Quals Question
Prof. Olukotun

Computer Architecture

Q: What are precise exceptions?

A: All instructions before exception PC have been execute d and have modified machine state
All instructions following exception PC are unexecuted
The instructions following the exception PC are restartable

Q: Give an example instruction that must be handled precisely

A: Page fault

Q: How do you implement precise interrupts in a simple instruction pipeline that has in-order instruction completion?

A: Post exception in a status register that can be checked at a common commit point. At the commit point in the execution pipeline all interrupts must have occurred but the state to the machine must not have been modified. In a DLX style pipeline this point is just before the write-back stage.

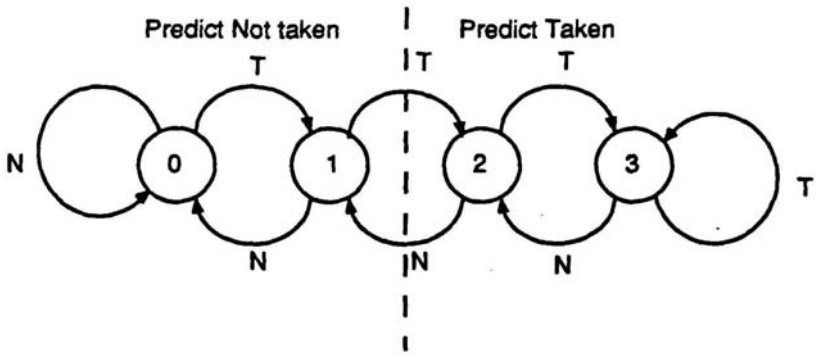
Q: How do you implement precise interrupts in a machine that has out-of-order instruction completion?

A: A possible answer is use a reorder buffer.

Q: How does the reorder buffer work?

The reorder buffer is FIFO queue that is placed between the output of the functional units and the write-back port of the register file. It keeps the register file in a precise state. The entries of the reorder buffer are en queued when instructions are issued. Each entry contains the following fields: destination register, result value, PC, interrupt status, valid. Instructions are removed from the head of the queue when they have valid result values after they have written back their results. Exceptions are checked for an instruction when the instruction reaches the head of the queue. If an instruction causes an exception all entries behind it in the reorder buffer are discarded and do not write back their results. Bypassing of result values from the reorder buffer is required for maximum performance.

2-bit branch prediction counter



Question:

The above counter requires 2 bits per branch in a branch history table.
Suppose you may keep as much state (bits) as you want, devise a general scheme to improve branch prediction accuracy. Use the code sequence and branch history trace below to devlop your answer. Show specifically how your scheme would be an improvement over the 2-bit counter prediction accuracy for branch b3.

```
b1:   if (aa == 2)
          aa = 0;

b2:   if (bb == 2)
          bb = 0;

b3:   if (aa != bb)
      {
          .....
      }
```

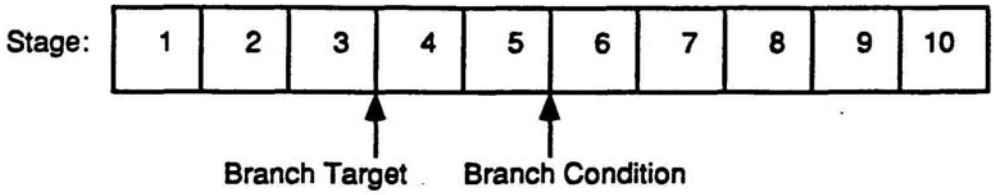
Time →

b1:	T	N	N	N	N	T	T	N	T	T	T	T
b2:	N	N	T	T	N	T	T	T	T	T	N	N
b3:	T	T	N	T	T	N	N	T	N	N	N	N

Answer:

There are many possible answers. The best approach is to realize that the behavior of branch b3 is correlated with the behaviors of branches b1 and b2. And that in general branch behavior may be corelated with previous branches. Thus, by keeping track of whether the last n branches were taken or not taken and by associating a 2 -bit counter with each combination of branch outcomes we may significantly increase branch prediction accuracy. Really good answers showed how the branch history trace for b3 can be broken into four cases using the branch histories of b1 and b2. The best answers showed how you would implement this scheme in a branch history table.

Kunle Olukotun
January 1994 Quals Questions



Question:

For this pipeline what percentage of branches must be taken so that the "always taken" and "always not taken" schemes have the same performance

Answer:

To solve this problem we need to equate the branch penalties for each prediction scheme. If t is the percentage of taken branches, then we get the following equation:

Always taken		Always not taken
$2t + 4(1 - t)$	=	$4t$

The rest is algebra

$t = 66.7\ \%$

To: shankle@ee.Stanford.EDU
Subject: Quals question
r: Thu, 09 Feb 95 10:48:36 PST
n: kunle@ogun.Stanford.EDU

Q. If I wanted to compare the performance of two computer systems on a set of floating point intensive benchmarks using MFLOPS, how would I go about it?

A. Measure normalized MFLOPS of each benchmark. Begin by counting the number of normalized floating point operations in the source program. This number, which is the same for both computers, is divided by execution time of the benchmark to produce MFLOPS. Use harmonic mean to get an average MFLOPS rating across the set of benchmarks for each computer. This measure will track execution time which is the real measure of performance. Compare the machines using the average MFLOPS rating.

Q. Compare a branch prediction buffer (BPB) and a branch target buffer (BTB).

A. Talk about the cost versus performance of the two schemes.
BPB: low cost. Can reduce cost by eliminating the tag. Only useful for conditional branches.

BTB: higher cost because it must store tag and target address. BTB can be used for both conditional branches and unconditional branches. The BTB has lower branch delay on a hit than the BPB.

Given the same area a BPB would have more entries and better prediction accuracy and than the BTB.

Kunle leave

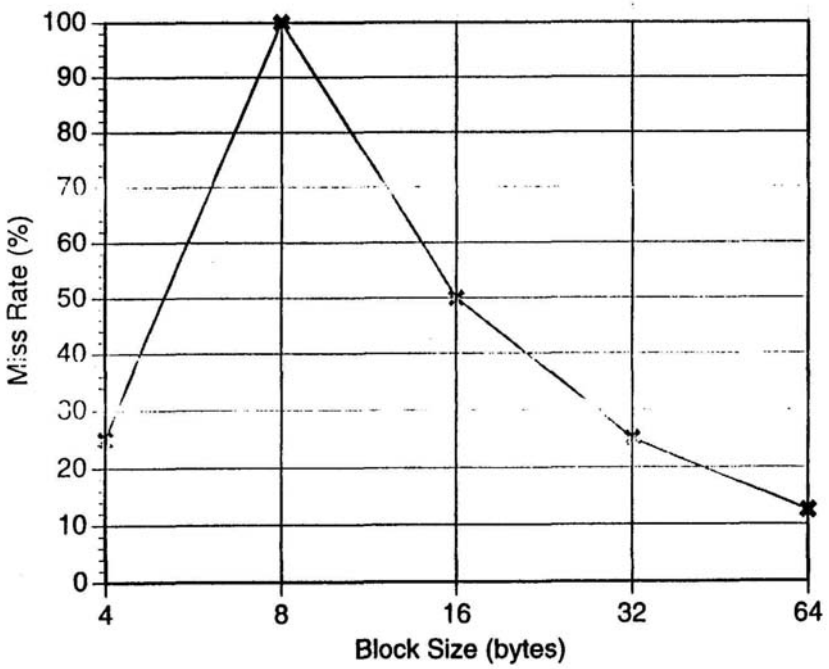
Cache

16384 bytes
Fully associative
LRU replacement

Application

```
int x[8192], y, i, j; /* 4 byte integers */

for (i = 0; i < 4; i++)
   for (j = 0; j < 8192; j += 2)
        y = x[j];
```



Q: Graph the miss rate vs. block size for the cache and application.
A: Shown on graph

Q: What types of locality are being exploited at the various block sizes?
A: 4B-temporal, 8B-none, 16B–64B-spatial

Q: What types of misses occur at the at the various block sizes? Use 3-Cs model
A: 4B-compulsory, 8B–64B-compulsory, capacity. No conflict misses.

Q: Which block size provides the best performance? Assume 4B wide refill bus.
A: Depends on latency (LA) and bandwidth (BW) of cache refill. To achieve higher performance with a 64B block than with a 4B block: LA > 60B/BW.

Kunle Olukotun

Computer Architecture

Which application class will benefit most from each enhancement? Explain your reasoning.

	Floating point Scientific	Large DB applicaiton
Large Icache		
Static branch prediciton		
Lots of registers in the ISA		
Victim cache		
Software Prefetching		
4-way S.A. Icache		
Fetch both branch target and fall-through		
Wide Instruction issue with dynamic scheduling		
More branch displacement bits		
Nonblocking cache with many outstanding misses		
Large Dcache block size		
Deep pipelining		
Hardware Prefetching		
High Main Memory bandwidth		

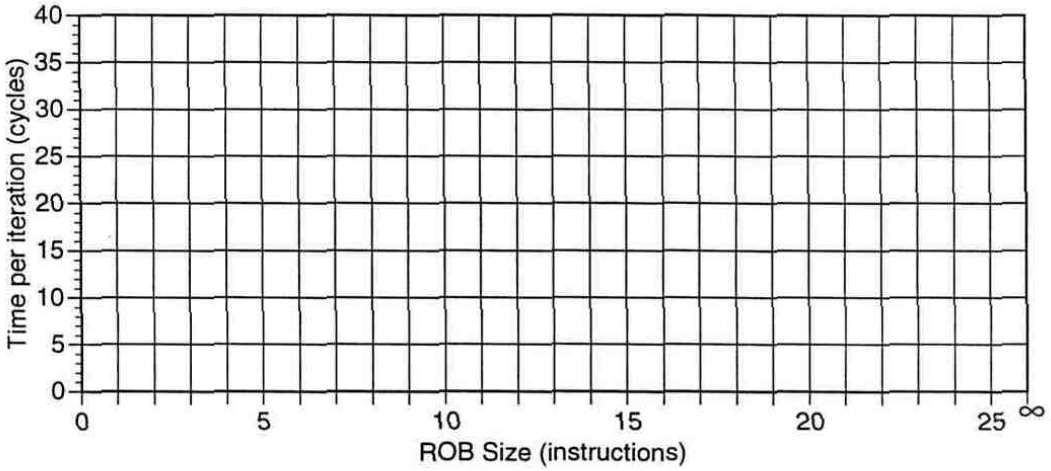
computer Architecture

K. Olukotun

1. What's a non-blocking cache (NBC)?
2. How does it improve AMAT?
3. What do you need in processor and memory system to make a NBC most effective?
4. Fill in the graph below? (assume a pipelined memory system)

```
for (i = 0; i < 1000; i +=2)
      B[i] = B[i] + C[i] + B[i+1] + C[i+1]; /* all arrays are 4-byte
                                              integers */

        top:  LD   R1, B[i]
              LD   R2, C[i]
              ADD R3, R1, R2
              LD   R4, B[i+1]
              ADD  R5, R3, R4
              LD   R6, C[i+1]
              ADD R7, R5, R6
              ST   R7, B[i]
              3 instrs to update i, B[i], C[i]
              BLT R8, #1000, top
```



Assumptions

1. fully associative non-blocking data cache, initially empty
2. 8 byte cache line size
3. write back, write allocate
4. 10 cycle miss penalty
5. single issue fully pipelined processor
6. perfect branch prediction

Quals Question: Professor Olukotun

1. Why is dynamic branch prediction important in modern processors?

2. Given then general classes of scientific and transaction processing applications. In which class are branches harder to predict? Why?

3. What is the simplest dynamic predictor you could use?

4. Given an unlimited number of single bit predictors without using any other kinds of history for any particular application at what point will the branch prediction accuracy saturate?

5. Suppose we want to improve the branch prediction accuracy beyond this point by using more branch history. There are two types of history, what are their names?

6. Define local and global history, explain how they can be used to improve branch prediction accuracy. Why are more predictors required? Why does this work?

7. Assume branch B1, B2 and B3 are executed repeatedly in a loop and two bits of history are kept for each branch. For each history bit determine which history information (local or global) will provide the best branch accuracy for branches B2 and B3. Indicate your answer by placing a G or an L in the appropriate box.

Assume any initial condition of the predictors that you like.

									G/L	G/L
b1:	T	N	T	T	T	N	T	N		
b2:	N	N	T	N	T	T	N	N		
b3:	T	T	N	T	N	T	T	T		

8. How can you decide dynamically which history to use?

Oyekunle
Olukotun

Parallelism and Locality

1. How do parallelism and locality get used in modern microprocessor designs?

Assumptions

- 20 issue OOO processor, unlimited window size
- Perfect branch prediction
- 4 MB F.A. 1-word line cache, single cycle access
- 100 cycle main memory access
- Nonblocking cache
- No structural hazards

1. Program mallocs and initializes a 1 MB linked list data structure
2. Program runs a C loop:

```
for (p=head; p!=NIL; p = p->link)
    ++(p->value);
```

```
        J       test
loop:   LW      R5, 0(R4)
        ADDI    R5, R5, #1
        SW      R5, 0(R4)
        LW      R4, 4(R4)
test:   BNEZ    R4, loop
```

Assume 100,000 iterations with this data set

1. X and Y are in main memory
2. C loop:

```
for (i=0; i < 100,000; i++)
    Y(i) = a*X(i) + Y(i);
```

```
foo:  LD     F2, 0 (R1)       ; load X(i)
      MULTD  F4, F2, F0       ; multiply a*X(i)
      LD     F6, 0 (R2)       ; load Y(i)
      ADDD   F6, F4, F6       ; add a*X(i) + Y(i)
      SD     0 (R2), F6       ; store Y(i)
      ADDI   R1, R1, #8       ; increment X index
      ADDI   R2, R2, #8       ; increment Y index
      SGTI   R3, R1, #100000; test if done
      BEQZ   R3, foo          ; loop if not done
```

2. Which loop is faster? Why? How much faster?

Kunle Olukotun

Perfect Branch Prediction vs. Perfect Data Cache

Warm up question

The genie of computer architecture has give you one wish. You can have a processor with perfect branch prediction or a perfect cache. Which do you choose and why?

Perfect Branch Prediction vs. Perfect Data Cache

Assumptions

- Single issue out-of-order processor, unlimited window size
- 10 cycle branch delay
- Single cycle access, nonblocking cache, 1 word block size
- Memory accesses have 10% miss rate
- 100 cycle main memory access
- No structural hazards

```
for (i=0; i < 100,000; i++)
      Y(i) = (X(i) == 0.0) ? X(i) : Y(i-1);

      addiu  $s3, $s0, #400000      ; initialize $s3
loop: lw     $f2, 0 ($s0)           ; load X(i)
      c.eq.s $f0, $f2               ; X(i) == 0.0
      bfpf   yi-1                   ; 50% mispredict
      j      yi                     ;
yi-1: lw     $f2, -4($s1)           ; Y(i-1)
yi:   sw     $f2, 0($s1)            ; store Y(i)
      addiu  $s0, $s0, #4           ; increment X index
      addiu  $s1, $s1, #4           ; increment Y index
      slt    $s2, $s0, $s3          ; test if done
      bnez   $s2, loop              ; 0% mispredict
```

Which performs better perfect branch prediction or perfect cache?

Perfect Branch Prediction vs. Perfect Data Cache

Assumptions

- Single issue out-of-order processor, unlimited window size
- 10 cycle branch delay
- Single cycle access, nonblocking cache, 1 word block size
- Memory accesses have 10% miss rate
- 100 cycle main memory access
- No structural hazards

```
for (p=head; p!=NIL;)
    if (p->value == 0)
        p = p->link1;
    else
        p = p->link2;
```

```
        J       test
loop:   lw      $s0, 0($s1)
        bnez    $s0, link2      ; 50% misprediction
        lw      $s1, 4($s1)
        J       test
link2:  lw      $s1, 8($s1)
test:   bnez    $s1, loop       ; 0% misprediction
```

Assume 100,000 iterations

Which performs better perfect branch prediction or perfect cache?

Kunle Olukotun

Hiding Memory Latency

Assumptions

- 32 KB cache
- 1 GB data structure

```
for (i = 10,000; i > 0, i--)
    for (p=head; p!=NIL;)
        if (p->value < i)
                p = p->left;
        else
                p = p->right;
```

```
        addiu   $s2, $s0, #10000        ; initialize $s2
iloop:  J       test
jloop:  lw      $s0, 0($s1)
        slt     $s0, $s0, $s2           ; p->value < i
        bnez    $s0, right
        lw      $s1, 4($s1)             ; p = p->left
        J       test
right:  lw      $s1, 8($s1)             ; p = p->right
test:   bnez    $s1, jloop
        addiu   $s2, $s2, -1            ; subtract 1
        bgtz    $s2, iloop
```

How well will your techniques work on this loop?

Hiding Memory Latency

- Name some techniques for hiding/tolerating memory latency?

- What characteristics of applications and architecture are required to make these techniques work?

Kunle Olukotun

Hash Table Code

Assumptions

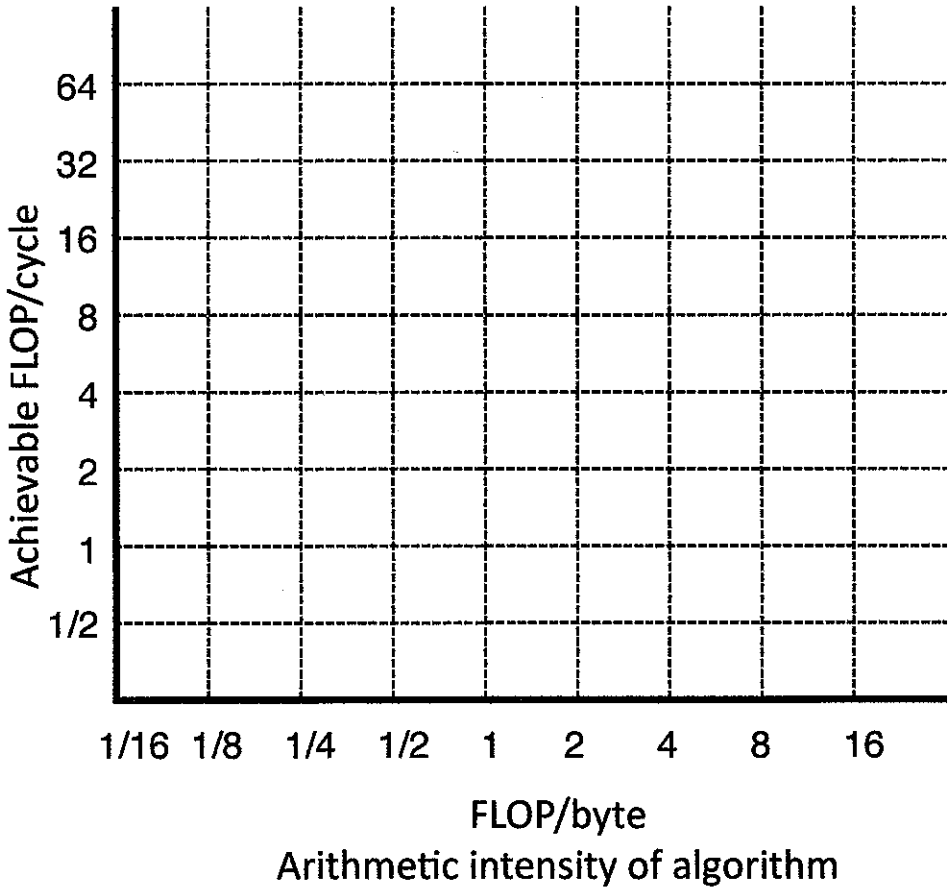
1. 1 KB F. A. data cache
2. 16 byte cache line
3. 100 cycle miss penalty
4. sizeof (Element) = 16 bytes
5. *Element = 8 bytes
6. N_ELEMENTS > 1024

```
1  Element element[N_ELEMENTS], *bucket[1024]
2  for (i = 0; i < N_ELEMENTS; i++)
   {
3    Element *ptrCurr, **ptrUpdate;
4    int hash_index;

     /* Find the location at which the new element is to be inserted. */
5    hash_index = element[i].value & 1023;
6    ptrUpdate = &bucket[hash_index];
7    ptrCurr = bucket[hash_index];
     /* Find the place in the chain to insert the new element. */
8    while (ptrCurr && ptrCurr->value <= element[i].value)
9    {
10     ptrUpdate = &ptrCurr->next;
11     ptrCurr = ptrCurr->next;
     }
     /* Update pointers to insert the new element into the chain. */
12   element[i].next = *ptrUpdate;
13   *ptrUpdate = &element[i];
   }
```

Explain how you would use architectural/software techniques to run this code as fast as possible

Performance Bounds

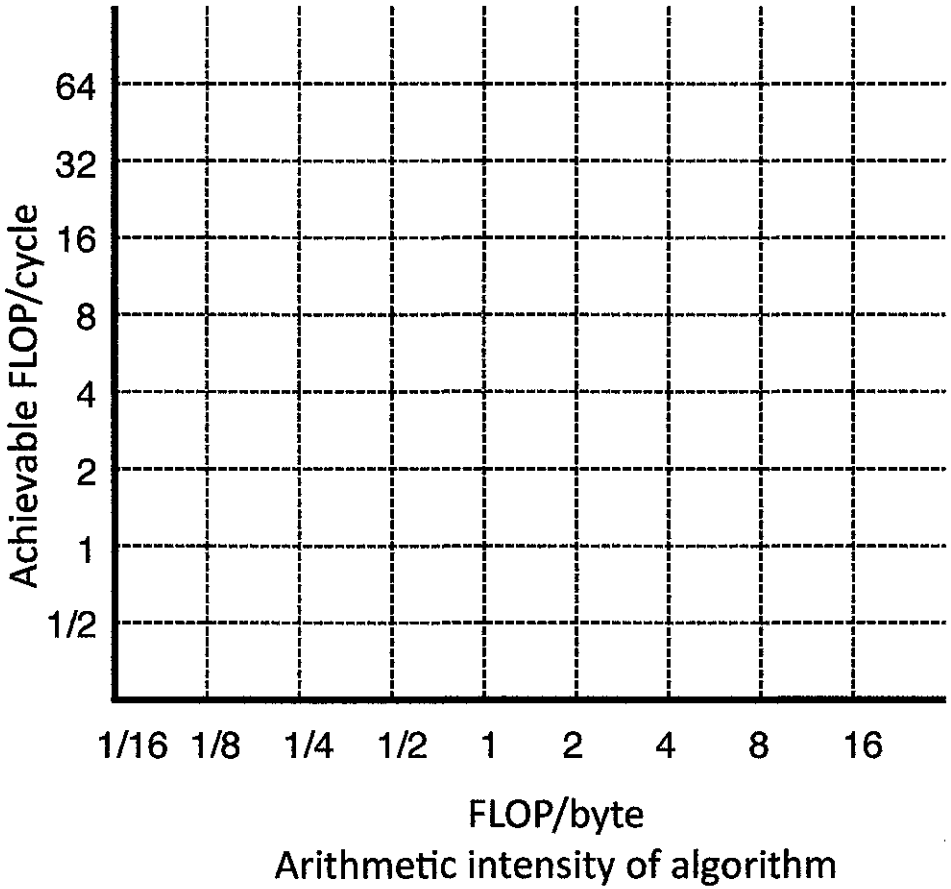


Given processor with:

1. 8 fully pipelined FP units
2. 8 byte/cycle memory BW
3. 4 byte floats

Plot the performance bounds on the graph

Performance Bounds



Given a processor with:

1. 8 fully pipelined FP units
2. 8 byte/cycle memory BW
3. 4 byte floats

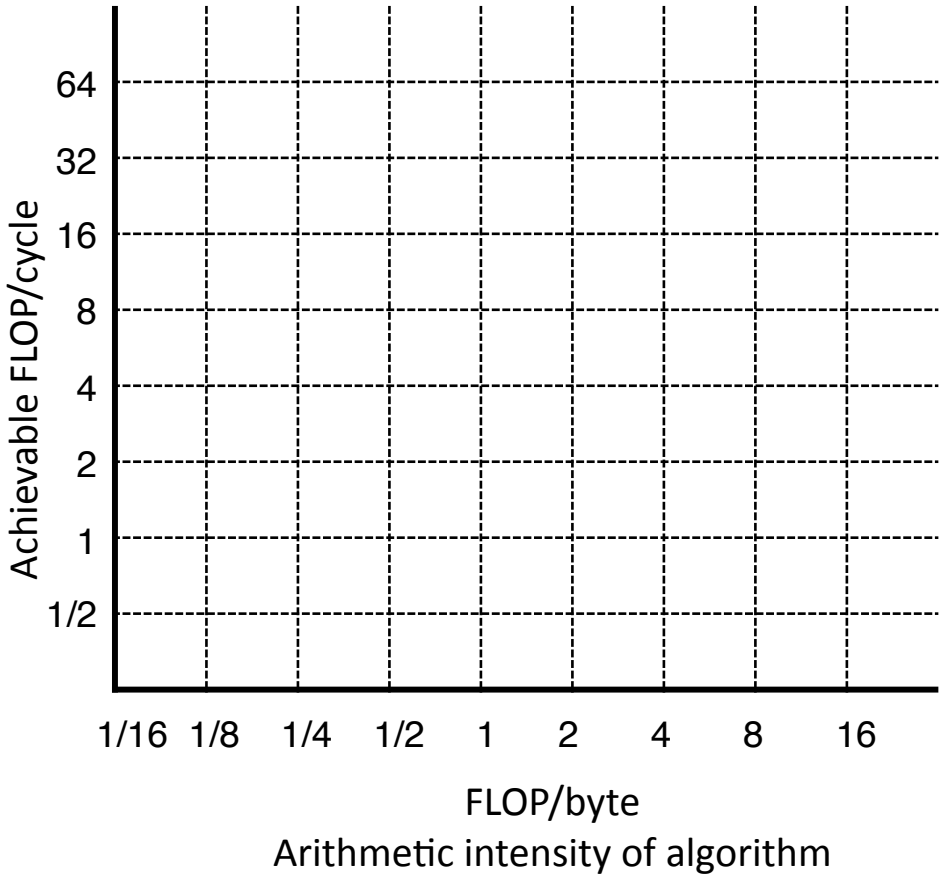
What's the performance bound on the SAXPY loop below?

1. X and Y are in main memory

2. C loop:
```
for (i=0; i < 100,000; i++)
        Y(i) = a*X(i) + Y(i);
```

```
foo:    LF      F2, 0 (R1) // load X(i)
        MULTF   F4, F2, F0 // multiply a*X(i)
        LF      F6, 0 (R2) // load Y(i)
        ADDF    F6, F4, F6 // add a*X(i) + Y(i)
        SF      0 (R2), F6 // store Y(i)
        ADDI    R1, R1, #4 // increment X index
        ADDI    R2, R2, #4 // increment Y index
        SGTI    R3, R1, #100000// test if done
        BEQZ    R3, foo    // loop if not done
```

Performance Bounds

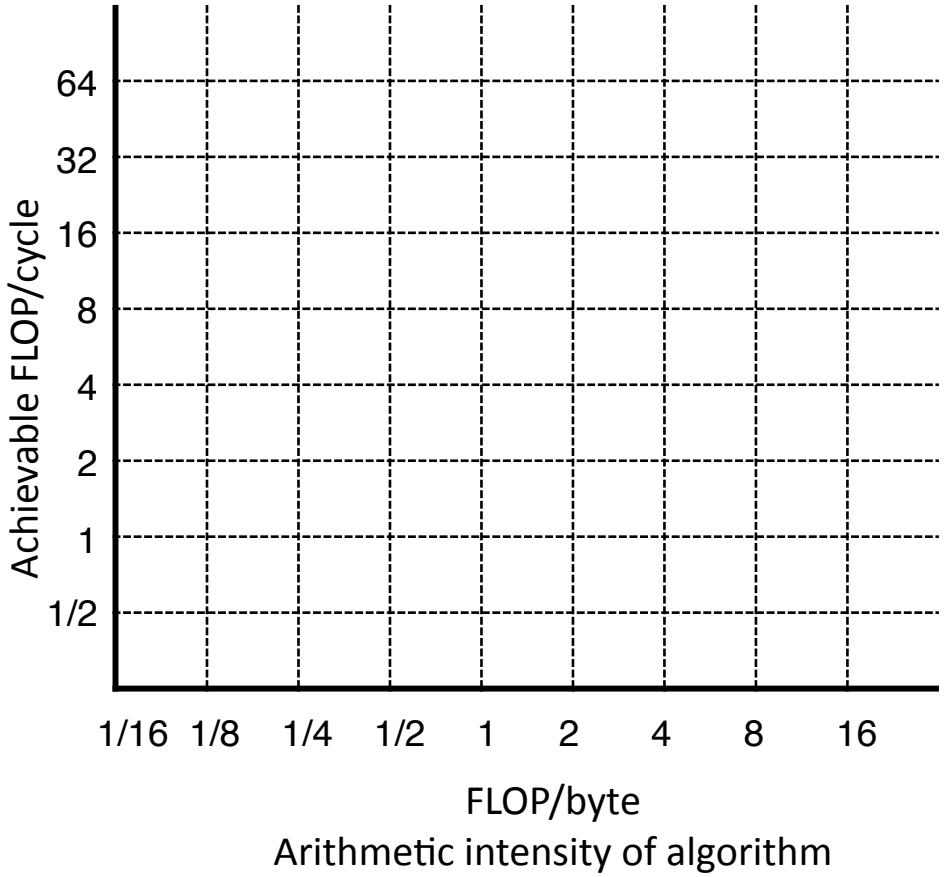


Given processor with:

1. 8 fully pipelined FP units
2. 8 byte/cycle memory BW
3. 4 byte floats

Plot the performance bounds on the graph

Performance Bounds



Given a processor with:

1. 8 fully pipelined FP units
2. 8 byte/cycle memory BW
3. 4 byte floats

What's the performance bound on the SAXPY loop below?

1. `X` and `Y` are in main memory

2. C loop:
```
for (i=0; i < 100,000; i++)
    Y(i) = a*X(i) + Y(i);
```

```
foo:    LF      F2, 0 (R1) // load X(i)
        MULTF   F4, F2, F0 // multiply a*X(i)
        LF      F6, 0 (R2) // load Y(i)
        ADDF    F6, F4, F6 // add a*X(i) + Y(i)
        SF      0 (R2), F6 // store Y(i)
        ADDI    R1, R1, #4 // increment X index
        ADDI    R2, R2, #4 // increment Y index
        SGTI    R3, R1, #100000// test if done
        BEQZ    R3, foo    // loop if not done
```

Computation on Meshes


```
for(i = 0; i < 100M; i++) {
     edge = edges_of_mesh[i]
   flux = flux_calc(edge)  /* millions of instructions */
   v0 = head(edge)
   v1 = tail(edge)
   Flux[v0] += flux
   Flux[v1] -= flux
}
```

1. What types of data locality exist in this loop?
2. How would you exploit this data locality? (two ways)
3. Instruction stream parallelism vs. data stream parallelism. What is benefit of each and which works here?
4. What are issues with data parallelism here?
5. Name three ways of dealing with them?