Fixed-Priority Multiprocessor Scheduling
[RTAS 2010]

Joint work with
Nan Guan, Martin Stigge and Yu Ge
Northeastern University, China
Uppsala University, Sweden

Real-time Systems

- N periodic tasks (of different rates/periods)
- How to schedule the jobs to avoid deadline miss?

On Single-processors

- Liu and Layland’s Utilization Bound [1973]
  (the 19th most cited paper in computer science)
  \[ \sum_{i \in T} U_i \leq N \left( \frac{2^{1/N}}{N} - 1 \right) \]
  \[ \Rightarrow \text{the task set is schedulable} \]

- \( N \to \infty \), \( N \left( \frac{2^{1/N}}{N} - 1 \right) = 69.3\% \)
- Scheduled by RMS (Rate Monotonic Scheduling)

Rate Monotonic Scheduling

- Priority assignment: shorter period \( \rightarrow \) higher prio.
- Run-time schedule: the highest priority first

Liu and Layland’s Utilization Bound

- Schedulability Analysis

Liu and Layland’s Utilization Bound

- Schedulability Analysis

- Liu and Layland’s bound:
  \[ 3 \times \left( \frac{2^{1/3}}{3} - 1 \right) = 77.9\% \]
Multiprocessor (multicore) Scheduling

- Significantly more difficult:
  - Timing anomalies
  - Hard to identify the worst-case scenario
  - Bin-packing/NP-hard problems
  - Multiple resources e.g. caches, bandwidth
  - ... ...

Open Problem (since 1973)

- Find a multiprocessor scheduling algorithm that can achieve Liu and Layland’s utilization bound

\[ \frac{\sum C_i}{T} \leq N(\frac{1}{N} - 1) \]

= the task set is schedulable

Best Known Results (before 2010)

- Lehoczky et al. CMU ECRTS 2009
- [OPODIS’08]
- [TPDS’05]
- [ECRTS’03]
- [RTSS’04]
- [RTCSA’06]

Our New Result

RTAS 2010
RTSS 2010_submitted

69.3
### Multiprocessor Scheduling

**Global Scheduling**

Would fixed-priority scheduling e.g. "RMS" work?

- Unfortunately "RMS" suffers from the Dhall's anomaly
- Utilization may be "0%"

---

**Dhall's anomaly**

(M+1 tasks and M processors)

\[
\frac{\epsilon}{1} \frac{\epsilon}{1} \ldots \frac{1}{(\epsilon+1)}
\]

\[
P_1 P_2 P_3 \ldots
\]

\[
U = \frac{M^{*}\epsilon + 1/(1+\epsilon)}{M} \rightarrow 0
\]

when \( \epsilon \rightarrow 0 \) and \( M \rightarrow +\infty \)

---

**Partitioned Scheduling**

Schedule the 3 tasks on 2 CPUs using "RMS"

---

**Deadline miss**
Multiprocessor Scheduling

Partitioned Scheduling

- The Partitioning Problem is similar to Bin-packing Problem (NP-hard)

- Limited Resource Usage, 50% necessary condition to guarantee schedulability

\[ \sum \frac{C_i}{T_i} \leq 1 \]

\[ U(r) = \frac{(M+1)(0.5 + \varepsilon)}{M} \rightarrow 0.5 \quad \text{when} \quad \varepsilon \rightarrow 0 \quad \text{and} \quad M \rightarrow +\infty \]

Partitioned Scheduling with Task Splitting

\[ \sum \frac{C_i}{T_i} \leq 1 \]

\[ U(r) = \frac{(M+1)(0.5 + \varepsilon)}{M} \rightarrow 0.5 \quad \text{when} \quad \varepsilon \rightarrow 0 \quad \text{and} \quad M \rightarrow +\infty \]

Multiprocessor Scheduling
**Partitioned Scheduling**

- Partitioning

```
1 2 3
4 5 6
7 8 9
```

- Bin-Packing with Item Splitting

- Resource can be “fully” (better) utilized

```
Bin1 Bin2 Bin3
1 2 3
4 5 6
```

**Previous Algorithms**

[Kato et al. [IPDPS'08] [Kato et al. RTAS'09] [Lakshmanan et al. ECRTS'09]]

- Sort the tasks in some order e.g. utilization or priority order
- Select a processor, and assign as many tasks as possible

```
8 7 6 5 4 3 2 1
```

**Lakshmanan’s Algorithm** [ECRTS’09]

- Sort all tasks in decreasing order of utilization

```
8 7 6 5 4 3 2 1
```

**Lakshmanan’s Algorithm** [ECRTS’09]

- Pick up one processor, and assign as many tasks as possible

```
8 7 6 5 4 3 2 1
```

**Lakshmanan’s Algorithm** [ECRTS’09]

- Pick up one processor, and assign as many tasks as possible

```
7 6 5 4 3 2 1
```

```
P1
```

```
P1
```

```
P1
```

```
P1
```
Lakshmanan’s Algorithm [ECRTS’09]

- Pick up one processor, and assign as many tasks as possible

<table>
<thead>
<tr>
<th>lowest util.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>highest util.</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

P1, P2

Lakshmanan’s Algorithm [ECRTS’09]

- Pick up one processor, and assign as many tasks as possible

<table>
<thead>
<tr>
<th>lowest util.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>highest util.</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

P1, P2
Lakshmanan’s Algorithm \cite{ECRTS'09}

- Pick up one processor, and assign as many tasks as possible

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6*</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>5*</td>
</tr>
</tbody>
</table>

key feature: “depth-first” partitioning with decreasing utilization order

Utilization Bound: 65%
Our Algorithm

- Sort all tasks in increasing priority order

```
7
6
5
4
3
2
1
```

"width-first" partitioning with increasing priority order

```
Our Algorithm

- Select the processor on which the assigned utilization is the lowest

```

```
7
6
5
4
3
2
1
```

```
P1  P2  P3
```

```
Our Algorithm

- Select the processor on which the assigned utilization is the lowest

```

```
7
6
5
4
3
2
1
```

```
P1  P2  P3
```

```
Our Algorithm

- Select the processor on which the assigned utilization is the lowest

```

```
5
4
3
2
1
```

```
P1  P2  P3
```

```
Our Algorithm

- Select the processor on which the assigned utilization is the lowest

```

```
4
3
2
1
```

```
P1  P2  P3
```

```
Our Algorithm

- Select the processor on which the assigned utilization is the lowest

```

```
7
6
5
4
3
2
1
```

```
P1  P2  P3
```
Our Algorithm

- Select the processor on which the assigned utilization is the **lowest**

```
<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>21</td>
<td>22</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>21</td>
<td>22</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
```

Our Algorithm

- Select the processor on which the assigned utilization is the lowest

Key feature: "width-first" partitioning with increasing priority order

<table>
<thead>
<tr>
<th>Processor</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilization</td>
<td>21</td>
<td>12</td>
<td>37</td>
</tr>
<tr>
<td>Priority</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Comparison

Why is our algorithm better?

By our algorithm split tasks generally have higher priorities

Ours: width-first & increasing priority order

Previous: depth-first & decreasing utilization order

<table>
<thead>
<tr>
<th>Processor</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilization</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Priority</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

Split Task

- Consider an extreme scenario:
  - suppose each subtask has the highest priority
  - schedulable anyway, we do not need to worry about their deadlines

The difficult case is when the tail task is not on the top

- the key point is to ensure the tail task is schedulable

<table>
<thead>
<tr>
<th>Subtasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ₁</td>
</tr>
<tr>
<td>τ₂</td>
</tr>
<tr>
<td>τ₃</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processor</th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilization</td>
<td>R₁</td>
<td>R₂</td>
<td>R₃</td>
</tr>
<tr>
<td>Deadline</td>
<td>T₁</td>
<td>T₂</td>
<td>T₃</td>
</tr>
</tbody>
</table>

Why is our algorithm better?

Split Task

- Subtasks should execute in the correct order

<table>
<thead>
<tr>
<th>Subtasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ₁</td>
</tr>
<tr>
<td>τ₂</td>
</tr>
<tr>
<td>τ₃</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processor</th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilization</td>
<td>R₁'</td>
<td>R₂'</td>
<td>R₃'</td>
</tr>
</tbody>
</table>

Split Task

- Subtasks get "shorter deadlines"

<table>
<thead>
<tr>
<th>Subtasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ₁</td>
</tr>
<tr>
<td>τ₂</td>
</tr>
<tr>
<td>τ₃</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processor</th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilization</td>
<td>R₁'</td>
<td>R₂'</td>
<td>R₃'</td>
</tr>
</tbody>
</table>
Split Task

- Subtasks should execute in the correct order

\[ \tau_i \]

\[ \tau_{i1} \]

\[ \tau_{i2} \]

\[ \tau_{i3} \]

\[ P_1 \]

\[ P_2 \]

\[ P_3 \]

\[ R_i \]

\[ T_i \]

\[ \Delta_i = T_i - R_{i1} - R_{i2} \]

These two are on the top: no problem with schedulability

Split Task

- Subtasks should execute in the correct order

Why the tail task is schedulable?

The typical case: two CPUs and task 2 is split to two sub-tasks

As we always select the CPU with the lowest load assigned, we know

\[ Y_i \leq U_{i1} \]

\[ Y_i \leq U_{i1} - U_{i2} \]

That is, the "blocking factor" for the tail task is bounded.

Theorem

For a task set in which each task \( \tau_i \) satisfies

\[ U_i \leq \frac{\Theta(N)}{1 + \Theta(N)} \]

we have

\[ \frac{\sum C_i / T_i}{M} \leq N^{2^{1/N} - 1} \]

\[ \Rightarrow \text{the task set is schedulable} \]

\[ \Theta(N) = N^{2^{1/N} - 1} \]

\[ N \to \infty, \quad \frac{\Theta(N)}{1 + \Theta(N)} \approx 0.41 \]

Problem of Heavy Tasks

lowest priority

[5]

[6]

[7]

P1

P2

P3

highest priority

[1]

[2]

[3]

[4]
Problem of Heavy Tasks

highest priority

lowest priority

Problem of Heavy Tasks

highest priority

lowest priority

Problem of Heavy Tasks

highest priority

lowest priority

Problem of Heavy Tasks

highest priority

lowest priority

Problem of Heavy Tasks

highest priority

lowest priority


Problem of Heavy Tasks

the heavy tasks' tail task may have too low priority level

Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>6^1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>
Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

<table>
<thead>
<tr>
<th>Task</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

Lowest priority

P1  P2  P3
6  4  2
5  5  9

Highest priority

1  1  1

Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

Lowest priority

P1  P2  P3
6  4  2
5  5  9

Highest priority

1  1  1

Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

Lowest priority

P1  P2  P3
6  4  2
5  5  9

Highest priority

1  1  1

Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

Lowest priority

P1  P2  P3
6  4  2
5  5  9

Highest priority

1  1  1

Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

Lowest priority

P1  P2  P3
6  4  2
5  5  9

Highest priority

1  1  1

Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

Lowest priority

P1  P2  P3
6  4  2
5  5  9

Highest priority

1  1  1

Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

Lowest priority

P1  P2  P3
6  4  2
5  5  9

Highest priority

1  1  1

Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

Lowest priority

P1  P2  P3
6  4  2
5  5  9

Highest priority

1  1  1

Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

Lowest priority

P1  P2  P3
6  4  2
5  5  9

Highest priority

1  1  1

Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

Lowest priority

P1  P2  P3
6  4  2
5  5  9

Highest priority

1  1  1

Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

Lowest priority

P1  P2  P3
6  4  2
5  5  9

Highest priority

1  1  1

Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

Lowest priority

P1  P2  P3
6  4  2
5  5  9

Highest priority

1  1  1

Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)

Lowest priority

P1  P2  P3
6  4  2
5  5  9

Highest priority

1  1  1

Theorem

- By introducing the pre-assignment mechanism, we have

\[
\frac{\sum C_i/T_i}{M} \leq N(2^{1/N} - 1)
\]

\Rightarrow \text{the task set is schedulable}

Liu and Layland’s utilization bound for all task sets!
Overhead
- In both previous algorithms and ours
  - The number of task splitting is at most $M-1$
  - task splitting -> extra "migration/preemption"
- Our algorithm on average has less task splitting

![Diagram showing task splitting](image)

Implementation
- Easy!
  - One timer for each split task
  - Implemented as "task migration"

![Diagram showing implementation](image)

Further Improvement
- Using Liu and Layland’s Utilization Bound

![Diagram showing further improvement](image)

Utilization Bound is Pessimistic
- The Liu and Layland utilization bound is sufficient but not necessary
- many task sets are actually schedulable even if the total utilization is larger than the bound

![Diagram showing utilization bound](image)

Exact Analysis
- Exact Analysis: Response Time Analysis [Lehoczky_89]
  - pseudo-polynomial

![Diagram showing exact analysis](image)
Utilization Bound v.s. Exact Analysis

- On single processors

Utilization bound Test for RMS

Exact Analysis for RMS

- On Multiprocessors

Can we do something similar on multiprocessors?

Utilization bound Test

the algorithm introduced above

Beyond Layland & Liu's Bound [RTSS 2010, rejected!]

- Our RTAS10 algorithm:
  - Increasing RMS priority order & worst-fit partitioning
  - Utilization test to determine the maximal load for each processor
  - The maximal load for each processor bounded by $69.3\% \times \frac{N}{2^N - 1}$

- Improved algorithm:
  - Employ Response Time Analysis to determine the maximal workload on each processor
  - More flexible behavior (more difficult to prove...)
  - Same utilization bound for the worst case, but
  - Much better average performance (by simulation)

I believe this is "the best algorithm" one can hope for "fixed-priority multiprocessor scheduling"

Conclusions

- The (multicore) Timing Problem is challenging
  - Difficult to guarantee Real-Time
  - and Difficult to analyze/predict

- Solutions: Partition & Isolation
  - Shared caches: coloring/partition
  - Memory bus/bandwidth: TDMA, ?
  - Processor cores: partition-based scheduling

Thanks!