Dynamic Slack Reclamation with Procrastination Scheduling in Real-Time Embedded Systems

Paper by Ravindra R. Jejurikar and Rajesh Gupta

Presentation by Tyler Bletsch
NC State University
19 October 2005

Introduction

- Must reduce energy usage!
- Two ways to save power
  - Slowdown (DVS)
    - Reduce dynamic power, increase execution time, static power unaffected
  - Shutdown
    - Turn (almost) all power off for a given period of time
    - We can use *task procrastination* to glob together slack times
- Static power usage is growing
  - This is due to increasing leakage current in newer & smaller processors, so slowdown isn't enough...
- Paper's goal:
  - Combine procrastination scheduling with dynamic slowdown techniques

System Model

- Tasks τ_i of form $\{T_i, D_i, C_i\}$
  - T_i = Period, D_i = Relative deadline, C_i = WCET
- Slowdown
  - η = DVS slowdown factor in [0,1]
  - η_i = Static slowdown of task i
  - η_{crit} = Slowdown with least energy per clock cycle
    - The minimum value of η worth caring about
- Dynamic Slack Reclamation's two parts:
  - Slack Reclamation Algorithm
    - Generic mechanism for all procrastination/slowdown hybrids
  - Slack Distribution Policy
    - Specific policy to choose how much slack goes to procrastination versus slowdown

Variables & Structures Used

- J_i : current job of task τ_i
- $R^r_i(t)$: available run-time of J_i at time t
- $R^F_i(t)$: free time (slack) available to J_i at time t
  - Run-time from the FRT-list with priority $\geq P(J_i)$
- $C^r_i(t)$: residual workload of job J_i
- $R^{crit}_i(t)$: run-time needed to finish J_i at speed η_{crit}
- Z_i : Statically derived procrastination delay
- Z^D_i : Dynamically derived procrastination delay
- *FRT-list* : Free Run Time List, a priority sorted list of available runtime from processes' slack

Algorithm 1: Slack Reclamation

1: **On arrival of a new job J_i:** $\{J_i$ **is an instance of task** $\tau_i\}$
2: $R^r_i(t) \leftarrow \frac{C_i}{\eta_i}$;
3: **Add job J_i to scheduler Ready Queue;**
4: **if (processor is in sleep state) then**
5: **Set Z^D_i to any number in the range $[0, max(Z_i, R^F_i(t))]$;**
6: **if (Timer is not active) then**
7: $timer \leftarrow Z^D_i$ **{Initialize timer}**
8: **else**
9: $timer \leftarrow min(timer, Z^D_i)$;
10: **end if**
11: **end if**

12: **On execution of each job J_i :**
13: **setSpeed** $(max(\eta_{crit}, \frac{C^r_i(t)}{R^r_i(t)+R^F_i(t)}))$;

14: **On completion of job J_i :**
15: **Add to FRT-list**$(R^r_i(t), \mathcal{P}(J_i))$;

Algorithm 2 specifies this

16: **On expiration of Timer** $(timer = 0)$**:**
17: **Wakeup Processor;**
18: **Scheduler schedules highest priority task;**
19: **Deactivate timer;**

Theorem 1: All tasks meet deadlines using this model, see [5] for proof.

Algorithm 2: Slack Distribution

- Replace line 5 of algorithm 1 with this:

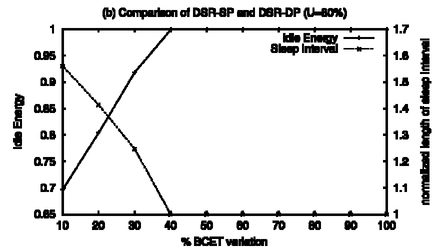
3: **if** $(R^F_i(t) + R^r_i(t) < R^{crit}_i(t))$ **then**
4: $Z^E_i \leftarrow 0$;
5: **else**
6: $Z^E_i \leftarrow R^F_i(t) + R^r_i(t) - R^{crit}_i(t)$; **{Note that** $Z^E_i \leq R^F_i(t)$**}**
7: **end if**
8: $Z^D_i \leftarrow max(Z_i, Z^E_i)$;

Theorem 2: All tasks meet deadlines using this model, follows from Theorem 1.

Experiment

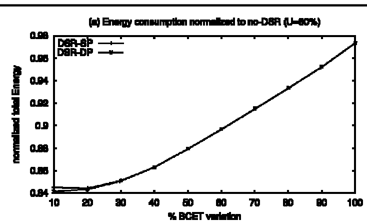
- Three algorithms tested:
  - **no-DSR**: Static slowdown (η_i) with static procrastination intervals (Z_i)
  - **DSP-SP**: Dynamic slowdown (algorithm 1) with static procrastination (Z_i)
  - **DSP-DP**: Dynamic slowdown (algorithm 1) with dynamic procrastination (algorithm 2)

Results (1)



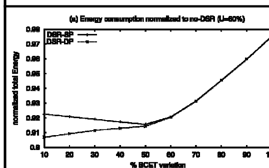
- DSR-DP normalized to DSP-SP, effect on sleep periods and idle energy, U=80%
- For BCET variation ≤ 30%, sleep intervals are affected

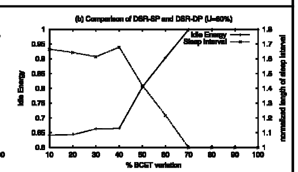
Results (2)



- DSR-DP & DSP-SP energy normalized to no-DSR levels, U=80%
- A savings of DSR-DP over DSR-SP for BCETvar ≤ 30% (the two are the same for BCETvar > 30%)

Results (3)





- Same for U=60% at BCETvar ≤ 60%
- Watch the axes! Both algorithms save ***less*** overall than in U=80%
- Also, same for:
  - U=50% at BCETvar ≤ 70%
  - U=40% at BCETvar ≤ 80%

Some points

- When U < η_{crit} (0.41 in the experiment):
  - "Static procrastination intervals dominate over dynamic slack available"
  - With nothing left to scavenge, DSR-DP does very little for these cases
- Overall, DSR-DP isn't a huge win over DSR-SP, because static procrastination already globs the majority of small idle times
- However, when these statically derived times are too short to shut down, the small boost given by DSR-DP could put them over the limit, and thus mean significant savings

Conclusion

- Slowdown reduces dynamic power, but static power is becoming the problem in modern processors
- Shutting down allows us to cut off all power for a time
- Task procrastination works to lengthen idle times in which we can shut down
- The paper combines these two existing methods to get the best of each
- Idle energy savings of up to 70% are realized
- This savings will become more important as static power use increases in future chip designs

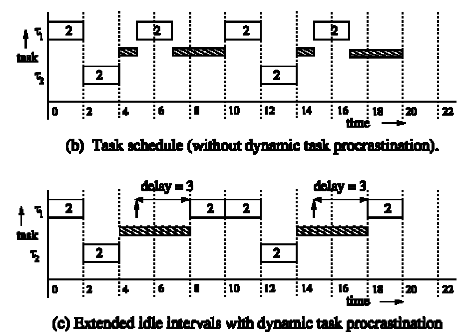
Any questions?

Additional reference slides follow...

Dynamic Task Procrastination (1)

- On task completion, if no tasks left to execute:
  - Shut down
- If a task arrives and we are shut down, then
  - Find the max time Z^D_i that we can wait and still finish all tasks on time based on WCET
  - Wake up the processor before the least Z^D_i
  - Start processor and run EDF normally

Dynamic Task Procrastination (2)

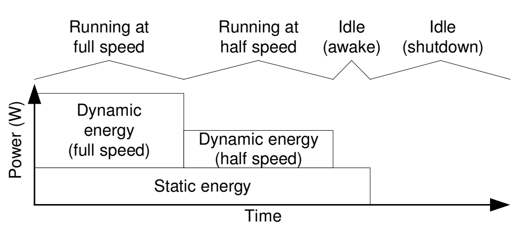


(b) Task schedule (without dynamic task procrastination).

(c) Extended idle intervals with dynamic task procrastination

Power Usage Overview

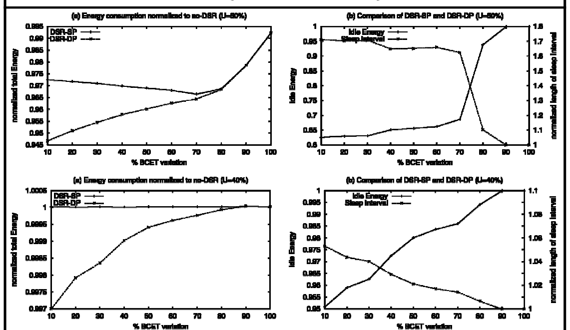
- The effect of slowdown and shutdown on power and energy:



Run-time Consumption

- A task τ_i consumes run-time in wall-clock seconds (i.e. η isn't involved in this calculation)
- If $R^F_i(t) > 0$, the run-time is taken from the *FRT-list*, else it uses its allotted run-time
- During idle periods, time is used from the *FRT-list* unless the list is empty
- These rules can be applied at job arrival & completion (rather than continuously)

Results for U={50%,40%}



- Same for U=50% at BCETvar ≤ 70%, U=40% at BCETvar ≤ 80%