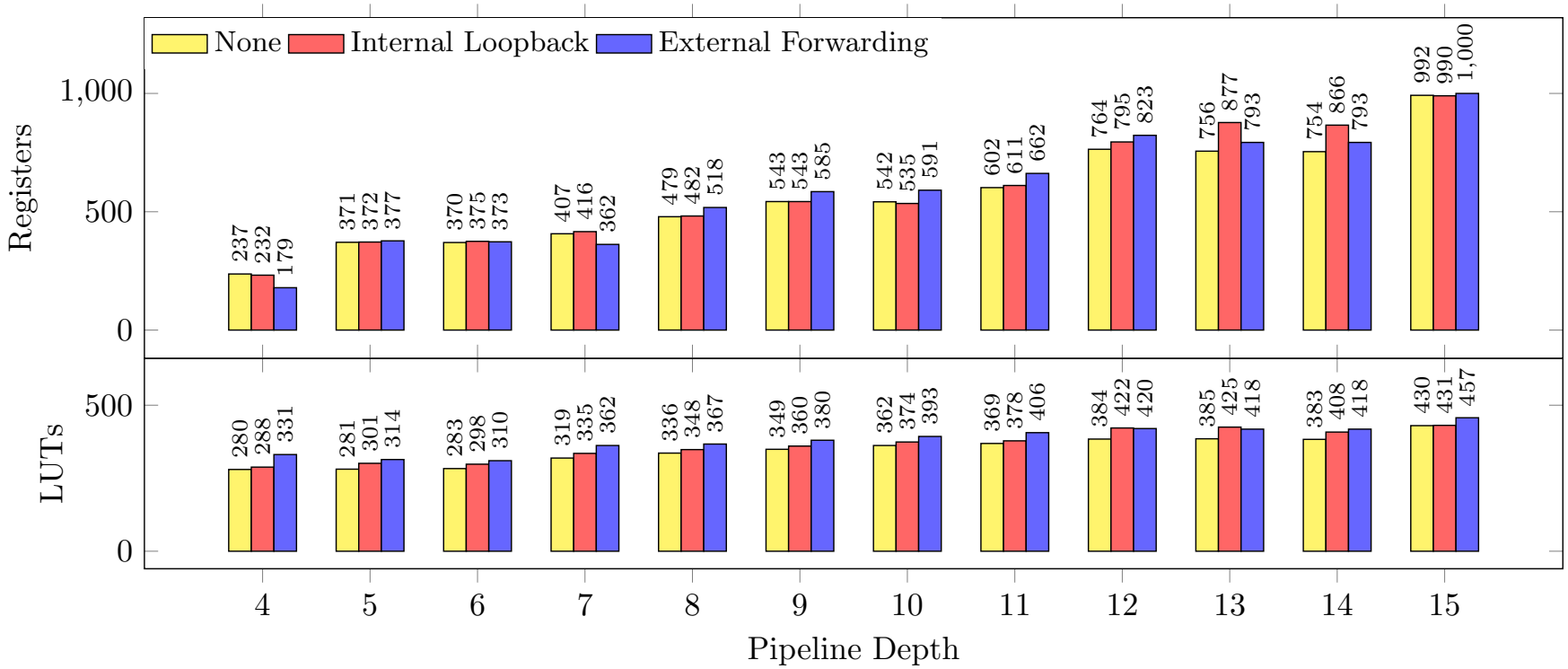
Processor Design Space

- Clusters near where DSP block throughput limits performance
- Significantly more registers than LUTs



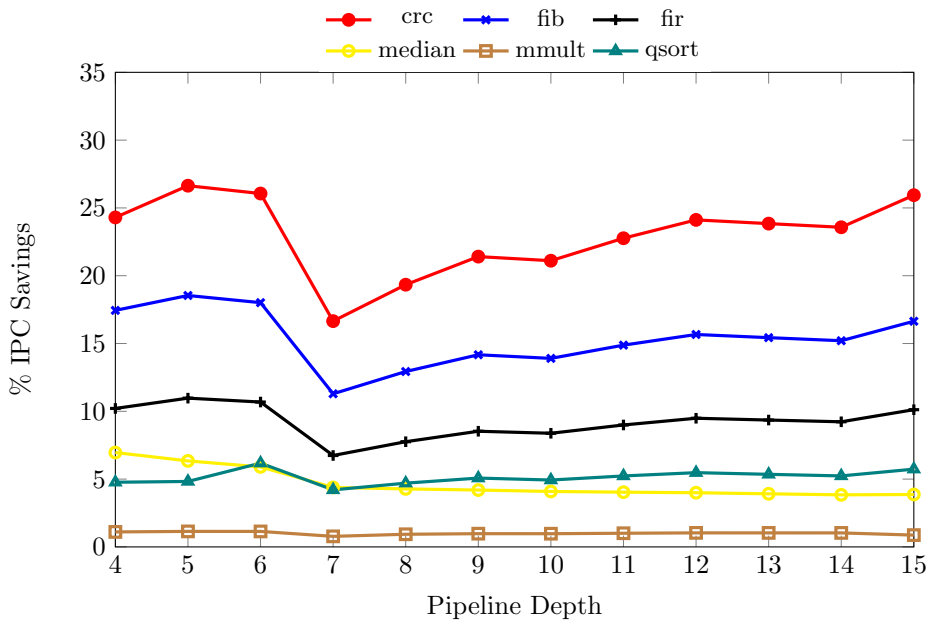
Processor Design Space

- Generally minimal impact for internal forwarding and a little more for external forwarding



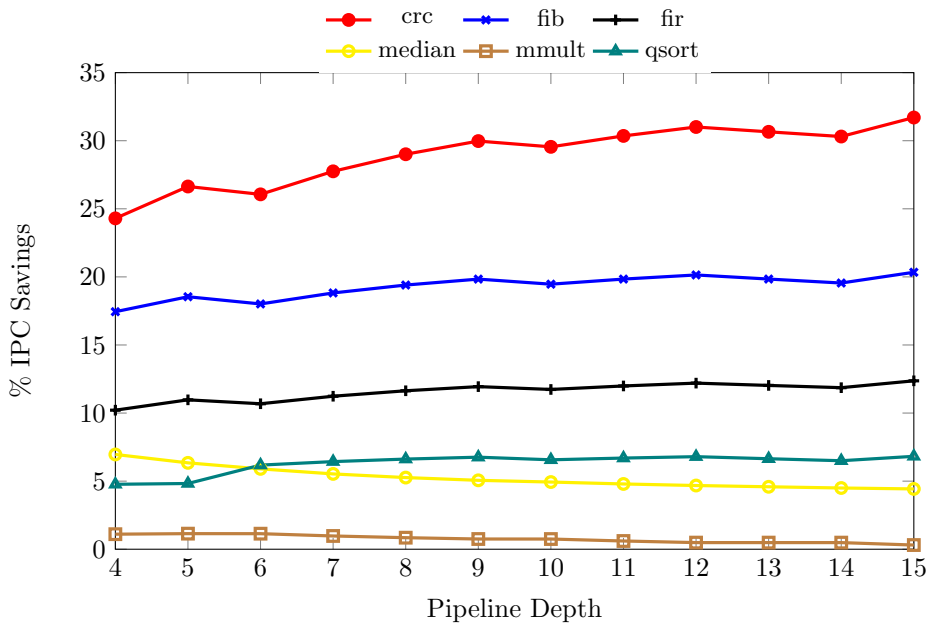
IPC Improvement

- External Loopback
  - Padding NOPs not totally eliminated
  - For depths of 4–6 (with 1-cycle EX), no NOPs needed
  - As EX depth increases, savings are curtailed due to need for some NOPs

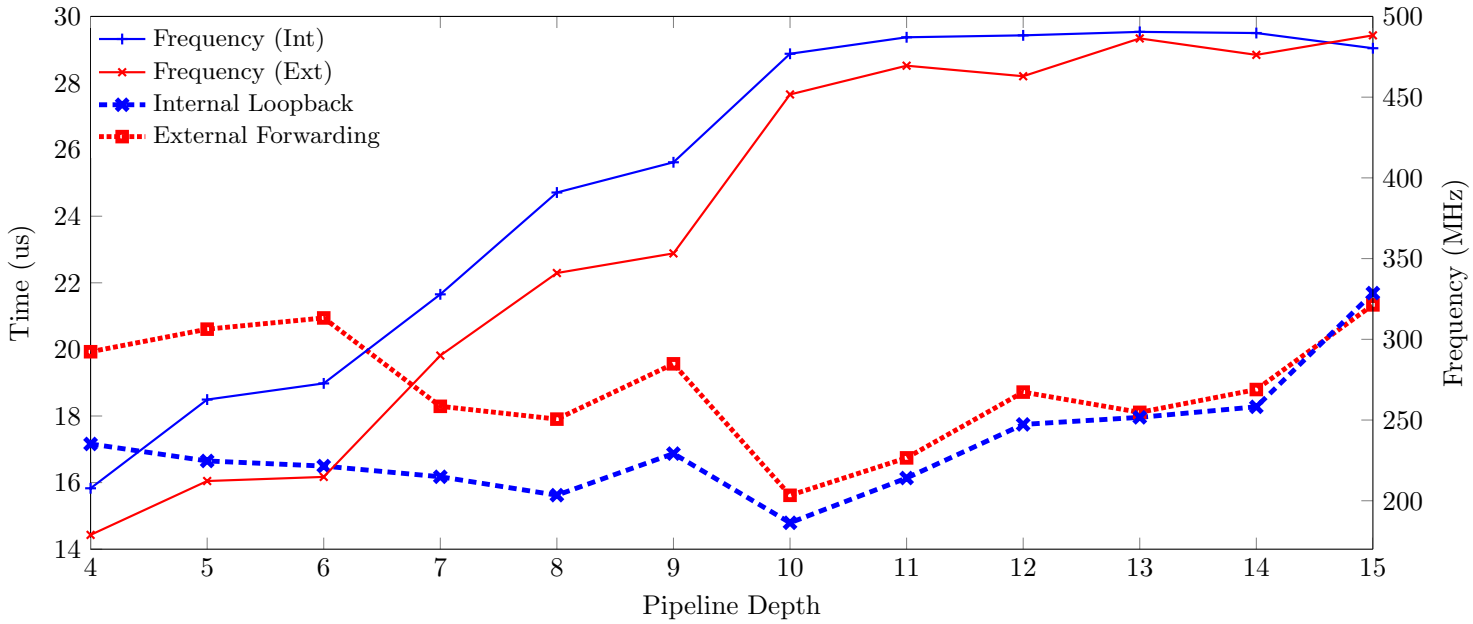


IPC Improvement

- Internal Loopback
  - Sustained savings over no forwarding
  - Benchmarks with long dependency chains
  - A 5–30% improvement

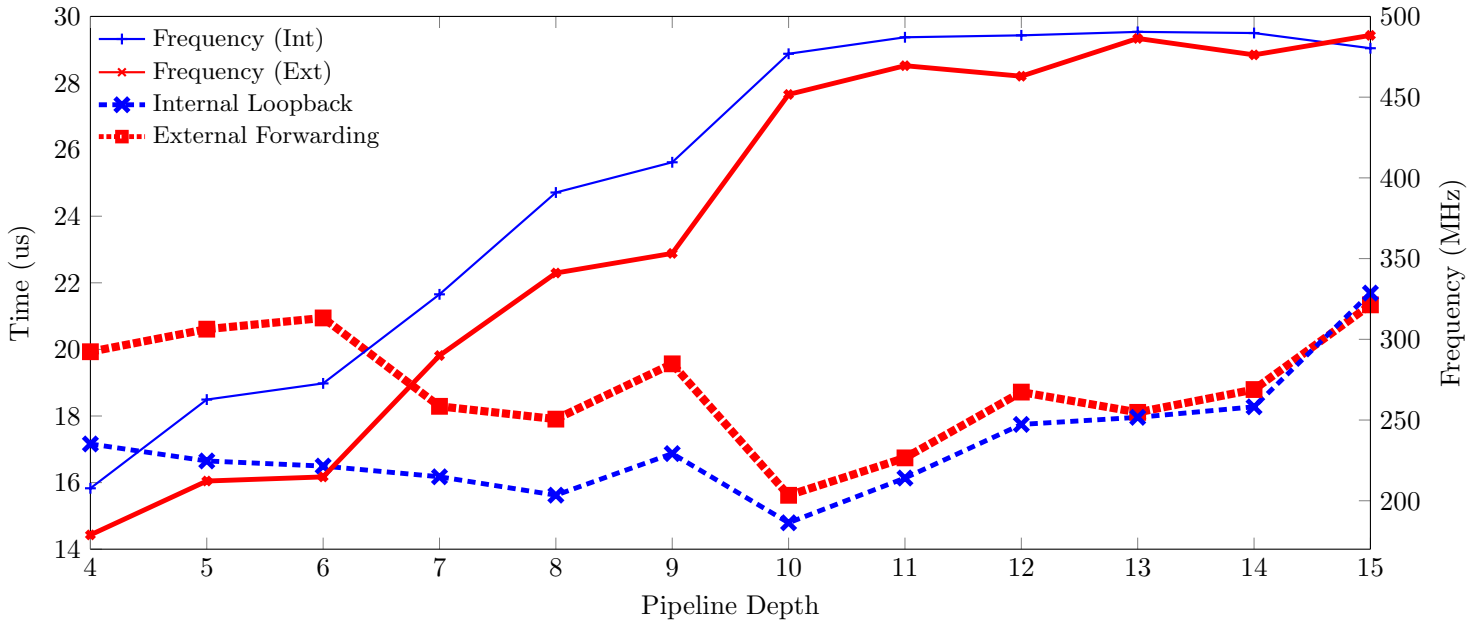


Execution Time



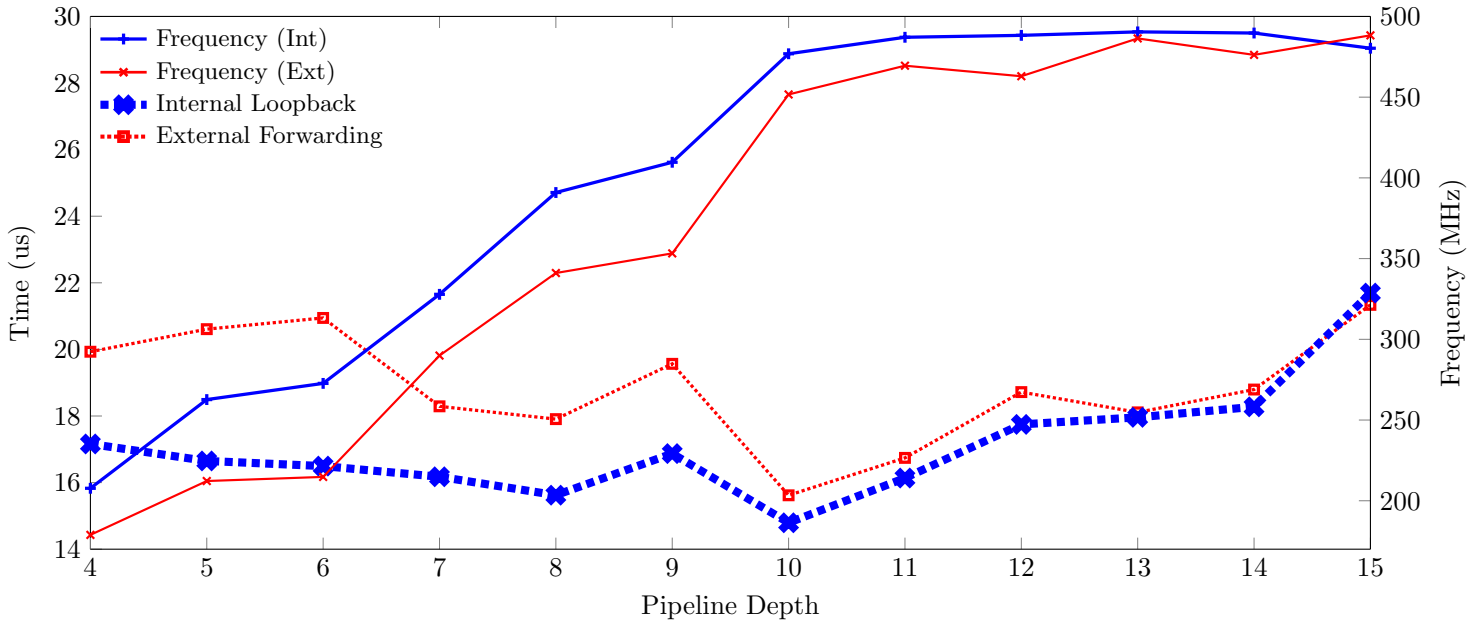
- Frequency and geomean wall clock time for all benchmarks
  - Frequency of external loopback implementation lags initially
  - At higher depths, differences disappear as the extra cycles are added to stages other than execute
  - A 25% improvement in runtime compared to no forwarding

Execution Time



- External loopback
  - Depth 4 — 6 increase in execution time due to lower operating frequency
  - Gap is significant in shallower depths, but closes as depth increases
  - Reduced frequency is fundamental barrier

Execution Time



- Internal loopback
  - Minimum runtime is at 10 cycle processor depth
  - Anomalous peak at depth 9, due to a significant EX cycle increase
  - Results in 11–20% improvement at low depths

Limitations

- As a post-assembly step, limited by the instruction order chosen by compiler
- Only ALU operations (add, subtract) can benefit from internal forwarding
- Ideally, a compiler that could ensure such instructions are kept together would improve results

Potential

- Aside from small benchmarks, explored potential for CHStone benchmarks

Bench mark	Static			Dynamic		
	Instr.	Occur.	%	Instr.	Occur.	%
adpcm	1367	184	13	71,105	8,300	11
aes	2259	51	2	30,596	3,716	12
blowfish	1184	314	26	711,718	180,396	25
gsm	1205	82	6	27,141	1,660	6
jpeg	2388	95	4	1,903,085	131,092	6
mips	378	15	3	31,919	123	0.3
mpeg	782	80	10	17,032	60	0.3
sha	405	64	15	990,907	238,424	24

Conclusion

- Exploiting low-level DSP block feature enabled efficient data forwarding to overcome significant number of padding NOPs
- Maintain frequency of iDEA at close to 500MHz
- Up to 25% improvement across range of benchmarks
- Demonstrated applicability to larger programs

NANYANG TECHNOLOGICAL UNIVERSITY

On Data Forwarding in Deeply Pipelined Soft Processors

Thank You