Definitions: Jobs and Tasks

**Job** (a single unit of work)

Release time $t_r$, relative deadline $D$ (absolute one is $t_d = D + t_r$) and execution time $C$. Tardiness of job, completed at $t_c$: $\max(0, t_c - t_d)$.

**Sporadic Task** (a sequence of similar dependent jobs)

Characterized by period $T$ and Worst-Case Execution Time (WCET) $C$. Releases jobs $j_1, \ldots, j_k, \ldots$, such that

- Distance between releases of $j_i$ and $j_{i+1}$ is at least $T$.
- Relative deadline of $j_i$ is $T$ time units from the release.
- WCET of $j_i$ does not exceed $C$.

Task $\tau_i$ utilization is $U_i = C / T$, task tardiness is supremum of all its jobs tardiness.
Definitions: Affinity Masks

**Affinity Mask** $\alpha_i$ of task $\tau_i$

A set of processors, that can execute given task $\tau_i$ (any processor that is not in the mask cannot execute the task).

**Affinity Graph** $AG(\tau)$

- $n$ vertices $\tau_1, \ldots, \tau_n$ (representing tasks)
- $m$ vertices $\pi_1, \ldots, \pi_m$ (representing cores)
- Has an edge $(\tau_i, \pi_j)$ if and only if $\pi_j \in \alpha_i$ (i.e., task $\tau_i$ can execute on core $\pi_j$).

\[\alpha_1 = \{1\}, \quad \alpha_2 = \{1, 2\}, \quad \alpha_3 = \{1, 2\}, \quad \alpha_4 = \{3\}.\]
Definitions: HRT/SRT-schedulability

Schedulability of \( \tau \)

\( \tau \) is HRT-schedulable (resp., SRT-schedulable) under scheduling algorithm S if each task in \( \tau \) has zero (resp., bounded) tardiness in any schedule for \( \tau \) generated by S.

Feasibility of \( \tau \)

\( \tau \) is HRT-feasible (resp., SRT-feasible) if, for any job release sequence, a schedule exists in which each task has zero (resp., bounded) tardiness.

Optimality of Scheduler S

S is HRT-optimal (resp., SRT-optimal) if every HRT-feasible (resp., SRT-feasible) task set \( \tau \) is HRT-schedulable (resp., SRT-schedulable) under S.
Problem and Motivation

Affinity masks usage

- simplify cache usage analysis
- reduce I/O-related overheads (interrupts)
- easier load balancing
- ensure security isolation

Affinity masks support

- Linux’s SCHED_DEADLINE policy (RT scheduler)
- Windows, Mac OS X, FreeRTOS, QNX, VxWorks, ...

Affinity masks schedulers

- [Baruah 2013]: HMR, requires future knowledge
- [Brandenburg 2014]: improvement for Linux scheduler, HMR
- [Bonifaci 2016]: HMR, only hierarchical, high overheads

HMR = high migrations rate

No optimal scheduler, available for implementation, exist.
Problem and Motivation

Affinity masks usage
- simplify cache usage analysis
- reduce I/O-related overheads (interrupts)
- easier load balancing
- ensure security isolation

Affinity masks support
- Linux’s SCHED_DEADLINE policy (RT scheduler)
- Windows, Mac OS X
- FreeRTOS, QNX, VxWorks, ...
Problem and Motivation

Affinity masks usage
- simplify cache usage analysis
- reduce I/O-related overheads (interrupts)
- easier load balancing
- ensure security isolation

Affinity masks support
- Linux’s SCHED_DEADLINE policy (RT scheduler)
- Windows, Mac OS X
- FreeRTOS, QNX, VxWorks, ...

Affinity masks schedulers
- [Baruah 2013]: HMR, requires future knowledge
- [Brandenburg 2014]: improvement for Linux scheduler, HMR
- [Bonifaci 2016]: HMR, only hierarchical, high overheads

HMR = high migrations rate
No optimal scheduler, available for implementation, exist.
Problem and Motivation

Our goals:
- Build *HRT/SRT-optimal* scheduler that supports *arbitrary* affinity masks
- Make scheduler *as fast as possible* (in the worst case)
- Reduce number of task *migrations* over cores
Our goals:
- Build \textit{HRT/SRT-optimal} scheduler that supports \textit{arbitrary} affinity masks
- Make scheduler \textit{as fast as possible} (in the worst case)
- Reduce number of task \textit{migrations} over cores

What do we do:
- Restrict affinity masks, preserving task set feasibility (affinity graph reduction)
**Static vs Dynamic Schedulers**

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>How to build</strong></td>
<td><strong>Pros</strong></td>
</tr>
<tr>
<td>Create schedule template</td>
<td>Easy to analyze</td>
</tr>
<tr>
<td>Generate “almost static” schedule</td>
<td>Small overheads</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cons</strong></td>
<td><strong>Cons</strong></td>
</tr>
<tr>
<td>Hard to adjust schedule</td>
<td>Hard to analyze</td>
</tr>
<tr>
<td>Offline phase may take lots of time</td>
<td>High overheads</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frame is a template of schedule over $[0, 1)$. We scale this template to $[0, F)$ and repeat.
<table>
<thead>
<tr>
<th></th>
<th>Baruah</th>
<th>UB Test</th>
<th>Max Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test</strong></td>
<td>$\forall i : \sum_{j \in \alpha_i} x_{ij} = 1$</td>
<td>$\forall S \subset \tau : U_S \leq \alpha_{\tau}$</td>
<td>$</td>
</tr>
<tr>
<td></td>
<td>$\forall j : \sum_{i} x_{ij} U_i \leq 1$</td>
<td>For every subset of tasks, its aggregated affinity mask size is at least its utilization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\forall i, j : x_{ij} \geq 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Complexity</strong></td>
<td>$\tilde{O}(mn \cdot (m + n)^{2.9})$</td>
<td>$mn \cdot 2^n$</td>
<td>$\tilde{O}(mn\sqrt{m+n})$</td>
</tr>
</tbody>
</table>

$\tilde{O}$ ignores logarithmic factors: $\tilde{O}(g(n)) = O(g(n) \log^k g(n))$ for some natural number $k$. 
Schedulability Conditions: Utilization Balance

\( \pi_1 \) \( \pi_2 \) \( \pi_3 \)

\( \tau_1 \) \( \tau_2 \) \( \tau_3 \) \( \tau_4 \)

(a) Affinity Graph \( AG(\tau) \)

Utilization Balance Feasibility Test check for \( S = \{ \tau_1, \tau_2 \} \):

\[ U_1 + U_2 \leq 2. \]
Schedulability Conditions: Utilization Balance

(a) Affinity Graph $AG(\tau)$

(b) Aggregated affinity mask of $\{\tau_1, \tau_2\}$.

Utilization Balance Feasibility Test check for $S = \{\tau_1, \tau_2\}$: $U_1 + U_2 \leq 2$. 

Sergey Voronov and James H. Anderson AM-RED: An Optimal Semi-Partitioned Scheduler Assuming Arbitrary Affinity Masks
Schedulability Conditions: Max Flow Test

(a) Affinity Graph $AG(\tau)$
Schedulability Conditions: Max Flow Test

(a) Affinity Graph $AG(\tau)$

(b) Flow Network $FN(\tau)$

Max Flow Feasibility Test: the maximum flow over $FN(\tau)$ is $U$. 
AM-Red Outline

- Run Max Flow Test
- Remove cycles from Share Graph
- Build extended frame
- Build frame

AM-Red = Affinity Masks REDuction
Max Flow Test passed: $|f| = U$

$f(\tau_i, \pi_j)$ is a flow between $\tau_i$ and $\pi_j$. 
From Flow Solution to Frame: Share Graph

Max Flow Test passed: \[ |f| = U \]

\[ f(\tau_i; \pi_j) \] is a flow between \( \tau_i \) and \( \pi_j \).

Share Graph \( SG(\tau) \)

Affinity graph \( AG(\tau) \) with \( (\tau_i; \pi_j) \) edges removed, if \( f(\tau_i; \pi_j) = 0 \).

\( \tau_1 \) \( \tau_2 \) \( \tau_3 \) \( \tau_4 \)
\( \pi_1 \) \( \pi_2 \) \( \pi_3 \)

(a) Affinity Graph \( AG(\tau) \)

Figure: \( f(\tau_3; \pi_2) = 0 \), all other edges in \( AG(\tau) \) have non-zero \( f \) values.

\( \tau_1 \) \( \tau_2 \) \( \tau_3 \) \( \tau_4 \)
\( \pi_1 \) \( \pi_2 \) \( \pi_3 \)

(b) Share Graph \( SG(\tau) \)
From Flow Solution to Frame: Cycles Removal

If Share Graph $SG(\tau)$ has cycles we can remove them one-by-one. $f_m$ is minimum of $f(\tau_i, \pi_j)$ over all cycle edges.

(a) Cycle in Share Graph before removal.

(b) “Cycle” in Share Graph after removal.

Figure: For dashed edges $f(\tau_i, \pi_j)$ decreases, for solid it increases. $f_m = f(\tau_2, \pi_1) = 0.1$. 
Let $I_E(\tau_i, \pi_j)$ be the union of all continuous intervals on core $\pi_j$ allocated to task $\tau_i$.

\[
\begin{align*}
\forall i, j : & \quad I_E(\tau_i, \pi_j) \text{ is a single continuous interval} \\
\forall i : & \quad \bigcup_j I_E(\tau_i, \pi_j) \text{ is a single continuous interval} \\
\forall j : & \quad \bigcup_i I_E(\tau_i, \pi_j) \text{ is a single continuous interval} \\
\forall i, j_1, j_2 : & \quad |I_E(\tau_i, \pi_{j_1}) \cap I_E(\tau_i, \pi_{j_2})| = 0 \text{ (no overlapping)}
\end{align*}
\]

Frame

\[
\begin{array}{c|ccc}
\tau_1 & \tau_2 & \tau_3 \\
\hline
\pi_1 & 1/3 & 2/3 & - \\
\pi_2 & 1/3 & - & 2/3 \\
\pi_3 & 1/3 & - & - \\
\pi_4 & - & - & 1/3 \\
\end{array}
\]
1: **for** \( \pi_j \in \text{cores order} \) **do**

2: **for** \( \tau_i \in \text{tasks order for } \pi_j \) **do**

3: allocate interval for \( \tau_i \) on \( \pi_j \) (right after previous one)
Proper Order properties:

\[ \forall i, j : \tau_i \text{ appears in task order of core } \pi_j \text{ if and only if } f(\tau_i, \pi_j) > 0 \]

\[ \forall i : \text{ task } \tau_i \text{ can be non-first task on at most one core} \]

\[ \forall i : \text{ if task } \tau_i \text{ is non-first on core } \pi_j, \text{ then on previous cores } \tau \text{ have not appeared} \]

How to get Proper Order:

- Run BFS over acyclic Share Graph
- Order cores in the discovery order
- Order tasks for each core in the discovery order
From Flow Solution to Frame: Extended Frame to Frame

\[ t \leftarrow t \mod 1 \] creates a correct schedule for \([0, 1]\) from an extended frame.

\[
\begin{array}{cccc}
\tau_1 & \tau_2 & \tau_3 \\
\pi_1 & 1/3 & 2/3 & - \\
\pi_2 & 1/3 & - & 2/3 \\
\pi_3 & 1/3 & - & - \\
\pi_4 & - & - & 1/3 \\
\end{array}
\]
AM-Red: Sporadic Tasks Scheduling

![Diagram showing allocation intervals for tasks π₁, π₂, and π₃, with job release, job deadline, and job completion for task T₃.](image)

- Allocation intervals for task T₃
- Allocation intervals for all other tasks

- Job release
- Job deadline
- Job completion
## AM-Red: Complexity

<table>
<thead>
<tr>
<th>Stage</th>
<th>Complexity</th>
<th>Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Flow</td>
<td>$\tilde{O}(mn\sqrt{m+n})$</td>
<td>$\Omega(mn)$</td>
</tr>
<tr>
<td>Cycles Removal</td>
<td>$O(m^2n^2)$</td>
<td>$\Omega(mn)$</td>
</tr>
<tr>
<td>Extended Frame</td>
<td>$O(m+n)$</td>
<td>$\Omega(m+n)$</td>
</tr>
<tr>
<td>Frame</td>
<td>$O(m+n)$</td>
<td>$\Omega(m+n)$</td>
</tr>
</tbody>
</table>

At most $m - 1$ migrating tasks; at most $2m - 2$ migrations per frame.

---

$\tilde{O}$ ignores logarithmic factors: $\tilde{O}(g(n)) = O(g(n)\log^k g(n))$ for some natural number $k$. 
<table>
<thead>
<tr>
<th>Stage</th>
<th>Complexity</th>
<th>Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Flow</td>
<td>$\tilde{O}(mn\sqrt{m+n})$</td>
<td>$\Omega(mn)$</td>
</tr>
<tr>
<td>Cycles Removal</td>
<td>$O(m^2n^2)$</td>
<td>$\Omega(mn)$</td>
</tr>
<tr>
<td>Extended Frame</td>
<td>$O(m + n)$</td>
<td>$\Omega(m + n)$</td>
</tr>
<tr>
<td>Frame</td>
<td>$O(m + n)$</td>
<td>$\Omega(m + n)$</td>
</tr>
</tbody>
</table>

At most $m - 1$ migrating tasks; at most $2m - 2$ migrations per frame.

Max Flow + Cycles Removal = find and modify $f(\tau_i, \pi_j)$.

$\tilde{O}$ ignores logarithmic factors: $\tilde{O}(g(n)) = O(g(n)\log^k g(n))$ for some natural number $k$. 
$AG(\tau)$ does not have any cycles $\Rightarrow SG(\tau)$ does not have cycles.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Flow</td>
<td>$O(m + n)$</td>
</tr>
<tr>
<td>Cycles Removal</td>
<td>0</td>
</tr>
<tr>
<td>Extended Frame</td>
<td>$O(m + n)$</td>
</tr>
<tr>
<td>Frame</td>
<td>$O(m + n)$</td>
</tr>
</tbody>
</table>

Max Flow: run BFS over $AG(\tau)$ and apply Ford-Fulkerson algorithm.  
FF heuristic: augmenting path searches over vertexes in reversed discovery order.

Input data size: $O(m + n)$. 

For any two masks $\alpha_i, \alpha_j$: $\alpha_i \subset \alpha_j$, or $\alpha_i \supset \alpha_j$, or $\alpha_i \cup \alpha_j = \emptyset$.

Importance: follows multiprocessor architecture.

Input data size: $O(mn)$, special packing is needed to ensure $O(m + n)$.

Hierarchical masks specialty: at most $2m - 1$ unique masks and tree masks structure.
AM-Red Enhancements: Hierarchical Masks

For any two masks $\alpha_i, \alpha_j$: $\alpha_i \subset \alpha_j$, or $\alpha_i \supset \alpha_j$, or $\alpha_i \cup \alpha_j = \emptyset$.

Importance: follows multiprocessor architecture.

Input data size: $O(mn)$, special packing is needed to ensure $O(m + n)$.

Hierarchical masks specialty: at most $2m - 1$ unique masks and tree masks structure.
AM-Red Enhancements: Hierarchical Masks

For any two masks $\alpha_i, \alpha_j$: $\alpha_i \subset \alpha_j$, or $\alpha_i \supset \alpha_j$, or $\alpha_i \cup \alpha_j = \emptyset$.

Importance: follows multiprocessor architecture.

Input data size: $O(mn)$, special packing is needed to ensure $O(m + n)$.

Hierarchical masks specialty: at most $2m - 1$ unique masks and tree masks structure.

Max Flow: compute shown order and apply Ford-Fulkerson algorithm. (note that direct flow network $FN(\tau)$ construction requires $\Omega(mn)$, so we avoid it) FF heuristic: augmenting path searches over tasks vertexes according to masks order.
(a) Light tasks.

(b) Medium tasks.

(c) Heavy tasks.

Figure: Exp. 1 (hierarchical masks): total number of migrations under AM-Red and HPA-EDF (assuming periodic releases), averaged over the generated task sets, as a function of relative system utilization.
Figure: Exp. 2 (hierarchical masks): maximum tardiness, averaged over the generated task sets, as a function of relative system utilization.

[Brandenburg 2014] Linux’s processor affinity API, refined: shifting real-time tasks towards higher schedulability