Low-Overhead Non-Preemptive Scheduling of Real-Time Tasks upon Multiprocessor Platforms

An extensive attempt on time-triggered systems

_Eghonghon-aye Eigbe_

_Supervised by: Mitra Nasri_

July 2020
Background
Real-time system

A system whose correctness depends on

- Logical correctness of results
- Timing correctness of results
Future real-time systems require a lot of computing power

Enter multiprocessor platforms

- Increased computing power
- Ability to exploit parallelism
- Increased scheduling complexity
Periodic **tasks** are system functions that periodically release instances (**jobs**) all through the lifetime of the system.

- **Partitioned**: No tasks can migrate. Performance depends on the assignment of tasks to cores.
- **Semi-partitioned**: Some tasks can migrate.
- **Global**: All tasks can migrate. Lots of overhead.

**Classification of multiprocessor schedulers**
Why non-preemptive scheduling?

- No added context switch cost
- Less cache-related preemption delay (CRPD)
- Less variation in WCET

Non-preemptive scheduling anomalies

Task 1

Task 2

Task 3

4 units

3 units

TU Delft
Non-preemptive scheduling anomalies

Executing for less than WCET can lead to deadline misses
Sustainable scheduling as a cure for some anomalies

Padding computation to ensure WCET always

Using a sustainable online policy

Manipulate offsets and release times to recreate any offline schedule

FIFO with Offset Tuning

Thesis Roadmap
Thesis Roadmap

- What is the true limit of partitioning?
  - A necessary test for partition-ability

- What if partitioning fails?
  - A semi-partitioning scheme

- How efficient are our algorithms at runtime?
  - Extend FIFO-OT to multiprocessor platforms

- Can this improve parallel task scheduling?
  - Decompose the parallel task into sequential parts
What is the true limit of partitioning?
What is the impact of sustainable partitioning?

- Sustainable Partitioning
- Global Scheduling
- Unsustainable Partitioning

% Utilisation

Mostly idle system

Very busy system
What is the impact of sustainable partitioning?

- **Sustainable Partitioning**
- **Global Scheduling**
- **Unsustainable Partitioning**

Assumes the online policy is not sustainable

- Mostly idle system
- Very busy system

Scheduleability Ratio

% Utilisation
What is the impact of sustainable partitioning?

- Sustainable Partitioning
- Global Scheduling
- Unsustainable Partitioning

No non-preemptive semi-partitioning policy (to the best of our knowledge)

Schedulability Ratio

0.2 0.4 0.6 0.8
% Utilisation

Mostly idle system  Very busy system
What is the impact of sustainable partitioning?

- Mostly idle system
- Very busy system

Automotive benchmark task sets

What is the impact of sustainable partitioning?

2 Cores 8 Tasks

- Sustainable Partitioning
- Global Scheduling
- Unsustainable Partitioning

Schedulability Ratio vs. % Utilisation
What is the impact of sustainable partitioning?

4 Cores 12 Tasks

- Sustainable Partitioning
- Global Scheduling
- Unsustainable Partitioning

% Utilisation

Schedulability Ratio
Why do we need a necessary test for partition-ability?

4 Cores 12 Tasks

What is the upper bound on partition-ability?
Building a necessary test for partition-ability

- **Recognize conflicts**
  - Conflicts are subsets of tasks that cannot go on the same core

- **Do a pairwise search through the task set**
  - Pairs of tasks must not be able to completely block each other
    - If they can, there is a conflict
Conflict graph model of a task set

- Each node is a task
- Each edge represents a conflict between two tasks
What can cliques tell us?

![Diagram showing relationships between tasks](image-url)
A necessary test

The size of the largest clique is the minimum number of cores required to partition the task set.
How do we perform against the upper bound?

2 Cores 8 Tasks

- Sustainable Partitioning
- Necessary Test
- Unsustainable Partitioning

Schedulability Ratio

% Utilisation
How do we perform against the upper bound?

2 Cores 8 Tasks

- Sustainable Partitioning
- Necessary Test
- Unsustainable Partitioning

Schedulability Ratio vs. % Utilisation
How do we perform against the upper bound?

4 Cores 12 Tasks

- Sustainable Partitioning
- Necessary Test
- Unsustainable Partitioning

Schedulability Ratio

% Utilisation

TU Delft
How do we perform against the upper bound?

8 Cores 9 Tasks

- Sustainable Partitioning
- Necessary Test
- Unsustainable Partitioning
What if partitioning fails?
Case Study

<table>
<thead>
<tr>
<th>Task Number</th>
<th>Period $T$</th>
<th>Utilisation $U$</th>
<th>Execution Time $C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>0.65</td>
<td>65</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>0.45</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>0.4</td>
<td>8</td>
</tr>
</tbody>
</table>

Conflict Graph

Not partition-able on 2 cores
Case Study

<table>
<thead>
<tr>
<th>Task Number</th>
<th>Period</th>
<th>Utilisation</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>0.65</td>
<td>65</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>0.45</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>0.4</td>
<td>8</td>
</tr>
</tbody>
</table>

Schedulable on 2 cores
Our approach to semi-partitioned scheduling

Partitioning heuristics work by progressively fitting tasks onto cores

But semi partitioning allows some migration

So we progressively fit jobs

Largest objects first principle
Liquid-path scheduling
Liquid-path scheduling

[Diagram showing two liquid-paths with jobs represented by bars.]

TU Delft
Liquid-path scheduling

Do this till we have placed all jobs for one hyperperiod (after this the whole system repeats itself)
Evaluation

Mostly idle system

Very busy system
Evaluation

2 Cores 8 Tasks

- Sustainable Partitioning
- Global Scheduling
- Necessary Test
- Liquid Path Scheduler
- Unsustainable Partitioning

Schedulability Ratio

% Utilisation

TU Delft
Evaluation

4 Cores 12 Tasks

- Sustainable Partitioning
- Global Scheduling
- Necessary Test
- Liquid Path Scheduler
- Unsustainable Partitioning

Schedulability Ratio

% Utilisation

TU Delft
Un-partitionable task sets are schedulable

8 Cores 9 Tasks

- Sustainable Partitioning
- Global Scheduling
- Necessary Test
- Liquid Path Scheduler
- Unsustainable Partitioning

![Graph showing schedulability ratio vs. utilization](image)
Can this improve parallel task scheduling?
The DAG model

DAG node

Execution time

Precedence constraint
DAG scheduling

Directly schedule on a precedence aware scheduler

OR

Decompose the DAG into a set of sequential tasks and then schedule those
We choose decomposition!

- Reduced complexity of the scheduler
- Sequential analysis applies
- Performance is highly dependent on how well we decompose
Our decomposition strategy
The ideal schedule
Sectioning the ideal schedule

- Separate ideal schedule into sections
Assigning deadlines

- Assign deadlines and stretch schedule
Restitching nodes

- Re-stitch nodes for non-preemptive execution
How do we assign deadlines?

- Give sections with more parallelism more slack
- Give sections with larger execution times more slack
- Attempt to keep density of threads less than 1
- Attempt to keep density of sections less than $m$
Scheduling decomposed DAG tasks
Evaluation

- $n=3$, direct scheduling
- $n=3$, decomposition + sustainable

- $n=10$, direct scheduling
- $n=10$, decomposition + sustainable

Schedulability Ratio

- 100%
- 75%
- 50%
- 25%
- 0%

Utilisation (% of Number of Cores)

- 20%
- 40%
- 60%
- 80%
Evaluation

Decomposition outperforms direct scheduling

4 Cores
Decomposition outperforms direct scheduling

8 Cores

- n=5, direct scheduling
- n=5, decomposition + sustainable
- n=10, direct scheduling
- n=10, decomposition + sustainable

Schedulability Ratio

0% 20% 40% 60% 80%

Utilisation (% of Number of Cores)
A sustainable low-overhead scheduler
FIFO as a sustainable scheduler

<table>
<thead>
<tr>
<th>Task Number</th>
<th>Period</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

FIFO order of task releases

| Task 1 | Task 2 | Task 3 | Task 1 | Task 2 | Task 1 | Task 1 | Task 1 | Task 2 | Task 1 |
FIFO has low schedulability

FIFO order of task releases

| Task 1 | Task 2 | Task 3 | Task 1 | Task 2 | Task 1 | Task 1 | Task 1 | Task 2 | Task 1 |

missed
Recreating an offline schedule at runtime

Original

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 3</th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 1</th>
</tr>
</thead>
</table>

Modified

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 1</th>
<th>Task 3</th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 1</th>
</tr>
</thead>
</table>

idle time

TU Delft
Building an offset table

Modified

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 1</th>
<th>Task 3</th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 1</th>
<th>Task 2</th>
<th>Task 1</th>
</tr>
</thead>
</table>

Original

| Task 1 | Task 2 | Task 3 | Task 1 | Task 2 | Task 1 | Task 2 | Task 1 | Task 2 | Task 1 |
Building an offset table

Original

Modified
Building a multiprocessor offset table

- **Mimic partitioning**
  - Only look from the perspective of one core at a time

- **Handle migration**
  - Make it possible to have offsets larger than the period
  - This makes the core skip the jobs that don’t belong to it
Implementation of multiprocessor FIFO-OT

- **Chosen platform**
  - Raspberry pi 3
  - 4 Cores
  - Open source enough for bare metal OS

- **Measure how much overhead is incurred**
  - In terms of storage cost
  - In terms of processing time of the scheduler
Memory consumption

- Offset Table
- Full Table

Memory Required (bytes):
- 12k
- 10k
- 8k
- 6k
- 4k
- 2k
- 0

Number of Tasks:
- 5
- 10
- 15
- 20
- 25

TU Delft
Memory consumption

![Box plot comparing memory consumption for different numbers of tasks]

- **Offset Table**
- **Full Table**

**Memory Required (bytes)**
- 12k
- 10k
- 8k
- 6k
- 4k
- 2k
- 0

**Number of Tasks**
- 5
- 10
- 15
- 20
- 25
Memory consumption

Error bars represent 95% confidence intervals
Scheduling overhead

- Partitioned EDF
- FIFO-OT
- Global EDF

Scheduling Overhead (microseconds)

Number of Tasks
Scheduling overhead

![Box plot graph showing scheduling overhead for different number of tasks and scheduling algorithms: Partitioned EDF, FIFO-OT, and Global EDF.](image)
Scheduling overhead

 Error bars represent 95% confidence intervals
Summary
Summary

What is the true limit of partitioning?
A necessary test for partition-ability

What if partitioning fails?
A semi-partitioning scheme

How efficient are our algorithms at runtime?
Extend FIFO-OT to multiprocessor platforms

Can this improve parallel task scheduling?
Decompose the parallel task into sequential parts

Thesis Roadmap
Conclusions

- **Sustainable partitioning** or **semi-partitioning** as a strategy gives **higher schedulability ratios** with **smaller runtime overheads**.

- The limits of partitioning depends on the amount of conflicts present in the task set.

- **Sustainable partitioning of decomposed DAG tasks** was found to surpass direct scheduling in all examined cases.

- **Sustainable multiprocessor scheduling** is achievable with **scheduling overhead of a partitioned solution** and using **90% less memory** than a table scheduler on average.
Future work

- Handle sporadic task sets
- Coreunner aware partitioning
- A test for task set feasibility
THANK YOU

**Thesis Roadmap**

- **What if partitioning fails?**
  - A necessary test for partition-ability
- **What is the true limit of partitioning?**
- **How efficient are our algorithms at runtime?**
  - Extend FIFO-OT to multiprocessor platforms
- **Can this improve parallel task scheduling?**
  - Decompose the parallel task into sequential parts