Select the speed based on worst-case execution time, WCET, and deadline.

Dynamic Speed adjustment techniques for linear code

Speed adjustment based on remaining WCET

Note: a task very rarely consumes its estimated worst case execution time.
Dynamic Speed adjustment techniques for linear code

Speed adjustment based on remaining WCET

Greedy speed adjustment:
Give reclaimed slack to next task rather than all future tasks
Dynamic Speed adjustment techniques for linear code

Speed adjustment based on remaining average execution time

An alternate point of view

Reclaimed slack

stolen slack
Dynamic Speed adjustment techniques for non-linear code

- Remaining WCET is based on the longest path
- Remaining average case execution time is based on the branching probabilities (from trace information).

2. Periodic, non-frame-based systems

- Each task has a WCET, \( C_i \) and a period \( T_i \)
- Earliest Deadline First (EDF) scheduling

- **Static speed adjustment**: If utilization \( U = \sum \frac{C_i}{T_i} < 1 \), then we can reduce the speed by a factor of \( U \), and still guarantee that deadlines are met.

Note: Average utilization, \( U_{av} \) can be much less than \( U \)
Greedy dynamic speed adjustment

- Giving reclaimed slack to the next ready task is not always a correct scheme.

Theorem: A reclaimed slack has to be associated with a deadline and can be given safely to a task with an earlier or equal deadline.

greedy dynamic speed adjustment

- Theorem: if tasks 1, ..., k are ready and will complete before the next task arrival, then we can swap the time allocation of the k tasks. That is we can add stolen slack to the reclaimed slack

Experimental rule: Do not be very aggressive and reduce the speed of a task below a certain speed (the optimal speed determined by $U_{av}$).
**Speed adjustment in Multi-processors**

1. the case of independent tasks on two processors

   Canonical execution ==> all tasks consume WCET

<table>
<thead>
<tr>
<th>Global queue</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   Deadline
   time

   P1  P2

<table>
<thead>
<tr>
<th>No speed management</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Static speed management</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   Non canonical execution ==> tasks consume ACET

   If we select the initial speed based on WCET, can we do dynamic speed adjustment and still meet the deadline?

   | 9,3 | 6,5 | 6,6 | 3,3 |

   P1  P2

<table>
<thead>
<tr>
<th>Greedy slack Reclamation (GSR)</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Slack sharing (SS)</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   deadline miss
   time

   deadline
   meets deadline

   deadline
2. dependent tasks

Use list scheduling

Ready_Q 4 3 6 6
Canonical execution
P1 2 4 6
P2 3 3 6

time

- Assuming that we adjust the speed statically such that canonical execution meets the deadline.
- Can we reclaim unused slack dynamically and still meet the deadline?

Use list scheduling

Ready_Q 4 3 6 6
Canonical execution
P1 2 4 6
P2 3 3 6

Non-canonical Execution with Slack sharing
P1 2 4 3
P2 1 6 6

deadline miss

time
Dynamic speed adjustment (2 processors)

**Solution:** Use a wait_Q to enforce canonical order in Ready_Q

- A task is put in Wait_Q when its last predecessor starts execution
- Tasks are ordered in Wait_Q by their expected start time under WCET
- Only the head of the Wait_Q can move to the Ready_Q
Dynamic speed adjustment (2 processors)

**Solution:** Use a wait_Q to enforce canonical order in Ready_Q

- A task is put in Wait_Q when its last predecessor starts execution
- Tasks are ordered in Wait_Q by their expected start time under WCET
- Only the head of the Wait_Q can move to the Ready_Q

![Diagram showing task execution order and wait/ready queues]

Theoretical results

For **independent tasks**, if canonical execution finishes at time $T$, then non-canonical execution with *slack sharing* finishes at or before time $T$.

For **dependent tasks**, if canonical execution finishes at time $T$, then, non-canonical execution with *slack sharing and a wait queue* finishes at or before time $T$.

**Implication:**
- Can optimize energy based on WCET (static speed adjustment)
- At run time, can use reclaimed slack to further reduce energy (dynamic speed adjustment), while still guaranteeing deadlines.
4. Static optimization when different tasks consume different power

Assuming that the power consumption functions are identical for all tasks.
Then to minimize the total energy, all tasks have to execute at the same speed.

If, however, the power functions, \( P_i(S) \), are different for tasks \( i = 1, \ldots, n \),
Then using the same speed for all tasks does not minimize energy consumption.

Let \( C_i \) = number of cycles needed to complete task \( i \)

Minimizing energy consumption

Example: Three tasks with \( C_1 = C_2 = C_3 \)
If \( P_i(S) = a_i S^2 \), for task \( i \),
then, energy consumed by task \( i \) is \( E_i = a_i / t_i \).
If \( a_1 = a_2 = a_3 \),
then \( t_1 = t_2 = t_3 \) minimizes total energy.

\[
\begin{array}{c}
\text{Start time} \\
\text{Task 1} \\
\text{Task 2} \\
\text{Task 3} \\
\text{deadline}
\end{array}
\]

\[
\begin{array}{c}
\text{Start time} \\
\text{Task 1} \\
\text{Task 2} \\
\text{Task 3} \\
\text{deadline}
\end{array}
\]

\[
\begin{array}{c}
\text{Start time} \\
\text{Task 1} \\
\text{Task 2} \\
\text{Task 3} \\
\text{deadline}
\end{array}
\]
Minimizing energy consumption

Example: Three tasks with $C_1 = C_2 = C_3$

- If $P_i(S) = a_i S^2$, for task $i$,
  - then, energy consumed by task $i$ is $E_i = a_i / t_i$

If $a_1$, $a_2$ and $a_3$ are different

then $t_1 = t_2 = t_3$ does not minimize total energy

The problem is to find $S_i$, $i=1, \ldots, n$, such that to

- minimize $\sum_{i=1}^{n} t_i P_i(S_i)$
- subject to $\sum_{i=1}^{n} t_i \leq D$
- and $S_{\text{min}} \leq S_i \leq S_{\text{max}}$

Note that $t_i = \frac{C_i}{S_i}$

- We solved this optimization problem, consequently developing a solution for arbitrary convex power functions.
- Algorithm complexity: $O(n^2 \log n)$
Maximizing the system's reward

**General problem assumptions:**
- tasks have different power/speed functions
- tasks have different rewards as functions of number of executed cycles

---

**Theorem:** If power functions are all quadratic or cubic in $S$, then reward is maximum when power is the same for all tasks.

---

Given the speeds, we know how to maximize the total reward while meeting the deadline.
The problem:
find the speeds \( S_i \), and execution times, \( t_i \), \( i = 1, \ldots, n \), such as to

\[
\text{maximize } \sum_{i=1}^{n} R(C_i)
\]

subject to

\[
\sum_{i=1}^{n} t_i \leq D
\]

\[
\sum_{i=1}^{n} t_i P(t_i) \leq E
\]

\[
S_{\text{min}} \leq S_i \leq S_{\text{max}}
\]

\[
l_i \leq C_i \leq u_i
\]

Note that \( t_i = \frac{C_i}{S_i} \)

We solved this optimization problem analytically for specific forms of the power functions. For arbitrary power functions, we use heuristics.

1) Ignore energy and maximize reward, \( R \), within the deadline

2) If exceed available energy;
   - remove \( \Delta t \) from a task such that decrease in \( R \) is minimal
   - use \( \Delta t \) to decrease the speed of a task, such as to maximize the decrease in energy consumption

3) Repeat step 2 until the energy constraints are satisfied
An iterative refinement algorithm:
1) Define a required error, $\epsilon$
2) Solve the problem with an initial $\Delta t$
3) Set $\Delta t = \Delta t / 2$
4) Solve the problem for the new $\Delta t$
5) Find the improvement in the solution
6) If improvement larger than $\epsilon$, go to 3

Dual use of time slack

Slack can be used for
1) Fault tolerance
   - Add checkpoints
   - Reserve recovery time
2) Reduce processing speed
**Dual use of time redundancy**

**Observation:** May continue executing at $S_{\text{max}}$ after recovery.

**Disadvantage:** consumes more energy when a fault occurs (rare)

**Advantage:** recovery in an early section can use slack created by execution of later sections at $S_{\text{max}}$

Motivates non-uniform checkpoints.

---

**Non-uniform check-pointing**

**Can save more power by using non-uniformly spaced checkpoints.**

**Placement of checkpoints is different from placement of the PMPs.**
**Optimal number of checkpoints**

More checkpoints = more overhead + less recovery slack

For a given
- slack \( C/D \) and
- checkpoint overhead \( \rho \),
we can find the number of checkpoints that
- minimizes energy consumption, and
- guarantee recovery and timeliness.

**Triple Modular Redundancy Vs. Duplex**

\( \rho \): overhead of checkpoint  

\( \text{Load} \): slack in the system  

\( \pi \): ratio of static/dynamic power

Energy efficiency of TMR Vs. Duplex depends on \( \rho, \pi \), and \( \text{load} \)
Including the hardware cost

Assuming the **same number of processors** and a given task set, to obtain the **same reliability**, will it be more energy efficient to use TMR or Duplex?

**Example:** 6 processors and 6 identical tasks.

Need to look at
- the fault model,
- the energy model and
- the overheads.
- the communication cost

Reconfigurable clusters of servers

Reconfigure your cluster to
- achieve required reliability
- minimize energy consumption
- optimize performance
- meet quality of service

**Simplex mode**

**Duplex mode**

**TMR mode**

**Inactive processors**

**Example:**
- At a given server’s speed, it is more energy efficient to activate an additional server than to increase the speed of the active servers.
- At a given server’s speed, it is more energy efficient to power down one of the servers than to reduce the speed of the active servers