Fixed-Priority Multiprocessor Scheduling

Real-time Systems

- N periodic tasks (of different rates/periods)

Utilization/workload: \( \frac{C_i}{T_i} \)

- 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)

\[ \sum_{\tau_i \in \tau} U_i \leq N(2^{1/N} - 1) \]

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

- \( N \rightarrow \infty, \quad N(2^{1/N} - 1) = 69.3\% \)
- Scheduled by RMS (Rate Monotonic Scheduling)

Quetion (since 1973)

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

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

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

**Global Scheduling**
- New task
- Waiting queue

**Partitioned Scheduling**
- Partitioned Scheduling with Task Splitting

**Best Known Results**
- Liu and Layland’s Utilization Bound

**From Uppsala RTAS 2010**
- 69.3

- Liu and Layland’s Utilization Bound
- [OPODIS’08]
- [TPDS’05]
- [ECRTS’03]
- [RTSS’04]
- [RTCSA’06]
Multiprocessor Scheduling

Global Scheduling

Would fixed-priority scheduling e.g. “RMS” work?

Unfortunately “RMS” suffers from the Dhail’s anomaly

Utilization may be “0%”
Schedule the 3 tasks on 2 CPUs using "RMS"
Dhall’s anomali
(M+1 tasks and M processors)

\[ \frac{1}{\epsilon + 1} \]

Multiprocessor Scheduling

Partitioned Scheduling
Multiprocessor Scheduling

Partitioned Scheduling

Resource utilization may be limited to 50%
Partitioned Scheduling

- Partitioning

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>
```

Partitioned Scheduling

- Scheduling
  - reduced to single-processor scheduling on each processor

```
P1  P2  P3
1   4   7
2   5   8
3   6   9
```

100%  77.9%  Yes, schedulable!
Partitioned Scheduling

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

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

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

\[ U(r) = \frac{(M + 1)(0.5 + \varepsilon)}{M} \to 0.5 \]

when \( \varepsilon \to 0 \) and \( M \to +\infty \)
Partitioned Scheduling

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

- Limited Resource Usage

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

necessary condition to guarantee schedulability

\[ U(\tau) = \frac{(M + 1)(0.5 + \varepsilon)}{M} \rightarrow 0.5 \]

when \( \varepsilon \rightarrow 0 \) and \( M \rightarrow +\infty \)
Multiprocessor Scheduling

Partitioned Scheduling

- Partitioning

Diagram: 1, 2, 3, 4, 5, 6, 7, 8, 9

P1, P2, P3
Bin-Packing with Item Splitting

- Resource can be “fully” (better) utilized

![Bin Packing Diagram]

**Depth-First Partitioning Algorithms**

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

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

![Depth-First Partitioning Diagram]
Lakshmanan’s Algorithm [ECRTS’09]

- Sort all tasks in decreasing order of utilization

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

highest util.

lowest util.

Lakshmanan’s Algorithm [ECRTS’09]

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

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

highest util.

P1

lowest util.
Lakshmanan’s Algorithm [ECRTS’09]

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

```
lowest util.  1  2  3  4  5  6  7  highest util.
```

P1

```
lowest util.  1  2  3  4  5  6  7  highest util.
```

P1

```
lowest util.  1  2  3  4  5  6  7  highest util.
```

P1
Lakshmanan’s Algorithm [ECRTS’09]

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

```
1 2 3 4 5 6 7 8

1. Pick up one processor, and assign as many tasks as possible
2. P1
3. 6^1
4. 7
5. 8
6. P2
7. 6^1
8. 7
9. 8
```

Lakshmanan’s Algorithm [ECRTS’09]

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

```
1 2 3 4 5 6 7 8

1. Pick up one processor, and assign as many tasks as possible
2. P1
3. 6^1
4. 7
5. 8
6. P2
7. 6^1
8. 7
9. 8
```
Lakshmanan’s Algorithm [ECRTS’09]

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

```
      highest util.
          P1
          6
          7
          8

      lowest util.
          5
          4
          3
          2
          1
```

```
P2

   6
   5
```

Lakshmanan’s Algorithm [ECRTS’09]

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

```
      highest util.
          P1
          6
          7
          8

      lowest util.
          4
          3
          2
          1
```

```
P2

   6
   5
```
Lakshmanan’s Algorithm [ECRTS’09]

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

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td>3</td>
<td></td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Lakshmanan’s Algorithm [ECRTS’09]

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

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td>3</td>
<td></td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Lakshmanan’s Algorithm [ECRTS’09]

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

- Highest util.
- Lowest util.

Lakshmanan’s Algorithm [ECRTS’09]

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

- Highest util.
- Lowest util.
Lakshmanan’s Algorithm [ECRTS’09]

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

```
<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6^1</td>
<td>3^1</td>
<td>1^2</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>5</td>
<td>2^2</td>
</tr>
</tbody>
</table>
```

key feature: “depth-first” partitioning with decreasing utilization order
Lakshmanan’s Algorithm [ECRTS’09]

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

Utilization Bound: **65%**

Breadth-First Partitioning Algorithms [RTAS 2010]

- Sort all tasks in increasing priority order
Breadth-First Partitioning Algorithms

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

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

```
P1  P2  P3
```

Breadth-First Partitioning Algorithms

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

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

```
P1  P2  P3
```
Breadth-First Partitioning Algorithms

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

![Diagram showing processor utilization]

Breadth-First Partitioning Algorithms

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

![Diagram showing processor utilization]
Breadth-First Partitioning Algorithms

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

lowest priority

```
| 3 | 2 |
```

highest priority

```
| 1 |
```

Breadth-First Partitioning Algorithms

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

lowest priority

```
| 2 | 1 |
```

highest priority

```
| 1 |
```
Breadth-First Partitioning Algorithms

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

lowest priority

$$\begin{array}{ccc}
P1 & P2 & P3 \\
2^1 & 7 & 4 & 6 & 3 & 5 \\
\end{array}$$

highest priority

1

Breadth-First Partitioning Algorithms

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

lowest priority

$$\begin{array}{ccc}
P1 & P2 & P3 \\
2^1 & 7 & 4 & 6 & 3 & 5 \\
\end{array}$$

highest priority

1
Breadth-First Partitioning Algorithms

Select the processor on which the assigned utilization is the lowest

lowest priority

highest priority

Breadth-First Partitioning Algorithms

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

Why is the breadth algorithm better?

breadth-first & increasing priority order

depth-first & decreasing utilization order

Fact for “Light” Tasks whose utilization less than 0.41

For a task set in which each task $\tau_i$ satisfies

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

we have

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

$\Rightarrow$ the task set is schedulable

$\Theta(N) = N(2^{\frac{1}{N}} - 1)$ \quad $N \to \infty$, \quad $\frac{\Theta(N)}{1 + \Theta(N)} \approx 0.41$
Solution for Heavy Tasks
whose utilization larger than 0.41

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

<table>
<thead>
<tr>
<th>Priority</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowest</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Highest</td>
<td>1</td>
</tr>
</tbody>
</table>

P1 P2 P3

Solution for Heavy Tasks

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

<table>
<thead>
<tr>
<th>Priority</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowest</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Highest</td>
<td>1</td>
</tr>
</tbody>
</table>

P1 P2 P3
Solution for Heavy Tasks

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

<table>
<thead>
<tr>
<th>Priority</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Priority</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Solution for Heavy Tasks

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

lowest priority

highest priority

Solution for Heavy Tasks

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

lowest priority

highest priority
Solution for Heavy Tasks

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

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

highest priority

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solution for Heavy Tasks

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

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

highest priority

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<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

  highest priority

Solution for Heavy Tasks

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

  lowest priority

  highest priority
Solution for Heavy Tasks

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

```
P1   P2   P3
12   3   2
  6   4   5
  9   8
```

avoid to split heavy tasks
(that may have low priorities)
FACT

- By introducing the pre-assignment mechanism, we have

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

\[ \Rightarrow \] 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
Implementation

- **Easy!**
  - One timer for each split task
  - Implemented as “task migration”

![Diagram](image)

Further Improvement

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>Schedulable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Using Liu and Layland’s Utilization Bound

Yes, schedulable by our algorithm

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
Exact Analysis

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

\[ R_k = \sum_{T_i < T_k} \left( \frac{R_k}{T_i} \right) C_i + C_k \]

task \( k \) is schedulable iff \( R_k \leq T_k \)

Utilization Bound v.s. Exact Analysis

- On single processors

Utilization bound Test for RMS

Exact Analysis for RMS

[Lehoczky_89]
On Multiprocessors

- Can we do something similar on multiprocessors?

Utilization bound Test the algorithm introduced above

An Improved Algorithm

- Based on the similar idea
  - RMS priority assignment
  - worst-fit partitioning
  - increasing priority order

- Employ Response Time Analysis to determine the maximal workload on each processor
  - more flexible behavior (more difficult to prove ...)

- Same utilization bound
  - Liu and Layland’s utilization bound

- Much better performance
- Pseudo-polynomial
Summary

- Real-time Systems
  - infinite, multi-task, multi-rate...

- On Single-processors
  - Liu and Layland’s Utilization bound

\[ \sum_{i \in T} U_i \leq N(2^{1/N} - 1) \]

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

- Real-time Systems
  - infinite, multi-task, multi-rate ...

- On Single-processors
  - Liu and Layland’s Utilization bound

- On Multiprocessors
  - Global Scheduling
    - Dhall’s Effect

\[ [\text{OPODIS'08}] \]

Multiprocessor Scheduling
Summary

- Real-time Systems
  - infinite, multi-task, multi-rate ...

- On Single-processors
  - Liu and Layland’s Utilization bound

- On Multiprocessors
  - Global Scheduling
    - Dhall’s Effect
  - Partitioned Scheduling
    - Bin-packing

Our Recent Work
- Partitioned with Task Splitting
  - split tasks with high priorities
- Liu and Layland’s utilization bound
References

- [Andersson03ECRTS] Bjorn Andersson, Jan Jonsson: The Utilization Bounds of partitioned and Pfair Static-Priority Scheduling on multiprocessors are 50%. ECRTS 2003
- [Andersson08OPODIS] Bjorn Andersson: Global Static-Priority Preemptive Multiprocessor Scheduling with Utilization Bound 38%. OPODIS 2008
- [Andersson06RTCSA] Bjorn Andersson, Eduardo Tovar: Multiprocessor Scheduling with Few preemption. RTCSA 2006
- [Andersson01RTSS] Bjorn Andersson, Sanjoy K. Baruah, Jan Jonsson: Static-Priority Scheduling on multiprocessors. RTSS 2001
- [Lakshmanan09ECRTS] Karthik Lakshmanan, Ragunathan Rajkumar, John Lehoczky Partitioned Fixed-Priority Preemptive Scheduling for Multi-core Processors. ECRTS 20006