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)

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

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

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

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

- How to check whether all deadlines are met?
Liu and Layland’s Utilization Bound

- Schedulability Analysis

Liu and Layland’s bound:

\[ 3 \times (2^{1/3} - 1) = 77.9\% \]
Liu and Layland’s Utilization Bound

- Schedulability Analysis

Liu and Layland’s bound:

\[ 3 \times (2^{1/3} - 1) = 77.9\% \]

Yes, schedulable!
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
  - ... ...
Find a multiprocessor scheduling algorithm that can achieve Liu and Layland’s utilization bound

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

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

Global Scheduling

Partitioned Scheduling

Partitioned Scheduling with Task Splitting
Multiprocessor Scheduling
Global Partitioned
Fixed Priority
Dynamic Priority
Task Splitting

From Uppsala
RTAS 2010

Liu and Layland’s Utilization Bound

69.3

OPODIS’08
TPDS’05
ECRTS’03
RTSS’04
RTCSA’06

Multiprocessor Scheduling
Global
Partitioned
Task Splitting
Multiprocessor Scheduling

Global Scheduling

Would fixed-priority scheduling e.g. “RMS” work?
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 anomali
Dhall’s anomaly

Schedule the 3 tasks on 2 CPUs using "RMS"
Dhall’s anomali
(M+1 tasks and M processors)

\[ U = \frac{M \epsilon + \frac{1}{1+\epsilon}}{M} \rightarrow 0 \]
when \( \epsilon \rightarrow 0 \) and \( M \rightarrow +\infty \)
Multiprocessor Scheduling

Partitioned Scheduling

CPU 1: 1, 2, 5
CPU 2: 3, 8, 6
CPU 3: 4, 9, 7
Multiprocessor Scheduling

Partitioned Scheduling

Resource utilization may be limited to 50%
Partitioned Scheduling

Partitioning

P1

P2

P3
Partitioned Scheduling

- Partitioning

P1: 1, 2, 3
P2: 4, 5, 6
P3: 7, 8, 9
Partitioned Scheduling

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

![Diagram of partitioned scheduling]

Yes, schedulable!
Partitioned Scheduling

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

- Limited Resource Usage, 50%

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

necessary condition to guarantee schedulability

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

when \( \varepsilon \rightarrow 0 \) and \( M \rightarrow +\infty \)
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 \( M \rightarrow +\infty \)
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 \( M \rightarrow +\infty \)
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 with Task Splitting
Partitioned Scheduling

Partitioning

1 2 3
4 5 6
7 8 9

P1 P2 P3
Bin-Packing with Item Splitting

- Resource can be “fully” (better) utilized
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

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

highest priority
```

P1
Lakshmanan’s Algorithm [ECRTS’09]

- Sort all tasks in decreasing order of utilization

![Diagram showing tasks sorted by utilization](image_url)
Lakshmanan’s Algorithm [ECRTS’09]

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

Diagram:
- Highest util.: 8, 7, 6, 5, 4, 3, 2
- Lowest util.: 1
- Processor P1
Lakshmanan’s Algorithm [ECRTS’09]

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

![Diagram showing processor assignment based on task utilizations.](image-url)
Lakshmananan’s Algorithm [ECRTS’09]

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

![Diagram showing the algorithm]

- Processors:
  - P1
  - Tasks: 6, 5, 4, 3, 2, 7, 8

- Assignment:
  - Lowest util: 1
  - Highest util: 7
Lakshmanan’s Algorithm [ECRTS’09]

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

```
6^2
5
4
3
2
1

P1
6^1
7
8
```

highest util.

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

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

![Diagram showing task assignment to processors P1 and P2 based on utilization, with tasks 6, 7, 8 assigned to P1 and no tasks assigned to P2.]
Lakshmanan’s Algorithm [ECRTS’09]

- Pick up one processor, and assign as many tasks as possible
Lakshmanan’s Algorithm [ECRTS’09]

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

![Diagram showing task assignment](image)
Lakshmanan’s Algorithm [ECRTS’09]

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

```
highest util.

P1
6^1
7
8

P2
4
5
6^2

lowest util.

1
2
3
```
Lakshmanan’s Algorithm [ECRTS’09]

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

P1
- 6
- 7
- 8

P2
- 3
- 4
- 5
- 6

highest util.

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

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

![Diagram showing two processors (P1 and P2) with tasks assigned based on utilization.]

- P1: Tasks 6, 7, 8
- P2: Tasks 2, 3, 4, 5, 6

Utilization:
- Highest utilization: 6
- Lowest utilization: 1

Legend:
- 6^1
- 7
- 8
- 2^2
- 3
- 4
- 5
- 6^2
Lakshmanan’s Algorithm [ECRTS’09]

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

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>6¹</td>
<td>2¹</td>
<td>2²</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>lowest util.</td>
<td>highest util.</td>
<td></td>
</tr>
</tbody>
</table>
Lakshmanan’s Algorithm [ECRTS’09]

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

```
lowest util.                                  P1          P2          P3
  8
  7
  6
  1

P1

highest util.
```

- P1
  - Tasks assigned: 6, 7, 8, 1
- P2
  - Tasks assigned: 3, 4, 5, 2
- P3
  - No tasks assigned
Lakshmanan’s Algorithm [ECRTS’09]

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

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

Diagram:
- Lowest priority: 7, 6, 5, 4, 3, 2
- Highest priority: 1
Select the processor on which the assigned utilization is the lowest

Breadth-First Partitioning Algorithms
Breadth-First Partitioning Algorithms

- Select the processor on which the assigned utilization is the lowest
Breadth-First Partitioning Algorithms

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

```
lowest priority
5
4
3
2

highest priority
1

P1
7
P2
6
P3
```
Breadth-First Partitioning Algorithms

- Select the processor on which the assigned utilization is the lowest
Breadth-First Partitioning Algorithms

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

lowest priority

highest priority

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>
**Breadth-First Partitioning Algorithms**

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

<table>
<thead>
<tr>
<th>Processor</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
</tr>
</tbody>
</table>
Breadth-First Partitioning Algorithms

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

lowest priority

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

highest priority

| 2^2 | 1   |
Breadth-First Partitioning Algorithms

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

Lowest priority

Highest priority

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

Why is the breadth algorithm better?

& increasing priority order
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
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

\[ Y^2 + U_2^2 \leq U_2^1 \]

\[ Y^2 \leq U_2^1 - U_2^2 \]

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

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

$\Rightarrow$ the task set is schedulable

$$\Theta(N) = N(2^{1/N} - 1) \quad N \to \infty, \quad \frac{\Theta(N)}{1 + \Theta(N)} \approx 0.41$$
Theorem

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

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

we have

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

$\Rightarrow$ the task set is schedulable

\[ \Theta(N) = N \left(2^{\frac{1}{N}} - 1\right) \]

$N \to \infty$, \quad \frac{\Theta(N)}{1 + \Theta(N)} \equiv 0.41
Problem of Heavy Tasks

lowest priority

highest priority

P1
P2
P3
Problem of Heavy Tasks

lowest priority

6
5
4
3
2

highest priority

1

5
4

9

9

P1
P2
P3
Problem of Heavy Tasks

highest priority

lowest priority

P1

P2

P3
Problem of Heavy Tasks

lowest priority

highest priority
Problem of Heavy Tasks

lowest priority

highest priority
Problem of Heavy Tasks

lowest priority

highest priority
Problem of Heavy Tasks

lowest priority

highest priority

P1
P2
P3
Problem of Heavy Tasks

lowest priority

highest priority

P1

P2

P3
Problem of Heavy Tasks

lowest priority

highest priority
Problem of Heavy Tasks

lowest priority

highest priority
Problem of Heavy Tasks
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>Priority Level</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest</td>
<td>P1</td>
</tr>
<tr>
<td></td>
<td>P2</td>
</tr>
<tr>
<td>Lowest</td>
<td>P3</td>
</tr>
</tbody>
</table>

- Tasks with priorities:
  1. 6
  2. 5
  3. 4
  4. 3
  5. 2
  6. 1
  7. 8
  8. 9
Solution for Heavy Tasks

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

lowest priority

highest priority

P1

P2

P3
Solution for Heavy Tasks

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

Lowest priority:

- 8
- 7

Highest priority:

- 1
- 2
- 3
- 4
- 5

Tasks assigned:

- P1: 6
- P2: 9
- P3: None
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>Highest</td>
<td>6</td>
</tr>
<tr>
<td>Lowest</td>
<td>7</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></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>
Solution for Heavy Tasks

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

lowest priority

- P1
- P2
- P3

highest priority

5 6
4
3
2
1
7 9
8
Solution for Heavy Tasks

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

P1

P2

P3

lowest priority

highest priority
Solution for Heavy Tasks

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

lowest priority

![Diagram showing priority levels and task assignments]
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)

lowest priority

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

highest priority
Solution for Heavy Tasks

- Pre-assigning the heavy tasks (that may have low priorities)
Solution for Heavy Tasks

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

Avoid to split heavy tasks (that may have low priorities)
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!
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”
Further Improvement

\[ N(2^{\frac{1}{N}} - 1) \]

Schedulable?

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

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

\[ \text{task } k \text{ is schedulable iff } R_k \leq T_k \]
Utilization Bound v.s. Exact Analysis

- On single processors

Utilization bound Test for RMS

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

Exact Analysis for RMS

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

[Lehoczky_89]
On Multiprocessors

Can we do something similar on multiprocessors?

Utilization bound Test
the algorithm introduced above

$100% \quad N(2^{1/N} - 1)$

$100% \quad N(2^{1/N} - 1)$
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 ... ...
Summary

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

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

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

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

- **Real-time Systems**
  - Infinite, multi-task, multi-rate ...

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

- **On Multiprocessors**

![Liu and Layland’s Utilization Bound](image)
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

![Graph showing Liu and Layland’s Utilization Bound with deadline miss and multiprocessor scheduling](Image)
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

\[\text{Multiprocessor Scheduling} \]

\[\text{Liu and Layland’s Utilization Bound} \]

\[\text{[OPODIS’08]} \]

\[\text{[ECRTS’03]} \]
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

\[
\sum \frac{C_i}{T_i} \leq \frac{N}{M} \left(2^{1/N} - 1\right)
\]
\[\Rightarrow \text{the task set is schedulable}\]
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