Problem 5 (1)

```c
int x[N];
for (register int k=0; k<K; k++) {
    for (register int i=0; i<N; i += stride) {  
        x[i] = x[i] + 1;  
    }
}
```

**Problem 5 (2)**

- **Stride = 2**, Reference pattern
  - \[x[0], x[2], x[4], x[6], \ldots\]
  - \[R \ W \ R \ W \ R \ W \ R \ W \]
  - \[M \ H \ H \ H \ H \ H \ M \ H \]

- Once cache is full
  - All cache lines are “dirty” (different than memory)
  - Need to write cache line to memory before replacing

- **Part c**
  - Approach 1
    - Insert explanation so that steps can be followed
  - Approach 2
    - Note that miss-hit and R-W patterns repeat after one cache line

- **Part c (Approach 2)**
  - \(T_1 = 1\) nsec
  - \(T_2 = (5+1+1+1) \times 5\) nsec = 40 nsec
  - **RWT**
    - Takes 2\(T_1\) to read and write one int in cache
    - Takes 2\(T_2\) to read and write one cache line
    - One cache line = 8 integers
    - \#reads (\#writes)/cache line = 8/S
      - Assuming write-back cache
    - \(RWT = 2T_1 + 2T_2/(8/S) = 2T_1 + S T_2/4 = 2 (1 + 5S)\)

- **Stride**
  - Effect of spatial locality at cache line scale
Problem 5 (4)

```c
int x[N];
for (register int k=0; k<K; k++) {
    for (register int i=0; i<N; i += stride) {
        x[i] = x[i] + 1; // Read; Write
    }
}
```

- Min EMAT (S=1) = 1 nsec (small x[]) or 11 nsec (large x[])
- T1 = 1 nsec; T2 = (5+1+1+1) x 5 nsec = 40 nsec (8 ints or 32 bytes)

**Array fits into cache (2 nsec):** Enough spatial locality

**High Temporal Locality (22 nsec):**

- 32-byte cache line (2+10S nsec) Some spatial locality

**Low Locality (162 nsec):**

Problem 6

- Memory subsystems are very complex
- Examples of spatial locality (different scale)
  - Many hits in a cache line
  - Hits confined to L1 cache ... to L1+L2 cache
- Example of temporal locality
  - Large stride means D() = csize/stride = small i.e., high temporal locality
  - Implies data will be referenced again very soon
- Raw data is in generic.out
- May need to run larger membench.c
- Some information
  - `/proc/cpuinfo`, `/proc/meminfo`
  - Manufacturer’s data sheet

Problem 6 (Larger membench)

- Plots have a logarithmic x-axis, not linear ➙
- R+W time versus Stride (bytes)
- 14 curves
  - 1 curve for each array size; vary stride
  - Array sizes are 4096 bytes, 8 KB, ... , 32 MB
  - Strides are 1 int (4 bytes), 2 ints, ... , (array size)/2
- May need to run several times to get stable results
  - Interference likely on CEC hosts since membench runs for atleast 1 sec per curve (especially grid.cec which runs VMware)
- Three regions in each curve for large arrays
  - **Small stride:** Cache line effect (hits after read)
  - **Very large stride:** Translation Lookaside Buffer (TLB)
  - **Large stride:** All misses
- Temporal Distance: D = csize/stride ➙
  - Small D ➔ High temporal locality

grid.cec (AMD opteron 244)

- Effect of VMware ???
  - Generic Read/Write (ns) Versus Stride
  - L1: 64KB/64KB, L2: 1MB, 64B cache line, TLB: 512/512
Tabular or graphical form shows data properties

Printing (printf, cout) can distort measurement !!!

gmtimeofday(2) uses a crude clock
  » Don’t expect to be able to measure tens of microseconds
  » Measure time for N calls and compute average
    * Avoid: gettimeofday(...); uname(...); gettimeofday(...);

tv_sec and tv_usec are long ints (32 bits)
  » Time in microseconds since the Epoch (Jan 1, 1970) is a number with atleast 16 digits ➔ Need more than 50 bits
    * ((t2.tv_sec*1000000+t2.tv_usec) - (t1.tv_sec*1000000+t1.tv_usec)) is too large for int ➔ May print as negative number

Quantify
  » “sys call takes 60 times longer than ...” is better than “sys call takes a lot longer than ...”

Be a skeptic: Do measurement results make sense?