EECE 276
Embedded Systems
Performance analysis and optimization
Theory
Theory

- Performance analysis: crucial for real-time systems
- Scaling/complexity
  - How the algorithm behaves as the size of the problem changes
- NP-Completeness
  - NP-complete problems: the only known algorithms scale exponentially
  - Example: (general) scheduling
    - Online optimal scheduler for systems with mutexes
    - Multiprocessor scheduling (many variants)
- Undecidability – Halting problem
  - Certain classes of computational problems cannot be solved algorithmically
  - Example: Does a program P terminate if given input I?
“Laws” of parallelism

- **Amdahl’s Law:**
  
  » Processors = \( n \), sequential portion (\%) of the code = \( s \)

  \[
  \text{Speedup} = \frac{n}{1 + (n - 1)s}
  \]

  » If \( s = 0 \) : linear speedup as a function of processors
  » If \( s > 0 \) : perfect speedup is not possible
  » Underlying assumption: problem size is constant, and thus there is a diminishing margin of return for speeding up the computation.
  » In reality: problems tend to scale with the size of the parallel system
“Laws” of parallelism

- Gustafson’s Law:
  - Problems scale up with the number of processors
  - Estimate for serial portion: s + p*n
  
  \[ \text{Speedup} \approx n \]

- Parallel processors should work on different portions of data (data parallel processing)

- Parallel embedded systems
  - DSP networks – “fabric”
  - DSP pipelines
Performance analysis

- Code execution time estimation
  - Instruction counting:
    - Generate assembly, count cycles on the WCET path
    - Note: pipelining and caches may distort results
  - Instruction execution time simulators
  - Using the system clock
    - Measure execution time using system calls
    - Note clock resolution
  - Code profiling
    - Which part of the code was executed most frequently
Performance analysis

- Analysis of polled loops:
  » Delays = HW delay + test flag(f) + process event(P)
  » Problem: overlapping events
    – Time for processing n-th overlapping event: nfP
    – Avoid overlapping events

- Analysis of co-routines
  » Explicit “yield” – cooperative scheduling
  » Trace WCET path through each task
Performance analysis

- Analysis of round-robin systems:
  - Cyclic execution of tasks, round-robin
  - Tasks=\( n \), Max. exec time=\( c \) (over all tasks), timeslice=\( q \).
  - Each task gets \( 1/n \) of the CPU time
  - Each task waits max. \( (n-1)*q \) time units.
  - Each task requires at most \( \lceil c/q \rceil \) units
  - The waiting time is: \( (n-1)*q*\lceil c/q \rceil \).
  - Worst-case time from readiness to completion:
    \[
    T = (n-1)*q*\lceil c/q \rceil + c
    \]
  - With context switching overhead \( o \):
    \[
    T = ((n-1)*q + n*o)*\lceil c/q \rceil + c
    \]
Performance analysis

- Analysis of fixed period systems:
  - For highest priority task: w.c.r.t = exec. time
  - For all other tasks:
    - $R_k = e_k + l_k$ (exec. time + max. delay caused by other tasks)
  - Consider task $t_m$ (higher priority than $t_k$):
    - Within $[0,R_k)$: release of $t_m$ occurs $\text{ceil}(R_k/p_m)$ times.
    - Maximum interference caused by $t_m$: $\text{ceil}(R_k/p_m)*e_m$
    - All interference with $t_k$: $I_k = \sum(m) \text{ceil}(R_k/p_m)*e_m$, where sum is over all the tasks with higher priority than $t_k$
    - Response time can be calculated as the solution to the recurrence relation:
      \[
      R_i^{n+1} = e_i + \sum_{j \in HP(i)} \text{ceil}(R_i^n / p_j)*e_j
      \]
      Solution: Iterate until a fix-point is reached.
Performance analysis

- Analysis of fixed-period systems:
  » RMA Example:

<table>
<thead>
<tr>
<th>t_i</th>
<th>e_i</th>
<th>p_i</th>
<th>R_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1</td>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>t_2</td>
<td>4</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>t_3</td>
<td>2</td>
<td>18</td>
<td>9</td>
</tr>
</tbody>
</table>

  Highest priority
  Start with 4, iterate
  Start with 2, iterate

- Sporadic/aperiodic interrupt systems
  » Calculate interrupt latency via counting
  » Consider (macro)instruction finish times