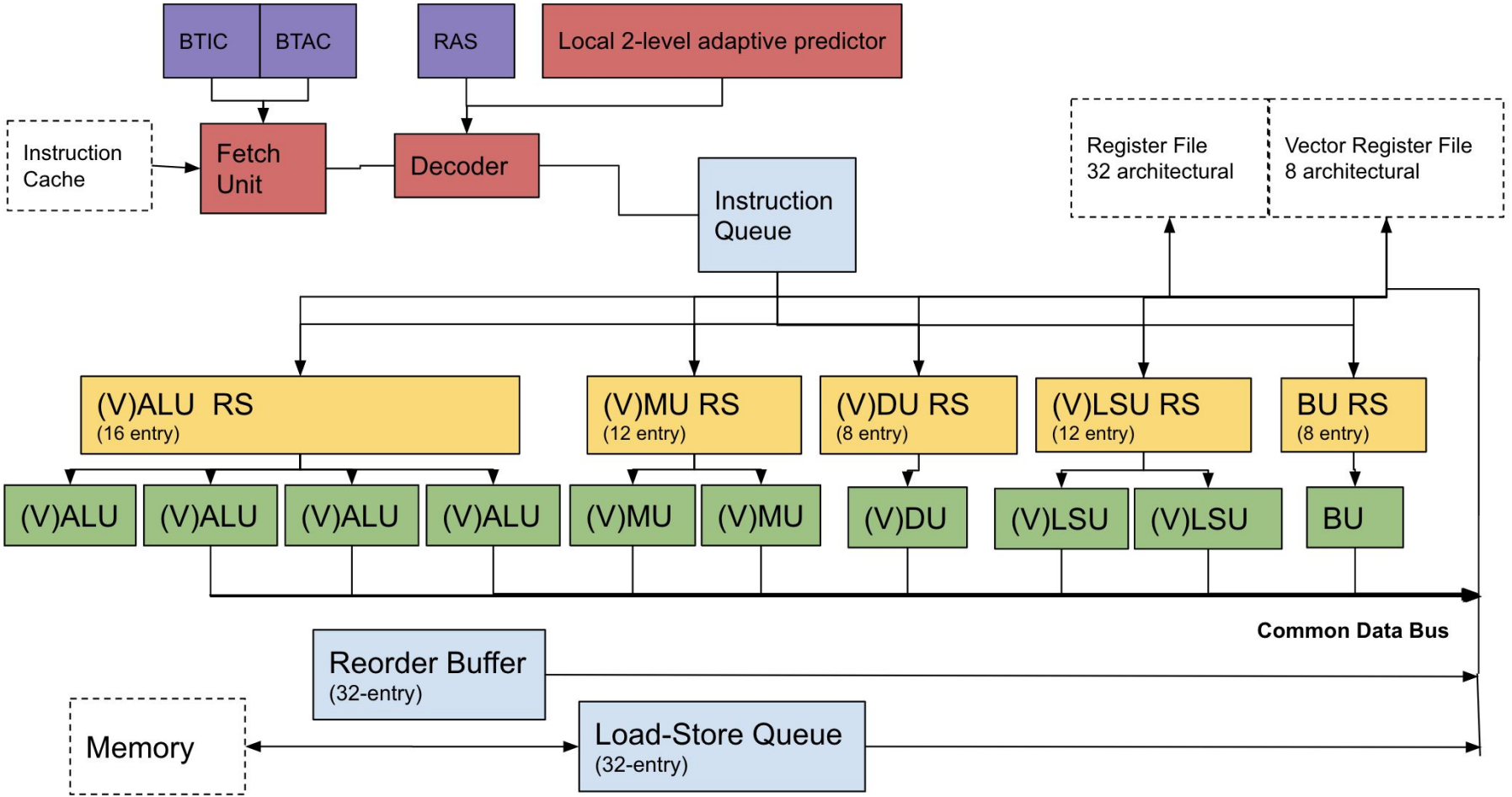
Architecture


BTIC
BTAC
RAS
Local 2-level adaptive predictor
Instruction Cache
Fetch Unit
Decoder
Instruction Queue
Register File 32 architectural
Vector Register File 8 architectural
(V)ALU RS
(16 entry)
(V)MU RS
(12 entry)
(V)DU RS
(8 entry)
(V)LSU RS
(12 entry)
BU RS
(8 entry)
(V)ALU
(V)ALU
(V)ALU
(V)ALU
(V)MU
(V)MU
(V)DU
(V)LSU
(V)LSU
BU
Common Data Bus
Reorder Buffer
(32-entry)
Memory
Load-Store Queue
(32-entry)

Features

Instruction Set

Arithmetic

ADD <dest reg> <src reg> <src reg>
ADDI <dest reg> <src reg> <src reg>
SUB <dest reg> <src reg> <src reg>
SUBI <dest reg> <src reg> <src reg>
MUL <dest reg> <src reg> <src reg>
DIV <dest reg> <src reg> <src reg>

Memory

LW <dest reg> <base address> <offset>
SW <src reg> <base address> <offset>

Unconditional Jump

J <label>
JAL <label>
JR
SYSCALL

- Code 10 - Exit
- Code 41 - Random Int

Conditional Branch

BEQ <reg> <reg> <label>
BNE <reg> <reg> <label>
BLE <reg> <reg> <label>
BLT <reg> <reg> <label>
BGE <reg> <reg> <label>
BGT <reg> <reg> <label>

Vector Instructions

Arithmetic

VADD <dest reg> <src reg> <src reg>
VADDI <dest reg> <src reg> <src reg>
VSUB <dest reg> <src reg> <src reg>
VSUBI <dest reg> <src reg> <src reg>
VMUL <dest reg> <src reg> <src reg>
VDIV <dest reg> <src reg> <src reg>

Memory

VLOAD <dest reg> <base address> <offset>
VSSTORE <src reg> <base address> <offset>

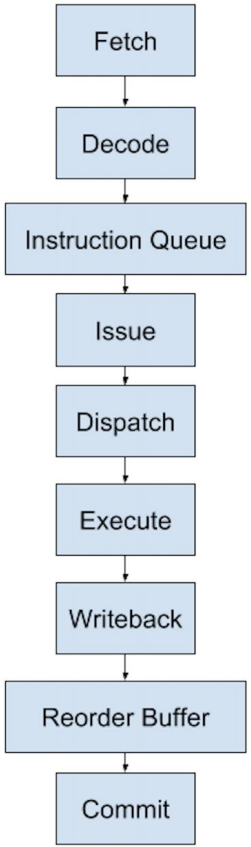
Create Mask Operations

VCMPEQ <reg> <reg> <label>
VCMPGT <reg> <reg> <label>
VCMPLT <reg> <reg> <label>

Use Mask Operations

VBLEND <reg> <reg> <mask> <dest>

Parallelism



N-way superscalar - configurable N

Pipeline

- 7 stage pipeline
- Decouples fetch and decode from execute with a pre-fetch buffer known as the Instruction Queue
- Decouples execute from commit with the Reorder Buffer

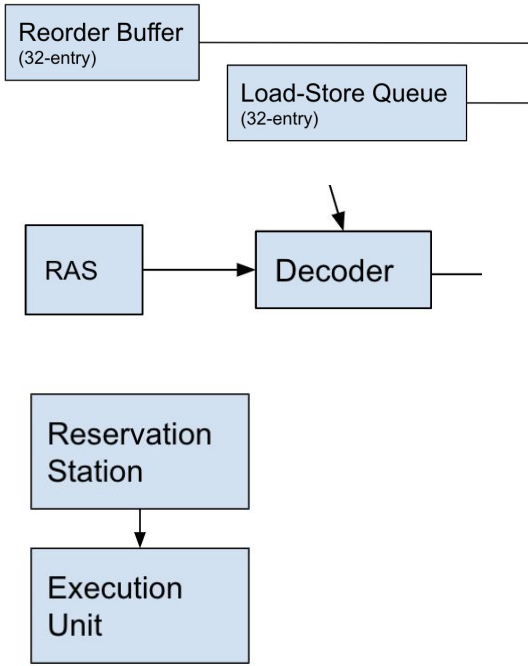
Execution-units

- Different types -
  - Standard - ALU, MU (Multiply Unit), DU (Division Unit) , LSU
  - Vector - VALU, VMU, VDU, VLSU
- Configurable number
- Supports multi-cycle instructions
  - register - 1, loads/stores - 3, multiply - 3, division - 5
  - vector register - 3, vector loads/stores - 5, vector multiply - 5, vector division - 7
- Supports only integer operations (no floating-point)
- Execution-units of multi-cycle instructions are FULLY pipelined. A notable exception if that of the Division-Unit (DU)

Vectorization

- Vector length of 4
- Support bit mask (i.e. VCMPEQ, VCMPLT, i.e. VCMPGT) and conditional load (i.e. VBLEND)
- Vectorized algorithms including dot-product and a max function implemented as a vectorized conditional load w.r.t. a vector mask.

Out-of-Order (OoO)



Tomasulo Algorithm

- OoO execution of register access instructions
- Detects and stalls on true register dependencies

Memory disambiguation

- OoO execution of memory access instructions
- Detects and resolves true memory dependencies
- Store-to-load forwarding

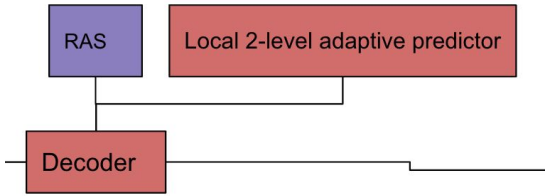
Register Renaming and Branch Recovery

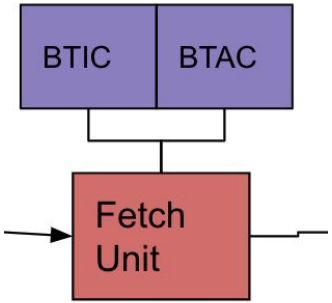
- Resolves false dependencies (WAW, WAR) by storing results in the reorder buffer No need for register renaming.
- Requires a register Register Alias Table (RAT) to keep track of mapping of registers to Reorder Buffer indices.
- Recover from mispredicted branches by flushing at commit.
- Recover from speculatively loaded value of memory address by flushing at commit.

Priority writeback

- When more than 4 instructions are available to be written-back.
- Priority is given to slow multi-cycle instructions i.e. DIV, MUL
- Reasoning - Assumably many more dependant instructions present in the pipeline for slower instructions

Branch Prediction





Speculative Execution

- Infinite speculative choices and speculative depth

Target Prediction

- Branch target address and target instruction caching support
  - Branch Target Address Cache (BTAC)
  - Branch Target Instruction Cache (BTIC)
- Resolve the target of unconditional relative jump instructions i.e. jump, jump and link.

Branch Prediction

- Supported implementations include
  - Static prediction
  - 1-level dynamic predictor: N-bit saturating counter (N is configurable)
  - 2-level dynamic predictor: Local 2-level adaptive predictor
    - Uses a branch history register table (S-bit saturating counter)
    - Uses a N-bit local pattern history table
    - N and S configurable

Return Address Prediction

- Prediction of unconditional in-direct jumps such as that of return
- Implementation hinges on Return Address Stack (RAS)
- RAS Checkpointing scheme to recover the original state of the RAS before failed speculative execution

Benchmark Tests

Bubble Sort	In-place sort of randomly initialized array	Erratic Inner Loop
Fibonacci	Calculates the 5th fibonacci number	Recursive
GCD	Computes greatest common divisor of 2 numbers	Erratic Inner Loop
Dot Product	Computes the inner/dot product of 2 vectors	Basic Loop, Vectorised
Matrix Multiplication	Matrix Multiplication	Complex Nested Loop
Max Function	Computes c[i] = a[i] if a[i] > b[i] else b[i]	Vectorised conditional load from mask

Experiments

Hypothesis - Branch Prediction

Statement 1

- 1-level or 2-level dynamic prediction schemes provides minimal support benefit over static prediction

Statement 2

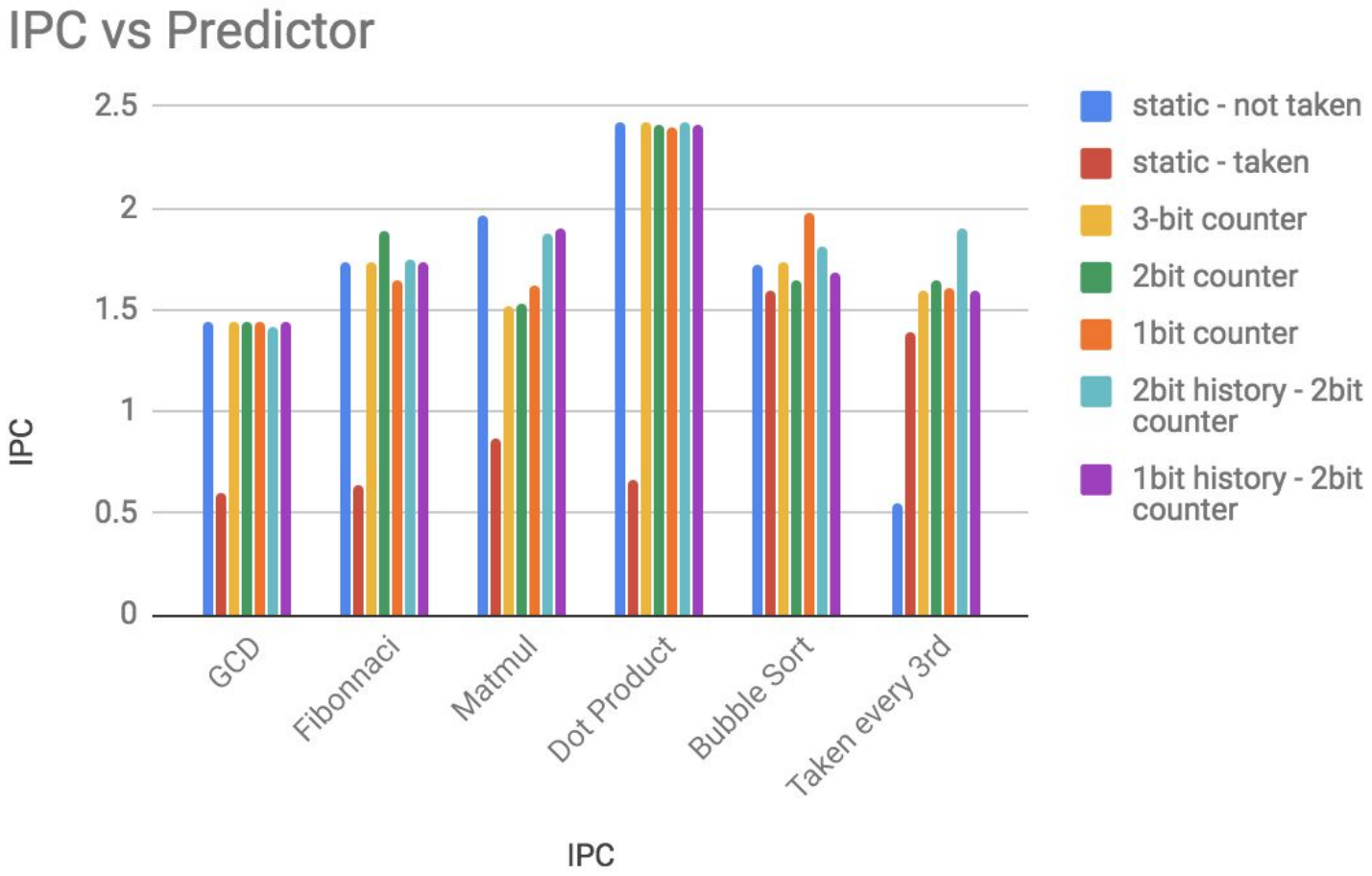
- 1-level prediction provides benefit in situations where a branch is more favourable taken or not-taken i.e. not a loop closing branch

Statement 3

- 2-level prediction schemes should do better in cases where there is a repeating pattern
- The “Taken every 3rd” program on the right has a loop with an IF-ELSE statement inside. This conditional branch will be taken every 3rd iteration. We assume to see benefits for adaptive predictors that take into account the history of the branch.

```
 4 main:
 5 li $s0 c0
 6 li $s1 c50
 7 j loop
 8
 9 loop:
10 mod $t0 $s0 c3
11 bne $t0 $zero else
12 addi $s0 $s0 c1
13 bge $s0 $s1 exit
14 j loop
15
16 else:
17 addi $s0 $s0 c1
18 bge $s0 $s1 exit
19 j loop
20
21 exit:
22 li $v0 c0
23 jr $ra
24
```

Experiment - Branch Prediction



Results - Branch Prediction

Observation 1
- The always not taken static predictor almost always achieves performance as good performance as the dynamic predictors, if not a little behind.

Reasoning -
- The static branch prediction is very effective on a primarily loop based program and performs with 50/50 accuracy on IF-ELSE branches.

Observation 2
- The 2-bit counter (almost) consistently performs better than the 3-bit counter
- The branch history predictor didn't performed as well as the 2-bit predictor but performs similarly for long programs such as matrix multiplication and dot product

Reasoning - Too many failed speculations to shift the counter towards the correct guess

Observation 3
- The 2-bit history-2-bit counter, adaptive predictor performed exceedingly well on the "taken every 3rd" toy program

Reasoning - adaptive predictors work best with repeating patterns - even/odd or every N-th iteration if-else in loops

Observation 4
- The change in IPC was small for better predictions

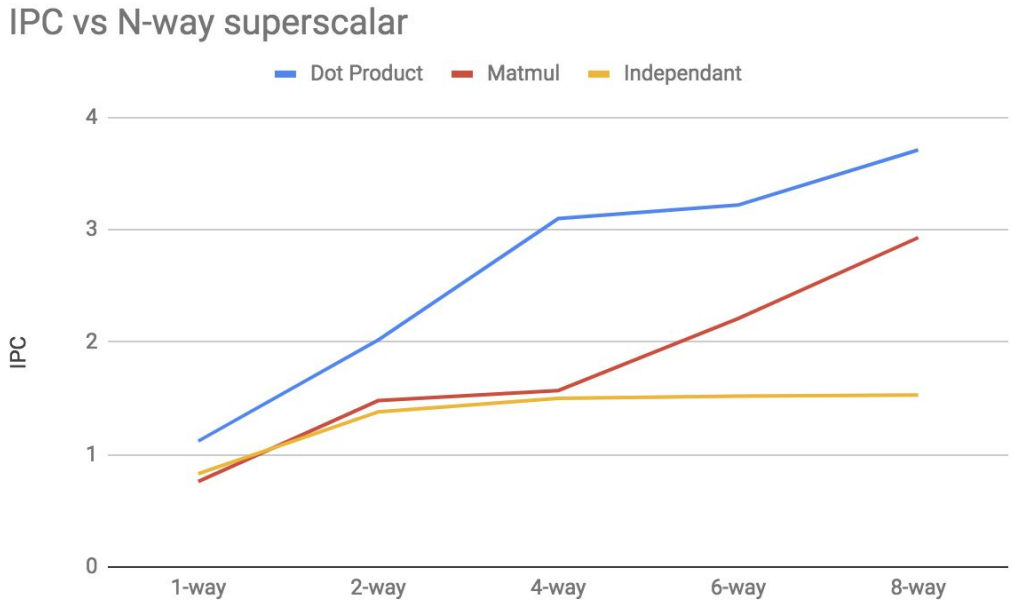
Reasoning -
- The branch prediction accuracy is already very high (theoretically greater than 85%)
- Architectural limitations - lack of execution units, Full Reorder-Buffer etc.
- Program limitations - RAW dependencies, Slow instructions etc.
- Large amount of work within each loop drowns out the advantage of BP

Hypothesis - Parallelism

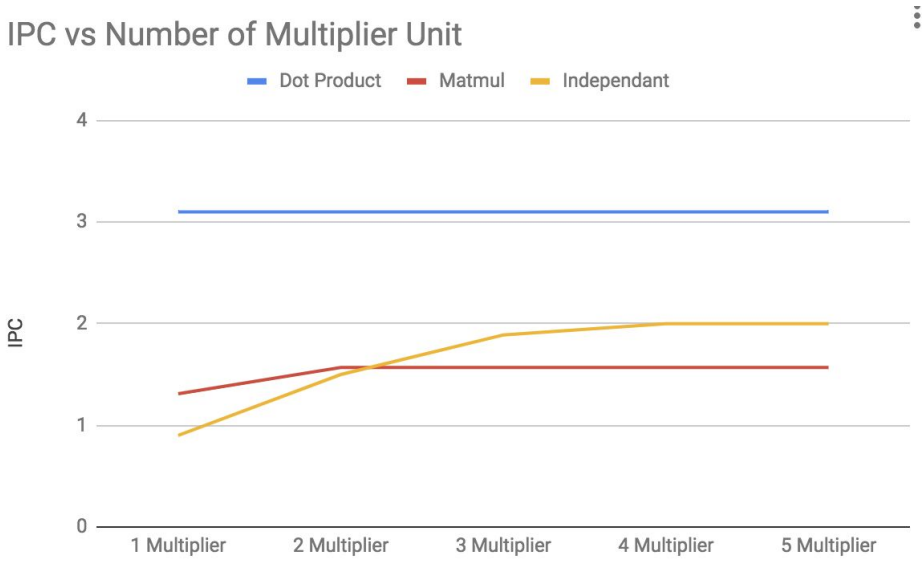
Comparing the effects of different methods of parallelism

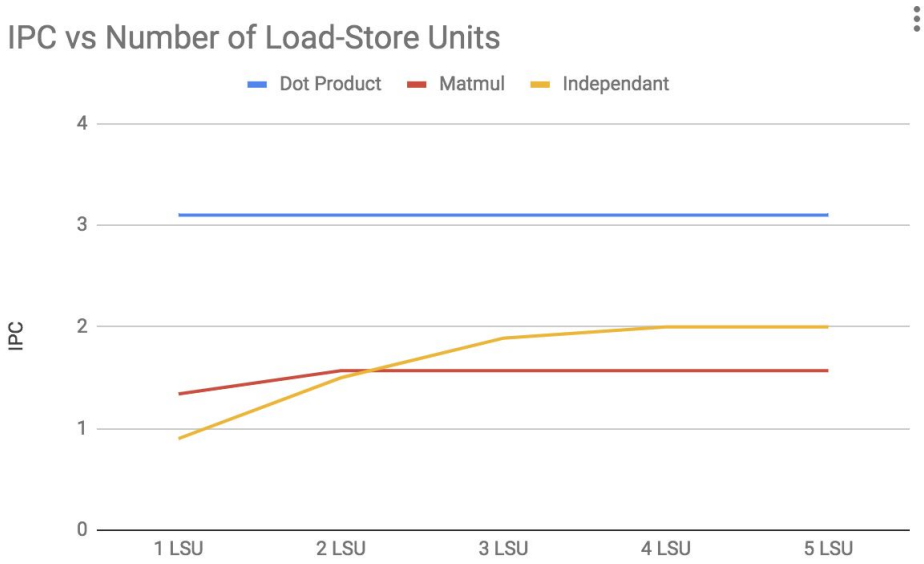
- Effect of pipeline width i.e. increasing the number of execution units
- Effect of pipeline depth i.e. pipelining the execution units
- Effect of superscalar
- Effect of vectorization

Programs of a relatively small number of true dependencies to total executed instructions will benefit greatly from the increased instruction and data parallelism.



Experiment - Parallelism





Result - Parallelism

Observation 1

- Programs with minimal dependencies as expected perform better under superscalar issue but rate of improvement slows down

Reasoning -

- In the presence of good branch prediction, the pipeline will be kept saturated upto the point where dependencies between instructions become the limiting factor

Observation 2

- Matrix multiply benefit more than the dot-product w.r.t. Superscalar issue

Reasoning -

- Loops that of a large amount of computation (e.g.. calculation of the indices, in matrix multiplication) lends itself better to superscalar and OoO execution.

Observation 3

- The stream of independant instructions benefitted largely from increasing the number and pipelining the execution units but matrix and vector operations did not.
- These instructions benefitted from an increase in Load-Store Units not Multiplier Units.

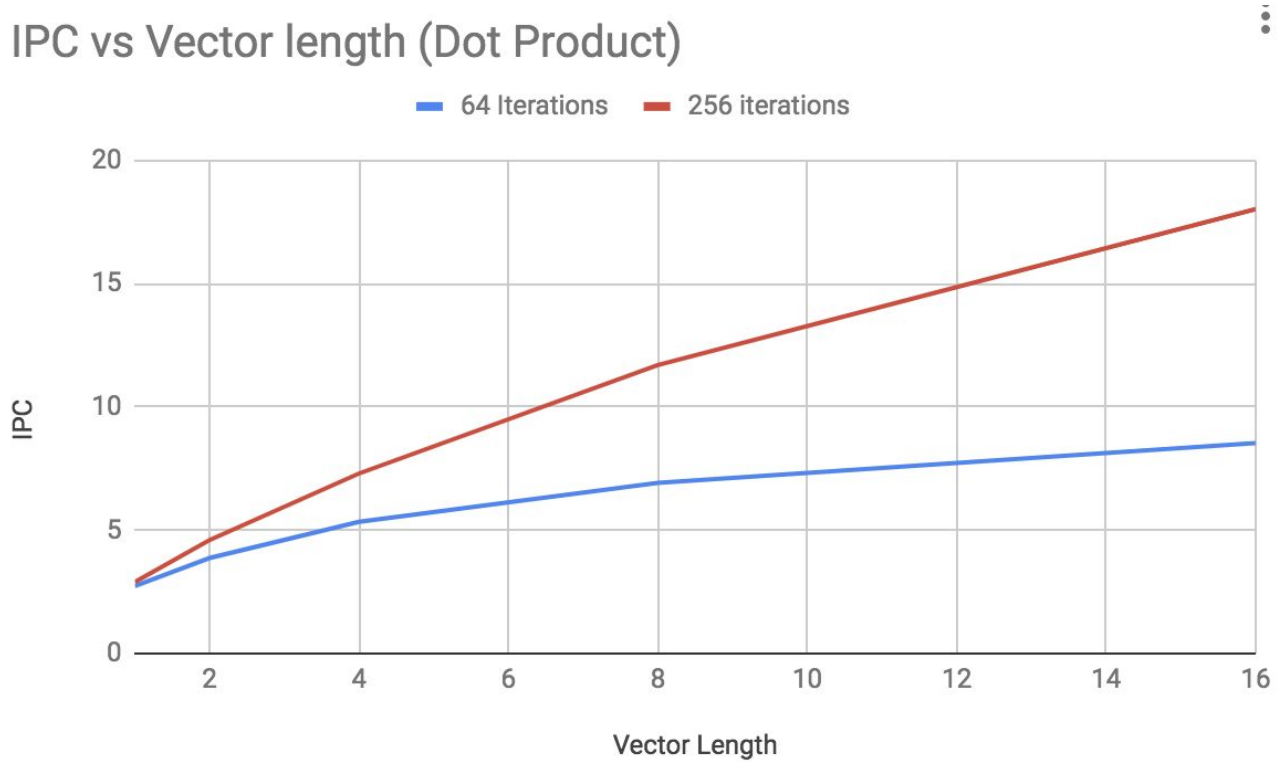
Reasoning -

- The stream of independant instructions stop benefitting after it cannot issue any more than is being executed.
- Matrix Multiplication and Dot Product operations are memory bandwidth bound, not compute bound.

Experiment - Parallelism

Observation 4

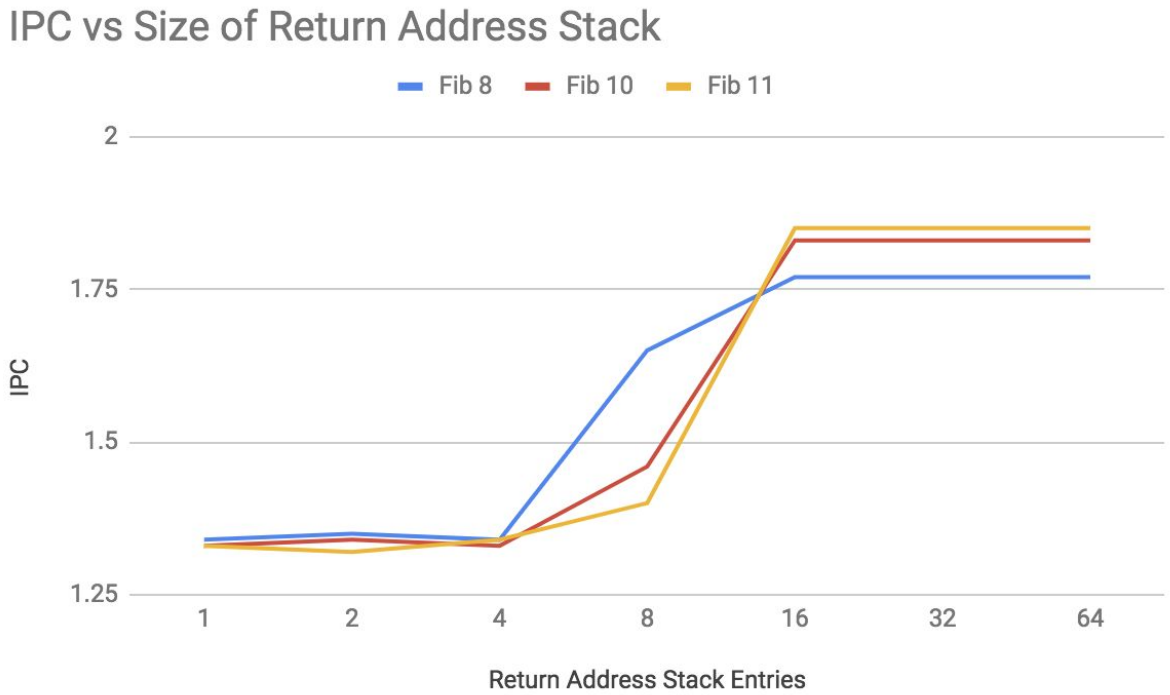
- Vectorization gives a consistent speedup for an given increase in the vector length
- Benefits longer loops, more iterations to vectorize



Hypothesis and Experiment - Return Address Prediction

Statement

- Recursive programs that make good use of the stack to store return addresses can benefit from return address prediction
- We analyse the benefit of various size of the Return Address Stack - a method of return address prediction



Results - Return Address Prediction

Observation

- The benefit of the number of entries plateaus after 16 entries
- Deeper nested recursive programs benefit more from a larger Return Address Stack (RAS)

Reasoning

- Rate of conditional-branch mispredictions increase significantly beyond this point, nullifying any benefit