Digital Design & Computer Arch.

Discussion Session II

Prof. Onur Mutlu

ETH Zürich
Spring 2021
24 June 2021

Discussion Session

- Branch Prediction I (HW5, Q3)
- Systolic Arrays I (HW5, Q10)
- Vector Processing III (HW6, Q3)
- GPUs and SIMD I (HW6, Q6)
- GPUs and SIMD III (HW6, Q8)
- GPUs and SIMD IV (HW6, Q9)
- Reverse Engineering Caches II (HW7, Q3)
- Tracing the Cache (HW7, Q4)
- Cache Performance Analysis (HW7, Q7)
- Memory Hierarchy (HW7, Q8)
- Prefetching (HW7, Q12)

Branch Prediction I

(HW5, Q3)

Assume the following piece of code that iterates through a large array populated with **completely (i.e., truly) random** positive integers. The code has four branches (labeled B1, B2, B3, and B4). When we say that a branch is *taken*, we mean that the code *inside* the curly brackets is executed.

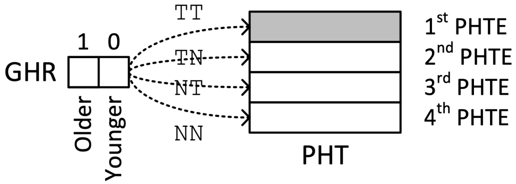
```
for (int i=0; i<N; i++) { /* B1 */
  val = array[i];       /* TAKEN PATH for B1 */
  if (val % 2 == 0) {   /* B2 */
    sum += val;       /* TAKEN PATH for B2 */
  }
  if (val % 3 == 0) {   /* B3 */
    sum += val;       /* TAKEN PATH for B3 */
  }
  if (val % 6 == 0) {   /* B4 */
    sum += val;       /* TAKEN PATH for B4 */
  }
}
```

a) Of the four branches, list all those that exhibit *local correlation*, if any.

b) Which of the four branches are *globally correlated*, if any? Explain in less than 20 words.

Now assume that the above piece of code is running on a processor that has a global branch predictor. The global branch predictor has the following characteristics.

- Global history register (GHR): 2 bits.
- Pattern history table (PHT): 4 entries.
- Pattern history table entry (PHTE): 11-bit signed saturating counter (possible values: -1024–1023)
- Before the code is run, all PHTEs are initially set to 0.
- As the code is being run, a PHTE is incremented (by one) whenever a branch that corresponds to that PHTE is taken, whereas a PHTE is decremented (by one) whenever a branch that corresponds to that PHTE is not taken.



c) After 120 iterations of the loop, calculate the *expected* value for only the first PHTE and fill it in the shaded box below. (Please write it as a base-10 value, rounded to the nearest one's digit.) *Hint: For a given iteration of the loop, first consider, what is the probability that both B1 and B2 are taken? Given that they are, what is the probability that B3 will increment or decrement the PHTE? Then consider…*

Systolic Arrays I

(HW5, Q10)

Figure 1 shows a systolic array processing element.

Each processing element takes in two inputs, M and N, and outputs P and Q. Each processing element also contains an "accumulator" R that can be read from and written to. The initial value of the "accumulator" is 0.

Figure 2 shows a systolic array composed of 9 processing elements. The smaller boxes are the inputs to the systolic array and the larger boxes are the processing elements. You will program this systolic array to perform the following calculation:

$$\begin{bmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

In each time cycle, each processing element will take in its two inputs, perform any necessary actions, and write on its outputs. The time cycle labels on the input boxes determine which time cycle the inputs will be fed into their corresponding processing elements. Any processing element input that is not driven will default to 0, and any processing element that has no output arrow will have its output ignored.

After all the calculations finish, each processing element's "accumulator" will hold one element of the final result matrix, arranged in the correct order.

a) Please describe the operations that each individual processing element performs, using mathematical equations and the variables M, N, P, Q and R.

b) Please fill in all 30 input boxes in Figure 2 so that the systolic array computes the correct matrix multiplication result described above. (Hint: Use a_{ij} and b_{ij}.)

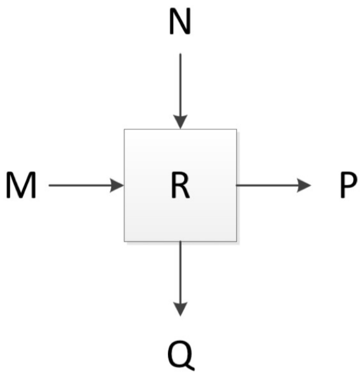


Figure 1: A systolic array processing element

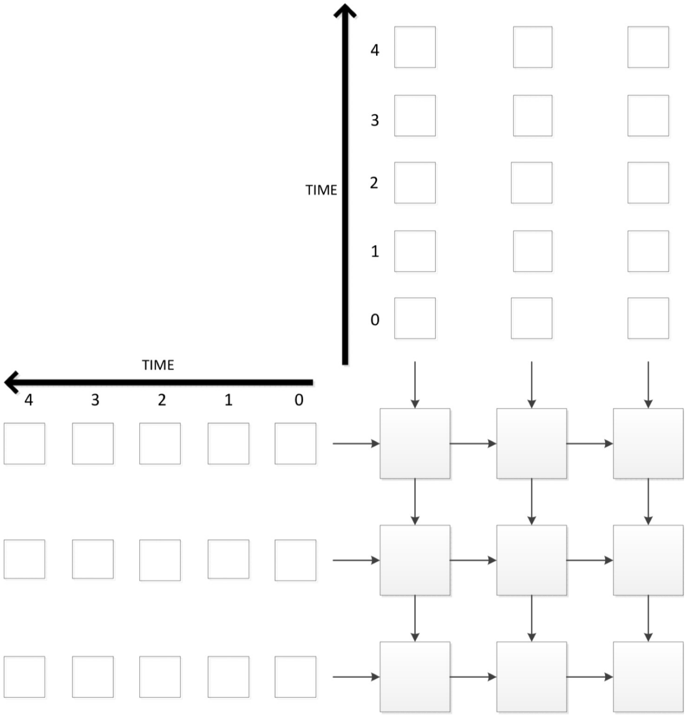


Figure 2: A systolic array

Vector Processing III (HW6, Q3)

Assume a vector processor that implements the following ISA:

Opcode	Operands	Number of cycles	Description
`SET`	`Vst, imm`	1	`Vst` ← `imm` (`Vst`: Vector Stride Register)
`SET`	`Vln, imm`	1	`Vln` ← `imm` (`Vln`: Vector Length Register)
`VLD`	`Vd, addr`	100, pipelined	`Vd` ← `MEM[addr]`
`VST`	`Vs, addr`	100, pipelined	`MEM[addr]` ← `Vs`
`VADD`	`Vd, Vs, Vt`	5, pipelined	`Vd` ← `Vs` + `Vt`
`VMUL`	`Vd, Vs, Vt`	10, pipelined	Vd$_i$ ← Vs$_i$ × Vt$_i$
`VDIV`	`Vd, Vs, Vt`	20, pipelined	Vd$_i$ ← Vs$_i$ / Vt$_i$

Assume the following:

- The processor has an in-order pipeline.
- The size of a vector element is 4 bytes.
- Vst and Vln are 10-bit registers.
- The processor does not support chaining between vector functional units.
- The main memory has N banks.
- Vector elements stored in consecutive memory addresses are interleaved between the memory banks. For example, if a vector element at address A maps to bank B, a vector element at address A + 4 maps to bank (B + 1)%N, where % is the modulo operator and N is the number of banks. N is not necessarily a power of two.
- The memory is byte addressable and the address space is represented using 32 bits.
- Vector elements are stored in memory in 4-byte-aligned manner.
- Each memory bank has a 4 KB row buffer.
- Each memory bank has a single read and a single write port so that a load and a store operation can be performed simultaneously.
- There are separate functional units for executing VLD and VST instructions.

a) What should the minimum value of N be to avoid stalls while executing a VLD or VST instruction, assuming a vector stride of 1? Explain.

b) What should the minimum value of N be to avoid stalls while executing a VLD or VST instruction, assuming a vector stride of 2? Explain.

c) Assume:

- A machine that has a memory with as many banks as you found is part (a).
- The vector stride is set to 1.
- The value of the vector length is set to M (but we do not know M)

The machine executes the following program:

```
VLD  V1, A                // V1 ← MEM[A]
VLD  V2, (A + 32768)      // V2 ← MEM[A + 32768]
VADD V3, V1, V1           // V3 ← V1, V1
VMUL V4, V2, V3           // V4_i ← V2_i, V3_i
VST  V4, (A + 32768*2)    // MEM[A + 32768*2] ← V4
```

It takes 4,306 cycles to execute the above program. What is M? Explain.

d) If we modify the vector processor to support chaining, how many cycles would be required to execute the same program in part (c)? Explain.

GPUs and SIMD I

(HW6, Q6)

We define the *SIMD utilization* of a program run on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of a program.

The following code segment is run on a GPU. Each thread executes **a single iteration** of the shown loop. Assume that the data values of the arrays A and B are already in vector registers so there are no loads and stores in this program. (Hint: Notice that there are 2 instructions in each thread.) A warp in the GPU consists of 32 threads, there are 32 SIMD lanes in the GPU. Assume that each instruction takes the same amount of time to execute.

```
for (i = 0; i < N; i++) {
    if (A[i] % 3 == 0) {      // Instruction 1
        A[i] = A[i] * B[i];   // Instruction 2
    }
}
```

a) How many warps does it take to execute this program? Please leave the answer in terms of N.

b) Assume integer arrays A have a repetitive pattern which have 24 ones followed by 8 zeros repetitively and integer arrays B have a different repetitive pattern which have 48 zeros followed by 64 ones. What is the SIMD utilization of this program?

c) Is it possible for this program to yield a SIMD utilization of 100%? Circle one: YES NO. If YES, what should be true about array A for the SIMD utilization to be 100%? What should be true about array B? If NO, explain why not.

d) Is it possible for this program to yield a SIMD utilization of 56.25%? Circle one: YES NO. If YES, what should be true about array A for the SIMD utilization to be 56.25%? What should be true about array B? If NO, explain why not.

e) Is it possible for this program to yield a SIMD utilization of 50%? Circle one: YES NO. If YES, what should be true about array A for the SIMD utilization to be 50%? What should be true about array B? If NO, explain why not.

Now, we will look at the technique we learned in class that tries to improve SIMD utilization by merging divergent branches together. The key idea of the *dynamic warp formation* is that threads in one warp can be swapped with threads in another warp as long as the swapped threads have access to the associated registers (i.e., they are on the same SIMD lane).

Consider the following example of a program that consists of 3 warps X, Y and Z that are executing the same code segment specified at the top of this question. Assume that the vector below specifies the direction of the branch of each thread within the warp. 1 means the branch in Instruction 1 is resolved to taken and 0 means the branch in Instruction 1 is resolved to not taken.

X = {10000000000000000000000000000010}
Y = {10000000000000000000000000000001}
Z = {01000000000000000000000000000000}

f) Given the example above. Suppose that you perform dynamic warp formation on these three warps. What is the resulting outcome of each branch for the newly formed warps X', Y' and Z'.

g) Given the specification for arrays A and B, is it possible for this program to yield a better SIMD utilization if dynamic warp formation is used? Explain your reasoning.

GPUs and SIMD III

(HW6, Q8)

We define the *SIMD utilization* of a program that runs on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of the program. As we saw in lecture and practice exercises, the SIMD utilization of a program is computed across the complete run of the program.

The following code segment is run on a GPU. Each thread executes **a single iteration** of the shown loop. Assume that the data values of the arrays A and B are already in vector registers, so there are no loads and stores in this program. (Hint: Notice that there are 3 instructions in each iteration.) A warp in the GPU consists of 32 threads, and there are 32 SIMD lanes in the GPU. Please answer the following six questions.

```
for (i = 0; i < 1025; i++) {
    if (A[i] < 33) {        // Instruction 1
        B[i] = A[i] << 1;   // Instruction 2
    }
    else {
        B[i] = A[i] >> 1;   // Instruction 3
    }
}
```

a) How many warps does it take to execute this program?

b) What is the maximum possible SIMD utilization of this program? (Hint: The warp scheduler does not issue instructions when no threads are active).

c) Please describe what needs to be true about array A to reach the maximum possible SIMD utilization asked in part (b). (Please cover all cases in your answer.)

d) What is the minimum possible SIMD utilization of this program?

e) Please describe what needs to be true about array A to reach the minimum possible SIMD utilization asked in part (d). (Please cover all cases in your answer.)

f) What is the SIMD utilization of this program if A[i] = i? Show your work.

GPUs and SIMD IV

(HW6, Q9)

We define the *SIMD utilization* of a program run on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of a program.

The following code segment is run on a GPU. Each thread executes **a single iteration** of the shown loop. Assume that the data values of the arrays A, B, and C are already in vector registers so there are no loads and stores in this program. (Hint: Notice that there are 6 instructions in each thread.) A warp in the GPU consists of 64 threads, and there are 64 SIMD lanes in the GPU.

```
for (i = 0; i < 4096; i++) {
    if (B[i] < 8888) {
        A[i] = A[i] * C[i];
        A[i] = A[i] + B[i];
        C[i] = B[i] + 1;
    }
    if (B[i] > 8888) {
        A[i] = A[i] * B[i];
    }
}
```

a) How many warps does it take to execute this program?

b) When we measure the SIMD utilization for this program with one input set, we find that it is 134/320. What can you say about arrays A, B, and C? Be precise (Hint: Look at the “if” branch).

c) Is it possible for this program to yield a SIMD utilization of 100%? Circle one: YES NO. If YES, what should be true about arrays A, B, C for the SIMD utilization to be 100%? Be precise. If NO, explain why not.

d) What is the lowest SIMD utilization that this program can yield? Explain.

Reverse Engineering Caches II (HW7, Q3)

You are trying to reverse-engineer the characteristics of a cache in a system, so that you can design a more efficient, machine-specific implementation of an algorithm you are working on. To do so, you have come up with three patterns that access various *bytes* in the system in an attempt to determine the following four cache characteristics:

- Cache block size (8, 16, 32, 64, or 128 B)
- Cache associativity (1-, 2-, 4-, or 8-way)
- Cache size (4 or 8KB)
- Cache replacement policy (LRU or FIFO)

However, the only statistic that you can collect on this system is *cache hit rate* after performing the access pattern. Here is what you observe:

Sequence	Addresses Accessed (Oldest → Youngest)								Hit Rate
1.	0	4	8	16	64	128			1/2
2.	31	8192	63	16384	4096	8192	64	16384	5/8
3.	32768	0	129	1024	3072	8192			1/3

Assume that the cache is initially empty at the beginning of the first sequence, but not at the beginning of the second and third sequences. The sequences are executed back-to-back, i.e., no other accesses take place between the three sequences. Thus, **at the beginning of the second (third) sequence, the contents are the same as at the end of the first (second) sequence.**

Based on what you observe, what are the following characteristics of the cache? Explain to get points.

a) Cache block size (8, 16, 32, 64, or 128 B)?

b) Cache associativity (1-, 2-, 4-, or 8-way)?

c) Cache size (4 or 8 KB)?

d) Cache replacement policy (LRU or FIFO)?

Tracing the Cache (HW7, Q4)

Assume you have three toy CPUs: 6808-D, 6808-T, and 6808-F. All three CPUs feature one level of cache. The cache size is 128 bytes, the cache block size is 32 bytes, and the cache uses LRU replacement. The only difference between the three CPUs is the associativity of the cache:

- 6808-D uses a direct mapped cache.
- 6808-T uses a two-way associative cache.
- 6808-F uses a fully associative cache.

You run the SPECMem3000 program to evaluate the CPUs. This benchmark program tests only memory read performance by issuing read requests to the cache. Assume that the cache is empty before you run the benchmark. The cache accesses generated by the program are as follows, in order of access from left to right:

A, B, A, H, B, G, H, H, A, E, H, D, H, G, C, C, G, C, A, B, H, D, E, C, C, B, A, D, E, F

Each letter represents a unique cache block. All 8 cache blocks are contiguous in memory. However, the ordering of the letters does not necessarily correspond to the ordering of the cache blocks in memory. For 6808-D, you observe the following cache misses in order of generation:

A, B, A, H, B, G, A, E, D, H, C, G, C, B, D, A, F

a) By using the above trace, please identify which cache blocks are in the same set for the 6808-D processor. Please be clear.

b) Please write down the sequence of cache misses for the 6808-F processor in their order of generation. (Hint: You might want to write down the cache state after each request).

c) For 6808-T, you observed the following five cache misses in order of generation: A, B, H, G, E. But, unfortunately, your evaluation setup broke before you could observe all cache misses for the 6808-T. Using the given information, which cache blocks are in the same set for the 6808-T processor?

d) Please write down the sequence of cache misses for the 6808-T processor in their order of generation.

e) What is the cache miss rate for each processor: 6808-D, 6808-T, 6808-F?

Cache Performance Analysis (HW7, Q7)

We are going to microbenchmark the cache hierarchy of a computer with the following two codes. The array data contains 32-bit unsigned integer values. For simplicity, we consider that accesses to the array latency bypass all caches (i.e., latency is not cached). timer() returns a timestamp in cycles.

```
(1) j = 0;
    for (i=0; i<size; i+=stride){
       start = timer();
       d = data[i];
       stop = timer();
       latency[j++] = stop - start;
    }
```

```
(2) for (i=0; i<size1; i+=stride1){
        d = data[i];
    }
    j = 0;
    for (i=0; i<size2; i+=stride2){
        start = timer();
        d = data[i];
        stop = timer();
        latency[j++] = stop - start;
    }
```

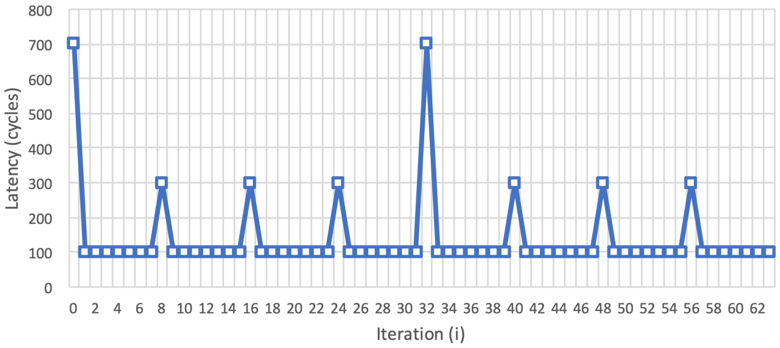
The cache hierarchy has two levels. L1 is a 4kB set associative cache.

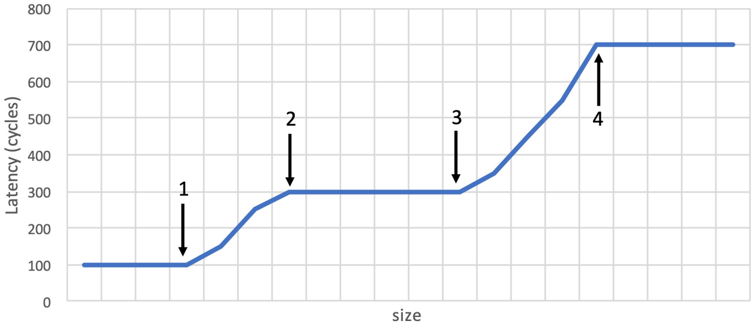
a) When we run code (1), we obtain the latency values in the following chart (upper right) for the first 64 reads to the array data (in the first 64 iterations of the loop) with stride equal to 1. What are the cache block sizes in L1 and L2?

b) Using code (2) with stride1 = stride2 = 32, size1 = 1056, and size2 = 1024, we observe latency[0] = 300 cycles. However, if size1 = 1024, latency[0] = 100 cycles. What is the maximum number of ways in L1? (Note: The replacement policy can be either FIFO or LRU).

c) We want to find out the exact replacement policy, assuming that the associativity is the maximum obtained in part (b). We first run code (2) with stride1 = 32, size1 = 1024, stride2 = 64, and size2 = 1056. Then (after resetting j), we run code (1) with stride = 32 and size = 1024. We observe latency[1] = 100 cycles. What is the replacement policy? Explain. (Hint: The replacement policy can be either FIFO or LRU. You need to find the correct one and explain).

d) Now we carry out two consecutive runs of code (1) for different values of size. In the first run, stride is equal to 1. In the second run, stride is equal to 16. We ignore the latency results of the first run, and average the latency results of the second run. We obtain the following graph (lower right). What do the four parts shown with the arrows represent?





Memory Hierarchy

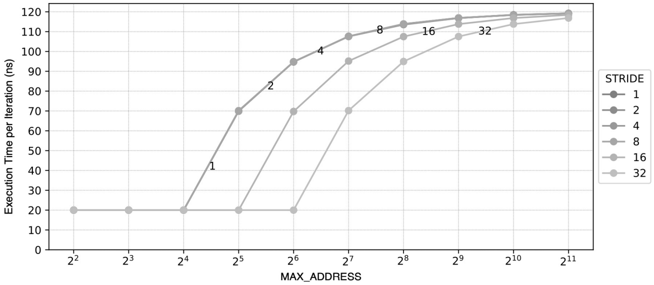
(HW7, Q8)

An enterprising computer architect is building a new machine for high-frequency stock trading and needs to choose a CPU. She will need to optimize her setup for *memory access latency* in order to gain a competitive edge in the market. She is considering two different prototype enthusiast CPUs that advertise high memory performance: (A) Dragonfire-980 Hyper-Z, (B) Peregrine G-Class XTreme

She needs to characterize these CPUs to select the best one, and she knows from Prof. Mutlu's course that she is capable of reverse-engineering everything she needs to know. Unfortunately, these CPUs are not yet publicly available, and their exact specifications are unavailable. Luckily, important documents were recently leaked, claiming that all three CPUs have:

- • Exactly 1 high-performance core
- • LRU replacement policies (for any set-associative caches)
- • Inclusive caching (i.e., data in a given cache level is present upward throughout the memory hierarchy. For example, if a cache line is present in L1, the cache line is also present in L2 and L3 if available.)
- • Constant-latency memory structures (i.e., an access to any part of a given memory structure takes the same amount of time)
- • Cache line, size, and associativity are all size aligned to powers of two
- Being an ingenious engineer, she devises the following simple application in order to extract all of the information she needs to know. The application uses a high-resolution timer to measure the amount of time it takes to read data from memory with a specific pattern parameterized by STRIDE and MAX ADDRESS :

```
start_timer()
repeat N times:
    memory_address <- random_data()
    READ[(memory_address * STRIDE ) % MAX_ADDRESS ]
end_timer()
```



*Assume 1) this code runs for a long time, so all memory structures are fully warmed up, i.e., repeatedly accessed data is already cached, and 2) N is large enough such that the timer captures **only** steady-state information.*

By sweeping STRIDE and MAX ADDRESS , the computer architect can glean information about the various memory structures in each CPU.

She produces Figure 1 for CPU A and Figure 2 for CPU B.

Your task: Using the data from the graphs, reverse-engineer the following system parameters. If the parameter does not make sense (e.g., L3 cache in a 2-cache system), mark the box with an "X" graphs provide insufficient information to ascertain a desired parameter, simply mark it as "N/A".

a) Fill in the blanks for Dragonfire-980 Hyper-Z.

b) Fill in the blanks for Peregrine G-Class XTreme.

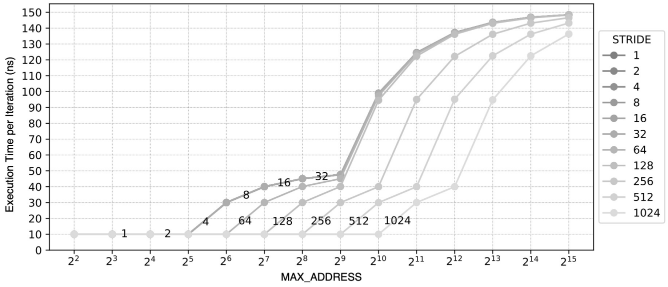


Figure 2: Execution time of the test code on CPU B for various values of *STRIDE* and *MAX_ADDRESS*. *STRIDE* values are labeled on curves themselves for clarity. Note that the curves for strides 1, 2, 4, 8, 16, and 32 overlap in the figure.

Prefetching

(HW7, Q12)

An architect is designing the prefetch engine for his machine. He first runs two applications A and B on the machine, with a stride prefetcher.

Application A:

```
uint8_t a[1000];
sum = 0;
for (i = 0; i < 1000; i += 4) {
    sum += a[i];
}
```

Application B:

```
uint8_t a[1000];
sum = 0;
for (i = 1; i < 1000; i *= 4) {
    sum += a[i];
}
```

i and *sum* are in registers, while the array a is in memory. A cache block is 4 bytes in size.

a) What is the prefetch accuracy and coverage for applications A and B using a stride prefetcher. This stride prefetcher detects the stride between two consecutive memory accesses and prefetches the cache block at this stride distance from the currently accessed block.

b) Suggest a prefetcher that would provide better accuracy and coverage for i) application A? ii) application B?

c) Would you suggest using runahead execution for i) application A. Why or why not? ii) application B. Why or why not?

Digital Design & Computer Arch.

Discussion Session II

Prof. Onur Mutlu

ETH Zürich
Spring 2021
24 June 2021