EECE 276
Embedded Systems
Performance calculations and improvements
I/O Performance

- Key contributor to performance in many embedded systems: I/O
  - Disk I/O
  - Network I/O
  - External events I/O (e.g. radar tracks)
- Compute-bound vs. I/O bound systems
  - Bottleneck in computation or in I/O?
I/O Performance

- Basic buffer size calculation:
  - Production rate = \( P(t) \), consumption rate = \( C(t) \) (packets/sec)
  - If \( C \geq P \) then no buffer is needed.
  - If \( C \leq P \) for a (burst) period of time \( T \), then
    \[
    B = (P-C) \times T
    \]
    NOTE: This works only if the buffer can be emptied before the next burst.

- Variable buffer-size calculation:
  \[
  B(t_2) = \int_{t_1}^{t_2} [p(t) - c(t)] dt
  \]
  Burst period \( T = t_2 - t_1 \)
I/O Performance

● Example:

\[ p(t) = \begin{cases} 
10,000t & 0 \leq t \leq 1 \\
10,000(2-t) & 1 < t \leq 2 \\
0 & \text{elsewhere} 
\end{cases} \]

\[ c(t) = \begin{cases} 
10,000(1/4)t & 0 \leq t \leq 2 \\
10,000(1-1/4t) & 2 < t \leq 4 \\
0 & \text{elsewhere} 
\end{cases} \]

If burst period \( T = [0..1.6] \) seconds, then \( B(1.6) \) yields \( \sim 600 \) bytes

(Compute integral for \( 0..1 \) and \( 1..1.6 \), etc.)

● If the burst period is determined by \( u(t) \) (i.e. a real valued function), compute integral from \( t1 \) to \( u(t1) \).
Performance optimization

- Compute at the slowest rate (that is still acceptable)
- Use fixed-point numbers instead of floating point
  » Scaled integers: shift radix point
- Use lookup tables with interpolation for functions (instead of complex code)
- Allow imprecise computations (e.g. fewer samples, larger errors, etc.) if needed to trade for time.
Compiler optimization techniques

- Common sub-expression elimination
- Intrinsic functions
  - Macros, inlines
- Constant folding
  - Compact constant ops
- Loop invariants
  - Expressions in loops that don’t change
- Loop induction elimination
  - Value of loop variable changes systematically
- Use registers or caches
  - Move code into cache, size data s.t. it fits cache
Compiler optimization techniques

- Eliminate dead/unreachable code
- Control flow optimization
  - Fewer branches
- Constant propagation
  - Variable set to constant and then used
- Dead variable elimination
  - Variable’s value is discarded
- Short-circuiting boolean code
  - if (aa && bb) ..
- Loop unrolling
  - Expand/replicate loop code
- Loop jamming
  - Integrate similar, neighboring loops
- Cross-jump elimination
  - Same code in multiple switch cases