Consider the following code snippet:

```
while(true)  {
    for (i=0;i<16;i++)
        y[i]=a[i]*b[i];
    z[0]=y[0];
    for (i=1;i<16;i++)
        z[i]=z[i-1]+y[i];
   }
```

For this example, only count add, multiply (alu) and memory operations (all array references). For simplicity assume looping, indexing is free.

1. Sequential execution time with $T_{mem} = 10$, $T_{alu} = 1$.

2. Bottleneck in sequential case? (compute or memory?)

3. Put z, y in small memory with $T_{smem} = 1$, a, b remain in large memory with $T_{lmem} = 10$, updated performance?

4. What parts are sequential? data parallel? reduce? parallel prefix?

5. Impact running on vector unit with vector length 4 (assume memory widened so a, b load vectorized as well)? New performance? Speedup over problem 3?

6. Amdahl's Law speedup (over problem 3) assuming infinite vector length?

7. Identify producer/consumer parallelism in example.

8. At what granularity can data be profitably passed between producer and consumer? (equivalently, how large does the buffer between producer and consumer need to be in order to allow them to run concurrently?)

9. Draw spatial pipeline for code taking one a[i], b[i] per cycle.

10. Critical path assuming associativity holds for addition?

11. VLIW with 1 load/store (single cycle load/store), 1 multiply, 1 add (treat y is intermediate that need not be stored to memory; assume need to store z, but can avoid needing to reread z[i-1] from memory.):
    (a) resource bound (per resources, overall)
    (b) critical path bound
    (c) schedule

12. VLIW with 4 load/store (single cycle load/store), 2 multiply, 2 add:
    (a) resource bound (per resources, overall)
    (b) critical path bound
    (c) schedule