Thermally Optimal Stop-Go Scheduling of Task Graphs with Real-Time Constraints

Pratyush Kumar **Lothar Thiele**

Computer Engineering and Networks Laboratory

ETH Zürich, Switzerland

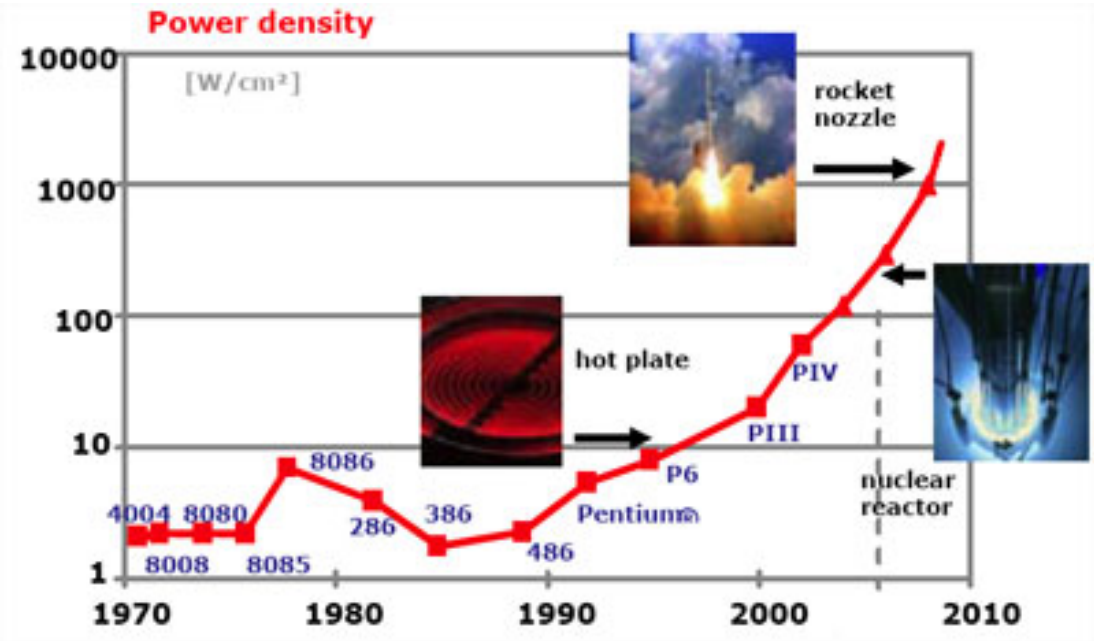
ASP-DAC 2011, Yokohama,

26th January, 2011

Outline

The heat is on!

- *Exponential* rise in processor power densities



[Source: Intel]

- Consequentially *high on-chip temperatures*
- Hot chips can cause short-term *functional errors* and long-term *reliability degradation*

What are the solutions?

1. *Better VLSI design*

   Thermal-aware design, new materials ...

2. *Hardware cooling solutions*

   Use better heat sinks, fans, air cooling (or even water cooling)

What are the solutions?

1. *Better VLSI design*

   Thermal-aware design, new materials ...

2. *Hardware cooling solutions*

   Use better heat sinks, fans, air cooling (or even water cooling)

3. *Voltage (and frequency) scaling (DVS)*

   When temperatures are too high, reduce supply voltage V_{DD} (and slow down the processor)

4. *Stop-go execution*

   When temperatures are too high, completely turn off the processor and peripherals

Cool real-time

- Performance degrades with DVS or stop-go execution
- For real-time applications it is important to ensure that tasks *do not miss deadlines* when such techniques are used
- Clearly, this is a case for *co-design*: thermal and real-time objectives must be considered simultaneously
- This is the focus of this work

Limitations in existing work

- Existing work considers only *sporadic tasks* with individual deadlines.
  - However real-life applications are usually expressed as *task graphs*: a set of tasks constrained by precedence constraints.
  - Such applications only have a consolidated deadline, with greater scheduling freedom.

Limitations in existing work

- Existing work considers only *sporadic tasks* with individual deadlines.
  - However real-life applications are usually expressed as *task graphs*: a set of tasks constrained by precedence constraints.
  - Such applications only have a consolidated deadline, with greater scheduling freedom.
- Existing work exclusively focuses on DVS
  - DVS is becoming less important as the headroom for voltage scaling reduces
  - DVS cannot be applied to other I/O components like radio, network bus, memory controller, etc.

Limitations in existing work

- Existing work considers only *sporadic tasks* with individual deadlines.
  - However real-life applications are usually expressed as *task graphs*: a set of tasks constrained by precedence constraints.
  - Such applications only have a consolidated deadline, with greater scheduling freedom.
- Existing work exclusively focuses on DVS
  - DVS is becoming less important as the headroom for voltage scaling reduces
  - DVS cannot be applied to other I/O components like radio, network bus, memory controller, etc.
- We address these two issues in this work

Outline

System Model

- We have a system that can exist in two modes: *active* and *idle*
- In active mode the system consumes some power and processes tasks at some rate
- In idle mode the system consumes a lower amount of power and processors no tasks

System Model

- We have a system that can exist in two modes: *active* and *idle*
- In active mode the system consumes some power and processes tasks at some rate
- In idle mode the system consumes a lower amount of power and processors no tasks
- The system cannot be put in the idle mode while running a task (requires context save and restore)
- Only control: Put the system in idle mode in between task executions

Thermal Model

- Heat dissipation is modelled using the *compact thermal model*
- From the compact model parameters, we can derive the following closed-loop solution to the temperature of the system

$$T(t) = T^{\infty} + (T(t_o) - T^{\infty}) \cdot e^{-a(t-t_o)}$$

where t_o is the starting time, T^{∞} is the steady-state temperature and a is the time-constant.

Thermal Model

- Heat dissipation is modelled using the *compact thermal model*
- From the compact model parameters, we can derive the following closed-loop solution to the temperature of the system

$$T(t) = T^{\infty} + (T(t_o) - T^{\infty}) \cdot e^{-a(t-t_o)}$$

  where t_o is the starting time, T^{∞} is the steady-state temperature and a is the time-constant.
- The power consumption is mode-dependent and thus the parameters T^{∞} and a are also mode-dependent. We use super-scripts to denote the modes
- The thermal properties of the system is given in the tuple $\mathbf{T} = \{T^{act}, T^{idl}, a^{act}, a^{idl}\}$

Application Model

- Application is modeled as a task graph $G = (V, E)$
- Each $v \in V$ is a task with an execution time τ_v
- Each edge $e = (v_1, v_2)$ denotes that the task v_1 must complete before the task v_2 starts
- It is required that all tasks of the task graph complete execution with ω_{max} units of time
- The application is characterized by $\mathbf{A} = \{G, \tau, \omega_{max}\}$

Scheduling problem

- A stop-go schedule of a task graph is characterized by the tuple $\mathbf{S} = \{\sigma, \mathbf{t}^{idl}\}$
- σ is the static-ordering of tasks: σ_i is the ith task to be executed
- $\mathbf{t}^{idl}$ is the amount of idle times inserted: t_j^{idl} is the idle time inserted before the execution of the jth task

Scheduling problem

- A stop-go schedule of a task graph is characterized by the tuple $\mathbf{S} = \{\sigma, \mathbf{t}^{idl}\}$
- σ is the static-ordering of tasks: σ_i is the ith task to be executed
- $\mathbf{t}^{idl}$ is the amount of idle times inserted: t_j^{idl} is the idle time inserted before the execution of the jth task

Problem Definition

Given is a system with thermal parameters $\mathbf{T}$ and an application $\mathbf{A}$.

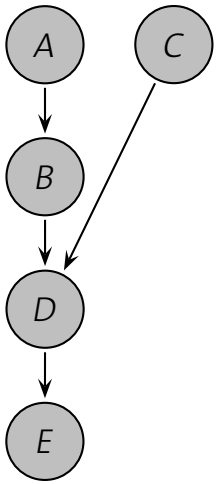
To find a stop-go schedule $\mathbf{S}$ that schedules the application to complete within its makespan and optimally minimizes the peak temperature

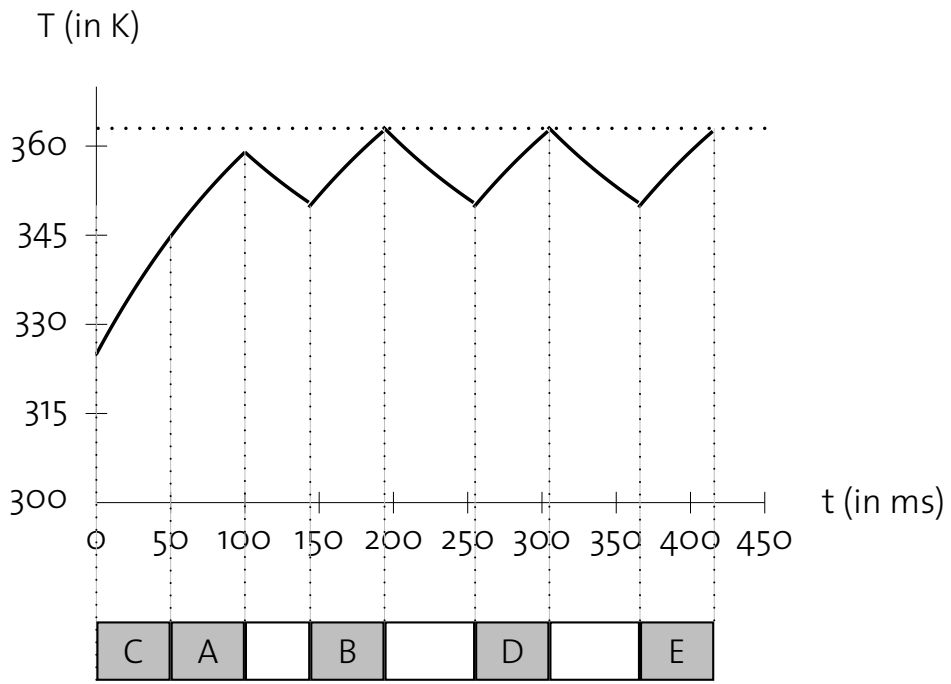
Putting it all together

System model: $T^{act} = 395K$, $T^{idl} = 325K$, $a^{act} = a^{idl} = 6.667\, s^{-1}$

Application model: Task graph with 5 tasks, $\tau_* = 50ms$, $\omega = 415ms$.

Schedule: $\sigma = (CABDE)$, $t^{idl} = (0, 0, 43, 61, 61)ms$





Outline

Step 1 : Given static-order of tasks

- Let the static-ordering σ be given
- We are to only compute optimal values of $\mathbf{t}^{\mathbf{idl}}$
- Let T_j denote the temperature of the system at the *finish* of the jth task

Step 1 : Given static-order of tasks

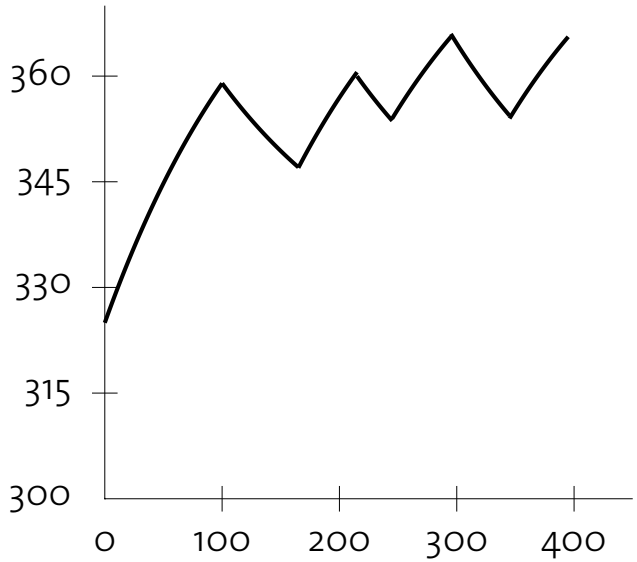
- Let the static-ordering σ be given
- We are to only compute optimal values of t^{idl}
- Let T_j denote the temperature of the system at the *finish* of the jth task
- We have the following result:

Theorem 1

Suppose an arbitrary non-preemptive stop-go schedule with given static-ordering of tasks, σ. Then, any change of the idle times such that none of T_j, $j \in \{1, \ldots, |V|\}$ decreases, and at least one increases, decreases the makespan.

▸ Additional slide: Proof sketch

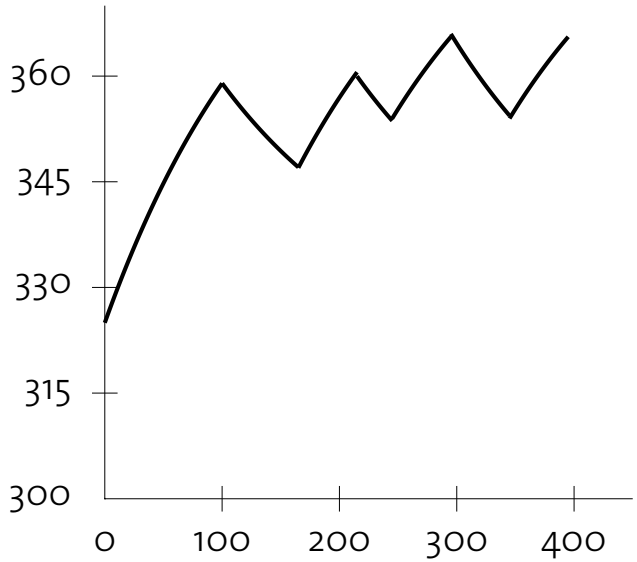
Interpretation of Theorem 1



$t^{idl} = (0, 0, 65, 30, 50)ms$

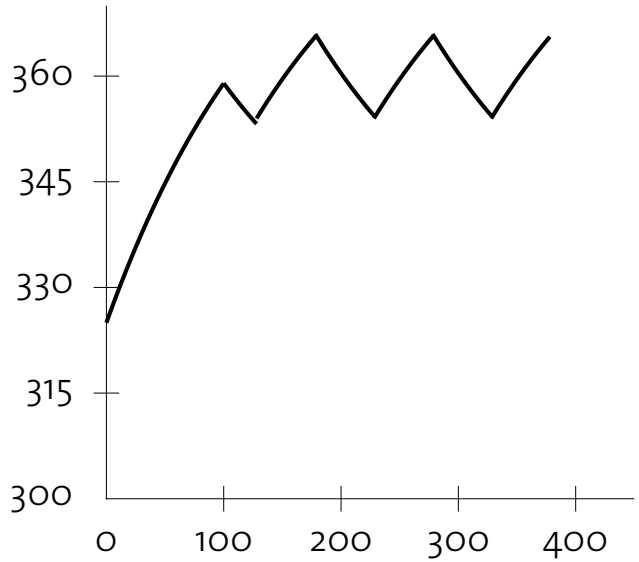
$\omega = 395ms$, $T^{max} = 366K$

Interpretation of Theorem 1



$t^{idl} = (0, 0, 65, 30, 50)ms$

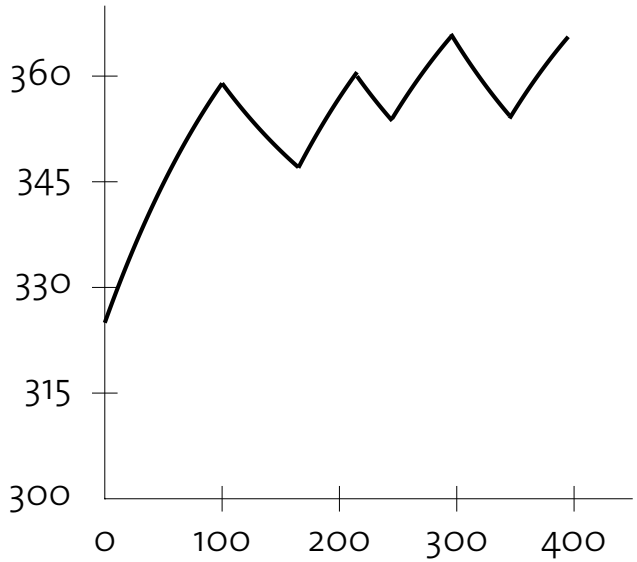
$\omega = 395ms$, $T^{max} = 366K$



$t^{idl} = (0, 0, 28, 50, 50)ms$

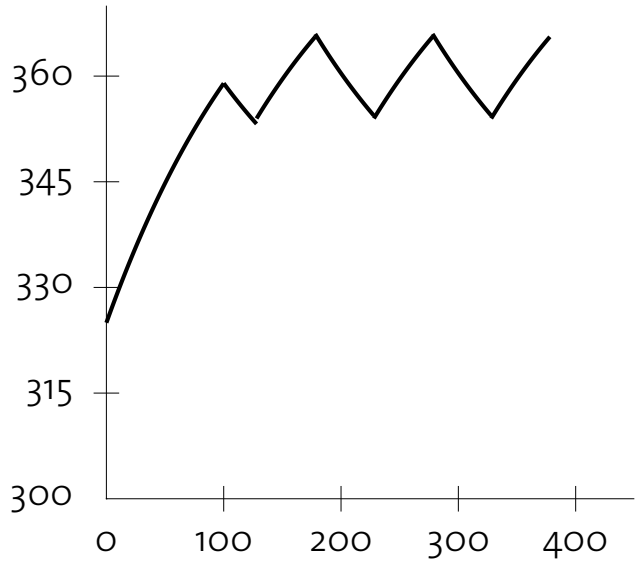
$\omega = 378ms$, $T^{max} = 366K$

Interpretation of Theorem 1



$t^{idl} = (0, 0, 65, 30, 50)ms$

$\omega = 395ms, T^{max} = 366K$



$t^{idl} = (0, 0, 28, 50, 50)ms$

$\omega = 378ms, T^{max} = 366K$

Insight

If the maximum temperature that we reach is T^{max}, we must not remain in idle longer than is required to end up with $T_j = T^{max}$

JUst Sufficient Throttling (JUST) schedule

Informal definition

JUST(σ) is a stop-go schedule where

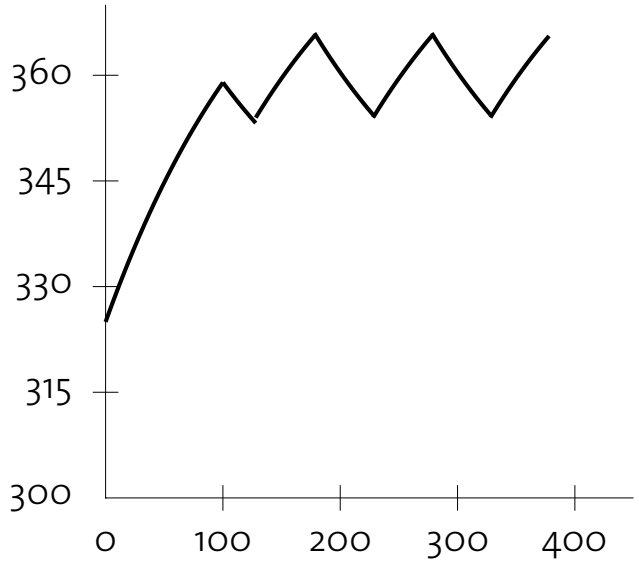
- the first p^{opt} tasks are executed with no idle time before them,
- the remaining tasks are executed with an idle time before them such that $T_j = T^{opt}$,
- the makespan $\omega = \omega_{max}$

JUst Sufficient Throttling (JUST) schedule

Informal definition

JUST(σ) is a stop-go schedule where

- the first p^{opt} tasks are executed with no idle time before them,
- the remaining tasks are executed with an idle time before them such that $T_j = T^{opt}$,
- the makespan $\omega = \omega_{max}$



$t^{idl} = (0, 0, 28, 50, 50)ms$

$\omega = 378ms, T^{max} = 366K$

Computation of $\mathfrak{p}^{opt}$ and $\mathfrak{T}^{opt}$

Are $\mathfrak{p}^{opt}$, $\mathfrak{T}^{opt}$ unique? How to compute them?

Yes, they are unique. Computation described in Algorithm 1 in paper.

Computation of $\mathfrak{p}^{opt}$ and T^{opt}

Are $\mathfrak{p}^{opt}$, T^{opt} unique? How to compute them?

Yes, they are unique. Computation described in Algorithm 1 in paper.

Brief Idea

- Let T_j^{cont} be the temperature of the system when executing the first j tasks without any idle time

Computation of p^{opt} and T^{opt}

Are p^{opt}, T^{opt} unique? How to compute them?

Yes, they are unique. Computation described in Algorithm 1 in paper.

Brief Idea

- Let T_j^{cont} be the temperature of the system when executing the first j tasks without any idle time
- We must have $T_{p^{opt}}^{cont} \leqslant T^{opt}$

Computation of p^{opt} and T^{opt}

Are p^{opt}, T^{opt} unique? How to compute them?

Yes, they are unique. Computation described in Algorithm 1 in paper.

Brief Idea

- Let T_j^{cont} be the temperature of the system when executing the first j tasks without any idle time
- We must have $T_{p^{opt}}^{cont} \leqslant T^{opt}$
- We must have $T_{p^{opt}+1}^{cont} > T^{opt}$

Computation of $\mathbf{p}^{opt}$ and $\mathbf{T}^{opt}$

Are $\mathbf{p}^{opt}$, $\mathbf{T}^{opt}$ unique? How to compute them?

Yes, they are unique. Computation described in Algorithm 1 in paper.

Brief Idea

- Let T_j^{cont} be the temperature of the system when executing the first j tasks without any idle time
- We must have $T_{p^{opt}}^{cont} \leqslant \mathbf{T}^{opt}$
- We must have $T_{p^{opt}+1}^{cont} > \mathbf{T}^{opt}$
- We must have have $\omega = \omega_{max}$
- Binary search over $\mathbf{p}^{opt}$

Optimality of JUST schedule

Theorem 2

JUST(σ) optimally minimizes the peak temperature amongst all stop-go schedules following a given static-order σ and satisfying the makespan constraint.

Optimality of JUST schedule

Theorem 2

JUST(σ) optimally minimizes the peak temperature amongst all stop-go schedules following a given static-order σ and satisfying the makespan constraint.

Proof idea

Take any schedule S with static-order σ.

Repeatedly apply Theorem 1 to modify S to become more and more similar to the JUST schedule.

Step 2: Integrated choice of σ and t^{idl}

▸ Skip this section

- Let σ^{opt} be the optimal static-order with a corresponding JUST(σ^{opt}) schedule with p^{opt} and T^{opt}
- Observe that
  1. Changing the order of execution of the first p^{opt} tasks does not change the makespan or peak temperature
  2. Changing the order of execution of the last ($|V| - p^{opt}$) tasks does not change the makespan or peak temperature
- To characterize σ^{opt}, it is sufficient to specify tasks is one among the first p^{opt} or not
- Hence, because of the special property of a JUST schedule, we have only a *binary decision problem*!

Algorithm to compute σ^{opt}

Brief Idea

- We start by assuming a target value of the peak temperature T^t, i.e. $T^{opt} \leqslant T^t$
- For a given T^t, we can compute σ^{opt} by solving a binary integer program (BIP) with binary variable x_ν denoting whether task ν belongs to the first p^{opt} tasks or not
- Precedence constraints, makespan constraints and peak temperature being below the target temperature can be modelled in the BIP
- If a feasible solution exists, we reduce T^t and again solve another instance of BIP

▸ Additional slide: BIP formulation

Periodically repeating applications

- We have considered a single run of a task graph
- But in practice applications run periodically, with period equal to maximum makespan (ω_{max})

Periodically repeating applications

- We have considered a single run of a task graph
- But in practice applications run periodically, with period equal to maximum makespan (ω_{max})
- What is different? The finishing temperature of the nth iteration is the finish temperature of the $(n+1)$th iteration
- We make a simple extension: In each iteration, compute the optimal `JUST` schedule
- Let $(T^{opt})_n$ denote the maximum temperature in the nth iteration

Periodically repeating applications

- We have considered a single run of a task graph
- But in practice applications run periodically, with period equal to maximum makespan (ω_{max})
- What is different? The finishing temperature of the nth iteration is the finish temperature of the $(n+1)$th iteration
- We make a simple extension: In each iteration, compute the optimal JUST schedule
- Let $(T^{opt})_n$ denote the maximum temperature in the nth iteration
- Questions: What is the maximum temperature of the system if run iteratively? Does JUST schedule optimally reduce it?

Results for periodic applications

Theorem 3

The series $\{(\mathbf{T}^{\mathbf{opt}})_{\mathbf{n}}\}$ converges for $\mathbf{n} \to \infty$ and the limit does not depend on the static-ordering of tasks for `JUST` schedules.

Corollary 4

With `JUST` schedules the highest temperature of a system running a periodic task graph is minimal and independent of the ordering of tasks.

Inferences from results

- If we want to run a task graph periodically, using `JUST` schedule in each period, optimally minimizes the peak temperature
- Further, the choice of σ is irrelevant to minimize the peak temperature
- Thus, we can choose σ w.r.t. other considerations, such as buffer capacity, and use the `JUST` schedule for this σ to optimally minimize peak temperature

Outline

System Properties

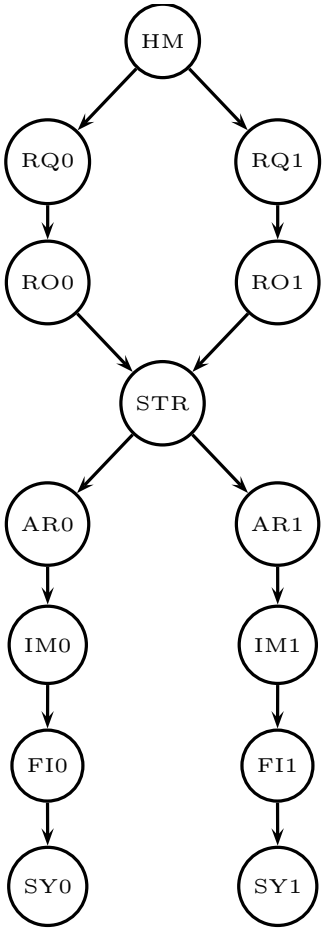
ARM-like core

$$T^{act} = 394K, T^{idl} = 325$$

$$a^{act} = a^{idl} = 6.667s^{-1}$$

$$T_o = 330K$$

Realistic example - MP3 application



Task ID	Execution Time ($\times 10^{-7}$s)
HM	236070
RQ0	139325
RQ1	139325
RO0	69385
RO1	69385
STR	73618
AR0	13088
AR1	13088
IM0	711744
IM1	711744
FI0	157184
FI1	157184
SY0	1866138
SY1	1866138

MP3 application - comparison with other schedules

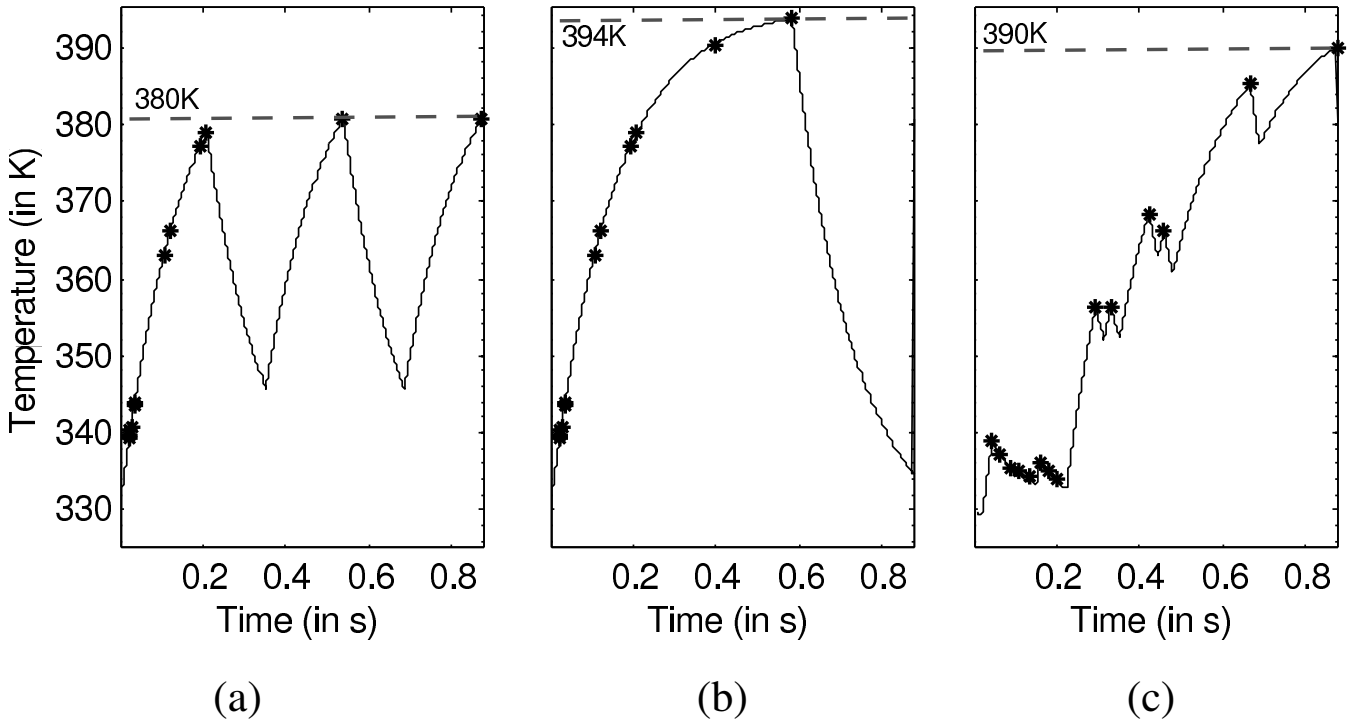
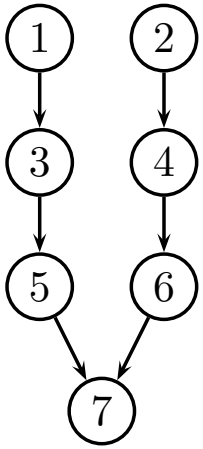


Figure: Temperature trace for (a) `JUST` schedule, (b) workload conserving execution, and (c) equally distributed idletimes

Synthetic example



Task ID	Execution Time (in msec)
1	30
2	140
3	20
4	50
5	50
6	50
7	50

Synthetic example - Results

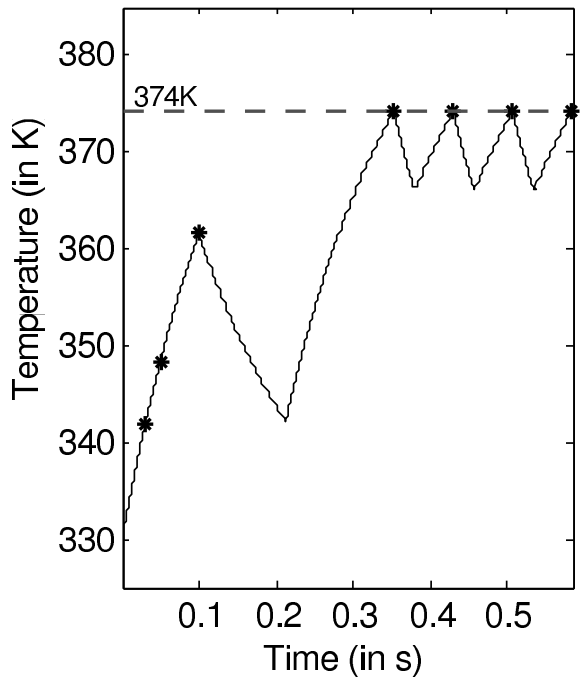


Figure: $T^t = 375K, \sigma = (1, 3, 5, 2, 4, 6, 7)$

$\{T^{opt}\} = (374K, 377K, 378K, 378.3K, \ldots)$

Synthetic example - Results

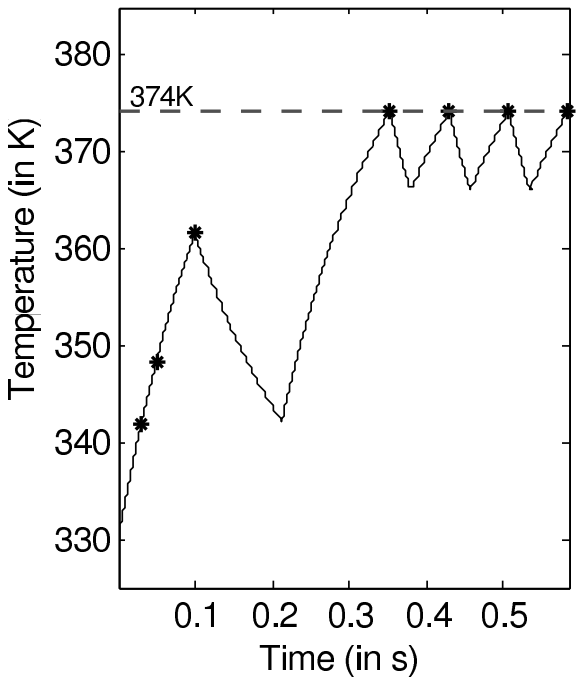


Figure: $T^t = 375K, \sigma = (1, 3, 5, 2, 4, 6, 7)$
$\{T^{opt}\} = (374K, 377K, 378K, 378.3K, \ldots)$

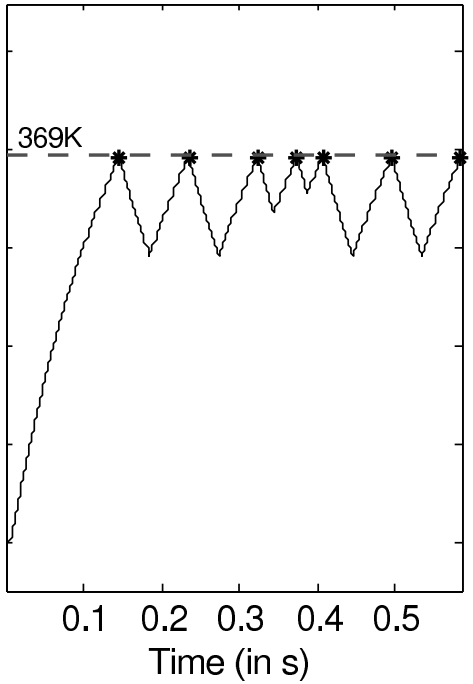


Figure: $T^t = 370K, \sigma = (2, 1, 3, 5, 4, 6, 7)$
$\{T^{opt}\} = (369K, 377K, 378K, 378.3K, \ldots)$

Outline

Conclusions and future work

- Stop-go execution is an effective way to manage on-chip temperature while executing real-time applications
- We proved optimality of the proposed `JUST` schedule for a given static-ordering
- We gave an algorithm to optimize the static-ordering as well
- For periodically repeating tasks, peak temperature is optimally minimized by `JUST` schedule and is independent of the static-ordering

Conclusions and future work

- Stop-go execution is an effective way to manage on-chip temperature while executing real-time applications
- We proved optimality of the proposed `JUST` schedule for a given static-ordering
- We gave an algorithm to optimize the static-ordering as well
- For periodically repeating tasks, peak temperature is optimally minimized by `JUST` schedule and is independent of the static-ordering
- Future work:
  - Extension to multi-processor distributed systems
  - Combination of DVS and stop-go scheduling

Thank you for your attention

Any questions?

Sketch of proof of theorem 1

1. Define F as

$$F = \prod_{j=1}^{|V|} \exp^{-a^{idl} t_j^{idl}}$$

2. We can reduce the above to

$$F = c \cdot \left(T_{|V|} - T'_{|V|} \right) \cdot \prod_{j=1}^{|V|-1} \frac{T_j - T'_j}{T_j - T^{idl}} \tag{1}$$

   From which we can derive

$$\frac{\partial F}{\partial T_{|V|}} > 0 \tag{2}$$

3. Also, we can show that $\frac{\partial F}{\partial \omega} < 0$. Hence the theorem

Back

Binary Integer Program for a given target temperature

$$
\begin{array}{lrcll}
\text{Maximise} & 1 & & & \\
\text{subject to} & x_v & \in & \{0,1\} & \forall\, v \in V \\
& x_{v1} & \leqslant & x_{v2} & \forall\, (v1, v2) \in E \\
& A1^T x & \geqslant & b1 & \\
& A2^T x & \leqslant & b2 &
\end{array}
$$

where $A1_v = \dfrac{T^t - T''_v}{f^{act}(t_j^{act})(T^t - T^{idl})}$, $b1 = f^{idl}(\omega - \tau_{tot})$, $A2_v = \tau(v)$,

$b2 = \tau^{tot} - (f^{act})^{-1}\left(\dfrac{T^{act} - T^t}{T^{act} - T_o}\right)$. Back