PREM-based Optimal Task Segmentation Under Fixed Priority Scheduling

Presented by: Giovani Gracioli
Authors: Muhammad R. Soliman
Rodolfo Pellizzoni
Outline

- Introduction
- Task Model
- Schedulability Analysis
- Task Set Segmentation
- Program Segmentation
- Evaluation
- Conclusion and Future Work
Introduction: MPSoC / PREM

Multi-Processor System-on-Chip

PE 0  PE 1  PE 2
Introduction: **MPSoc / PREM**

**Multi-Processor System-on-Chip**

![Diagram of Multi-Processor System-on-Chip](image)
Introduction: **MPSoc / PREM**

**Multi-Processor System-on-Chip**

- PE 0
- PE 1
- PE 2

- Memory Controller
- Main Memory

- Contention
- Arbitration
Introduction: **MPSoc / PREM**

**Multi-Processor System-on-Chip**

PE 0  PE 1  PE 2

Memory Controller

Main Memory

Contention

Arbitration

**PRedictable Execution Model**

Memory / Computation = Memory + Computation
Introduction: PREM (3-Phase Model)
Introduction: PREM (3-Phase Model)
Introduction: PREM (3-Phase Model)

Load the required code and data to SPM
Introduction: PREM (3-Phase Model)

- **Load**: Load the required code and data to SPM
- **Compute**: Execute the task directly from SPM
- **Unload**:
Introduction: PREM (3-Phase Model)

- **Load**: Load the required code and data to SPM
- **Compute**: Execute the task directly from SPM
- **Unload**: Write-back the modified data
Introduction: PREM (3-Phase Model)

- **Load**: Load the required code and data to SPM
- **Compute**: Execute the task directly from SPM
- **Unload**: Write-back the modified data

A single memory phase is executed at any one time in the system.

Diagram:
- PE 0 SPM
- PE 1 SPM
- PE 2 SPM
- DMA
- Main Memory
**Introduction:** PREM (3-Phase Model)

<table>
<thead>
<tr>
<th>Task Under Analysis</th>
<th>DMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other Tasks</td>
<td></td>
</tr>
</tbody>
</table>

PREM (3-Phase Model)
Introduction: PREM (3-Phase Model)
Introduction: PREM (3-Phase Model)
Introduction: PREM (3-Phase Model)
Introduction: PREM (3-Phase Model)
Introduction: PREM (3-Phase Model)

- Segmentation:
  - Large code / data footprint → do not fit in SPM.
  - Data accesses are input-dependent → only known at run-time
Introduction: PREM (3-Phase Model)

- Segmentation:
  - Large code / data footprint \(\rightarrow\) do not fit in SPM.
  - Data accesses are input-dependent \(\rightarrow\) only known at run-time

Contribution: How to compile a program based on PREM?
Introduction: Processor / Memory Schedule
Introduction: Processor / Memory Schedule

Partitioned, Fixed priority, Sporadic Tasks
Introduction: **Processor / Memory Schedule**

- Partitioned, Fixed priority, Sporadic Tasks

Diagram:
- PE 0 SPM
- PE 1 SPM
- DMA
- Main Memory
- TDMA: 0 1 0 1 0
Introduction: Processor / Memory Schedule

- Partitioned, Fixed priority, Sporadic Tasks
- Load OR unload one SPM partition
- TDMA
- DMA
- Main Memory
- PE 0 SPM
- PE 1 SPM
Task Model

- Sequential, conditional PREM tasks
- Non-preemptive segment execution
- Each task has a period $T_i$ and a deadline $D_i \leq T_i$
- Fixed memory time $\Delta$ to load/unload each segment
  - For a TDMA slot $\sigma$ and $M$ processors: $\Delta = (M+1) \times \sigma$
Task Model: **DAG Representation**

\[ G = (S, E) \]
Task Model: DAG Representation

\[ G = (S, E) \]

\[ t_s = 5 \]
\[ t_s = 6 \]
\[ t_s = 9 \]
\[ t_s = 11 \]
\[ t_s = 12 \]
\[ t_s = 2 \]
\[ t_s = 4 \]
\[ t_s = 5 \]
Task Model: DAG Representation

\[ G = (S, E) \]

\[ l = \max(t_s, \Delta = 5) \]
Task Model: Paths

\[ \begin{align*}
S^0 & \quad l = 5 \\
S^1 & \quad l = 9 \\
S^2 & \quad l = 11 \\
S^4 & \quad l = 12 \\
S^5 & \quad l = 5 \\
S^6 & \quad l = 5 \\
S^7 & \quad l = 5 \\
\end{align*} \]

\[ p \quad S^1 \quad S^2 \quad l = 2 \quad L = 20 \quad end = 11 \]
Task Model: **Paths**

- **$P$**
  - $s^0$, $s^1$, $s^2$, $s^7$
  - $I = 4$, $L = 30$, $end = 5$

- **$P'$**
  - $s^0$, $s^3$, $s^4$, $s^7$
  - $I = 4$, $L = 28$, $end = 5$

- **$P''$**
  - $s^0$, $s^3$, $s^5$, $s^6$, $s^7$
  - $I = 5$, $L = 26$, $end = 5$
Task Model: Path/DAG Domination

- If neither $P' \geq P$ nor $P \geq P'$, $P'$ and $P$ are incomparable.
- A DAG can be characterized by its dominating maximal paths G.C which replaces the concept of WCET for sequential programs.
- If it is possible to choose between two paths, a dominated path is (better) than the dominating path.
Task Model: Path/DAG Domination

\[ P' \geq P \quad \text{if} \quad P'.I \geq P.I \quad \& \quad P'.L \geq P.L \quad \& \quad P'.end \leq P.end \]

- If neither \( P' \geq P \) nor \( P \geq P' \), \( P' \) and \( P \) are incomparable.
- A DAG can be characterized by its dominating maximal paths G.C which replaces the concept of WCET for sequential programs.
- If it is possible to choose between two paths, a dominated path is (better) than the dominating path.

\[ G' \geq G \quad \text{if} \quad \forall P \in G, \exists P' \in G': \quad P' \geq P \]

- If neither \( G' \geq G \) nor \( G \geq G' \), \( G' \) and \( G \) are incomparable.
- If it is possible to choose between two DAGs, a dominated path is (better) than the dominating path.
Schedulability Analysis

\[ \tau_1 > \tau_2 > \tau_3 \]
Schedulability Analysis

\[ \tau_1 > \tau_2 > \tau_3 \]
Schedulability Analysis

\( \tau_1 > \tau_2 > \tau_3 \)

\( R_3(P) \)
Schedulability Analysis

\[ \tau_1 > \tau_2 > \tau_3 \]

\[ R_3(P) = P.L - P.end \]
Schedulability Analysis

\[ \tau_1 > \tau_2 > \tau_3 \]

\[ R_3(P) = P.L - P.end + \text{Inter}_3(R_3(P)) \]
Schedulability Analysis

\[
\tau_1 > \tau_2 > \tau_3
\]

\[
R_3(P) = P.L - P.end + \text{Inter}_3(R_3(P)) + (P.I + 1) \times l^l_{\text{max}}
\]
Schedulability Analysis

\[ \tau_1 > \tau_2 > \tau_3 \]

\[
R_3(P) = P.L - P.end + \text{Inter}_3(R_3(P)) + (P.I + 1) \times l_3^{\text{max}}
\]

\[ R_3(P) \leq D_3 - P.end \]
Schedulability Analysis

\[ \tau_1 > \tau_2 > \tau_3 \]

\[ R_3(P) = P.L - P.end + \text{Inter}_3(R_3(P)) + (P.I + 1) \cdot l_{3}^{\text{max}} \]

\[ \forall P \in G_3.C: R_3(P) \leq D_3 - P.end \]
Schedulability Analysis

\[ \tau_1 > \tau_2 > \tau_3 \]

- \( R_3(P) \) depends on \( l_3^{\text{max}} \) parameter only from lower priority tasks.
- If the higher priority interference is known and the task is segmented, a maximum length \( l_{\text{max}} \) can be forced on the lower priority tasks to preserve the schedulability of the task.

\[
R_3(P) = P.L - P.end + \text{Inter}_3(R_3(P)) + (P.I + 1) \times l_3^{\text{max}}
\]

\[ \forall P \in G_3, C; \quad R_3(P) \leq D_3 - P.end \]
Task Set Segmentation

Set $l_{\text{max}} = \infty$
Task Set Segmentation

Set $l_{max} = \infty$

Iterate over tasks from higher to lower priority
Task Set Segmentation

Set $l_{\text{max}} = \infty$

Iterate over tasks from higher to lower priority

Segment the task using $l_{\text{max}}$
Task Set Segmentation

Set $l_{max} = \infty$

Iterate over tasks from higher to lower priority

Segment the task using $l_{max}$

Compute $l_{max}$ for the next task
Task Set Segmentation

Set $l_{\text{max}} = \infty$

Iterate over tasks from higher to lower priority

Segment the task using $l_{\text{max}}$

Compute $l_{\text{max}}$ for the next task

If $l_{\text{max}} \leq 0$

Continue

Return Failure

Yes

No
Task Set Segmentation

- Set $l_{\text{max}} = \infty$
- Iterate over tasks from higher to lower priority
  - Return Success
- Segment the task using $l_{\text{max}}$
- Compute $l_{\text{max}}$ for the next task
  - If $l_{\text{max}} \leq 0$
    - Continue
  - If $l_{\text{max}} > 0$
    - Return Failure
Task Set Segmentation

- The paper proves that this algorithm results in an optimal task set segmentation that optimizes the schedulability.
- The program segmentation algorithm must preserve the optimality of the system by generating a set of DAGs that contains the best (dominated) DAGs from all the possible DAGs of the program.
Program Segmentation: **Structure (main)**

Region-based tree program structure
Program Segmentation: **Structure (main)**

Region-based tree program structure

*Sub-graph with single entry and single exit*
Program Segmentation: **Structure (main)**

```c
main() {
    X1;
    for(...) {
        X2;
    }
    f(...);
    X3;
}
```

Region-based tree program structure

*Sub-graph with single entry and single exit*
Program Segmentation: **Structure (main)**

```c
main() {
    X1;
    for(...) {
        X2;
    }
    f(...);
    X3;
}
```

Region-based tree program structure

*Sub-graph with single entry and single exit*
Program Segmentation: Structure (main)

```c
main() {
    X1;
    for(…){
        X2;
    }
    f(…);
    X3;
}
```

Region-based tree program structure

Sub-graph with single entry and single exit
Program Segmentation: **Structure (main)**

```c
main() {
    X1;
    for(...) {
        X2;
        f(...);
    }
    X3;
}
```

**Region-based tree program structure**

*Sub-graph with single entry and single exit*
Program Segmentation: Structure (main)

main() {
    X1;
    for(...) {
        X2;
        X2;
    }
    f(...);
    X3;
}
Program Segmentation: **Structure (main)**

```c
main() {
    X1;
    for(…){
        X2;
    }
    f(…);
    X3;
}
```

Region-based tree program structure

Sub-graph with single entry and single exit
Program Segmentation: **Structure (main)**

```c
main() {
    X1;
    for(...){
        X2;
    }
    f(...);
    X3;
}
```

Region-based tree program structure

*Sub-graph with single entry and single exit*
Program Segmentation: **Structure (main)**

```c
main() {
    X1;
    for(...) {
        X2;
    }
    f(...);
    X3;
}
```

Region-based tree program structure

Sub-graph with single entry and single exit
Program Segmentation: **Structure** (main)

```
main() {
  X1;
  for(...) {
    X2;
  }
  f(...);
  X3;
}
```

Region-based tree program structure

*Sub-graph with single entry and single exit*
Program Segmentation: **Structure** (\( f \))

\[
\begin{align*}
\text{f()} \{ \\
    \text{Y1;} \quad r_f^1 \\
    \text{if(..) } \quad r_f^2 \\
    \text{for(...)} \quad r_f^4 \\
    \text{Y2;} \quad r_f^6 \\
    \text{else} \quad r_f^5 \\
    \text{Y3;} \quad r_f^5 \\
    \text{Y4;} \quad r_f^3 \\
\}
\end{align*}
\]
Program Segmentation: Loop Transformations
Program Segmentation: **Loop Transformations**

- Loop Splitting
- Loop Tiling
Program Segmentation: Loop Transformations

Loop Splitting

Loop Tiling
Program Segmentation: Loop Transformations

Loop Splitting

Loop Tiling
Program Segmentation: Final Trees

main()

\[ r^0 \rightarrow r^1 \rightarrow r^2 \rightarrow r^3 \rightarrow r^4 \]

\[ r^1 \rightarrow r^2 \rightarrow r^2_m \rightarrow r^3 \rightarrow f(\ldots) \rightarrow r^4 \]

f()

\[ r^0_f \rightarrow r^1_f \rightarrow r^2_f \rightarrow r^3_f \]

\[ r^1_f \rightarrow r^2_f \rightarrow r^2_m \rightarrow r^3_f \rightarrow f(\ldots) \rightarrow r^4_f \]

\[ r^1_f \rightarrow r^2_f \rightarrow r^2_m \rightarrow r^3_f \rightarrow f(\ldots) \rightarrow r^4_f \]

\[ r^1_f \rightarrow r^2_f \rightarrow r^2_m \rightarrow r^3_f \rightarrow f(\ldots) \rightarrow r^4_f \]

\[ r^1_f \rightarrow r^2_f \rightarrow r^2_m \rightarrow r^3_f \rightarrow f(\ldots) \rightarrow r^4_f \]

\[ r^1_f \rightarrow r^2_f \rightarrow r^2_m \rightarrow r^3_f \rightarrow f(\ldots) \rightarrow r^4_f \]

\[ r^1_f \rightarrow r^2_f \rightarrow r^2_m \rightarrow r^3_f \rightarrow f(\ldots) \rightarrow r^4_f \]

\[ r^1_f \rightarrow r^2_f \rightarrow r^2_m \rightarrow r^3_f \rightarrow f(\ldots) \rightarrow r^4_f \]

\[ r^1_f \rightarrow r^2_f \rightarrow r^2_m \rightarrow r^3_f \rightarrow f(\ldots) \rightarrow r^4_f \]
Program Segmentation: **Valid Segmentation**

Assign each region or a sequence of regions to a segment

- Footprint
- Length
- Compilation
Assign each region or a sequence of regions to a segment

- Footprint
- Length
- Compilation

*Code + Data $\rightarrow$ SPM*
Program Segmentation: **Valid Segmentation**

Assign each region or a sequence of regions to a segment

- **Footprint**: `Code + Data \rightarrow SPM`.
- **Length**: `Segment length < l_{\text{max}}`.
- **Compilation**.
Program Segmentation: **Valid Segmentation**

Assign each region or a sequence of regions to a segment

- **Footprint**: Code + Data $\rightarrow$ SPM
- **Length**: Segment length $< l_{\text{max}}$
- **Compilation**: Regions
Program Segmentation: **Segmented Tree (1)**

- A tree where each node is a segment path.
- It is obtained by substituting region sequences with a set of paths.
- A segmented tree generates a set of DAGs where each DAG is constructed by taking one path out of each path set.
Program Segmentation: **Segmented Tree (2)**

Diagram of a segmented tree with nodes labeled as $r^0$, $r^1$, $r^2_p$, $r^2_m$, $r^2_s$, $r^3$, $r^4$, $r^1_f$, $r^2_f$, $r^3_f$, $r^4_f$, $r^5_f$, $r^t_f$, and $r^{last}_f$. The tree structure shows the relationships between these nodes, with arrows indicating the direction of the segments.
Program Segmentation: Segmented Tree (2)
Program Segmentation: **Segmented Tree (2)**

\[ l = 28 \quad l = 35 \]

\[ l = 23 \]

\[ r^0 \]

\[ f(\ldots) \]

\[ r^1 \quad r^2_p \quad r^2_m \quad r^2_s \quad r^3 \quad r^4 \]

\[ r^1_f \quad r^2_f \quad r^3_f \quad r^4_f \quad r^5_f \]

\[ r^t_f \quad r^{last}_f \]
Program Segmentation: \textbf{Segmented Tree (2)}

\begin{itemize}
  \item \( r_0 \)
  \item \( r_1 \)
  \item \( r^2 \)
  \item \( r^2_m \)
  \item \( r^2_s \)
  \item \( r^3 \)
  \item \( r^4 \)
  \item \( f(\ldots) \)
  \item \( r^1_f \)
  \item \( r^2_f \)
  \item \( r^3_f \)
  \item \( r^4_f \)
  \item \( r^5_f \)
  \item \( r^t_f \)
  \item \( r^{\text{last}}_f \)
\end{itemize}

\begin{itemize}
  \item \( l = 28 \) \rightarrow \( l = 35 \)
  \item \( l = 23 \)
  \item \( l = 35 \) \rightarrow \( l = 35 \) \rightarrow \( l = 23 \)
  \item \( l = 32 \) \rightarrow \( l = 32 \) \rightarrow \( l = 23 \)
\end{itemize}
Program Segmentation: **Segmented Tree (2)**
Program Segmentation: Segmented Tree (3)
Program Segmentation: **Algorithms (1)**

\[ \tau_1 > \tau_2 > \tau_3 \]

\[ R_3(P) = P.L - P.end + \text{Inter}_3(R_3(P)) + (P.I + 1) \cdot l_3^{max} \]
Program Segmentation: Algorithms (1)

\[ \tau_1 > \tau_2 > \tau_3 \]

\[ R_3(P) = P.L - P.end + Inter_3(R_3(P)) + (P.I + 1) \times l_{3\text{max}} \]

Minimize P.L
Program Segmentation: **Algorithms (1)**

\[ \tau_1 > \tau_2 > \tau_3 \]

\[ R_3(P) = P.L - P.end + \text{Inter}_3(R_3(P)) + (P.I + 1) \cdot l_3^{\text{max}} \]

- **Minimize** \( P.L \)
- **Maximize** \( P.end \)
Program Segmentation: Algorithms (1)

\[ R_3(P) = P.L - P.end + \text{Inter}_3(R_3(P)) + (P.I + 1) \times l_3^{\text{max}} \]

Minimize \( P.L \)
Maximize \( P.end \)
Minimize \( P.I \)
Program Segmentation: Algorithms (1)

\[ R_3(P) = P.L - P.end + \text{Inter}_3(R_3(P)) + (P.I + 1) \times l_{\text{max}}^3 \]

- Minimize \( P.L \)
- Maximize \( P.end \)
- Minimize \( P.I \)

Based on path domination \( \rightarrow \) keep dominated paths
Program Segmentation: **Algorithms (2)**

- The segmentation algorithms generates the possible paths for the segmented tree based on the constraints.
- The generated paths are filtered using path domination to eliminate the dominating (worse) paths.
- The DAGs generated from the segmented tree are filtered using the DAG domination to keep the dominated (better) DAGs.
- Pruning conditions are used to avoid enumerating all the DAGs which is very time consuming due to the parameterized split/tile transformations.
Evaluation

- The segmentation framework is implemented using LLVM compiler.
- Simple MIPS processor model: 5-stage pipeline, no branch prediction.
- Vary the SPM size between 4 kB to 512 kB exponentially.
- Multiple benchmarks from different suites.
- Test for system utilization between 0.2 – 0.95.
- For each system utilization → 100 task set, 5-15 tasks / task set.
- Results reported in terms of system schedulability.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Suite</th>
<th>LOC</th>
<th>Data(B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>adpcm_dec</td>
<td>TACLEBench</td>
<td>476</td>
<td>404</td>
</tr>
<tr>
<td>jpeg_transupp</td>
<td>TACLEBench</td>
<td>474</td>
<td>3459</td>
</tr>
<tr>
<td>fft</td>
<td>TACLEBench</td>
<td></td>
<td>24572</td>
</tr>
<tr>
<td>compress</td>
<td>UTDSP</td>
<td>131</td>
<td>136448</td>
</tr>
<tr>
<td>lpc</td>
<td>UTDSP</td>
<td>249</td>
<td>8744</td>
</tr>
<tr>
<td>spectral</td>
<td>UTDSP</td>
<td>340</td>
<td>4584</td>
</tr>
<tr>
<td>disparity</td>
<td>CortexSuite</td>
<td>87</td>
<td>2704641</td>
</tr>
</tbody>
</table>
## Evaluation (2)

<table>
<thead>
<tr>
<th>Proposed</th>
<th>Optimal</th>
<th>Length</th>
<th>Footprint</th>
<th>Compilation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$l_{\text{max}}$</td>
<td>$\text{SPM size}$</td>
<td>$\text{Regions}$</td>
</tr>
</tbody>
</table>
## Evaluation (2)

<table>
<thead>
<tr>
<th>Proposed</th>
<th>Length</th>
<th>Footprint</th>
<th>Compilation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal</td>
<td>$l_{\text{max}}$</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Optimal</td>
<td>$l_{\text{max}}$</td>
<td>SPM size</td>
<td>Regions</td>
</tr>
</tbody>
</table>
## Evaluation (2)

<table>
<thead>
<tr>
<th>Proposed</th>
<th><strong>Length</strong></th>
<th><strong>Footprint</strong></th>
<th><strong>Compilation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal</td>
<td>$l_{\text{max}}$</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Optimal</td>
<td>$l_{\text{max}}$</td>
<td><em>SPM size</em></td>
<td>Regions</td>
</tr>
<tr>
<td>Heuristic</td>
<td>Fixed $l_{\text{max}}$</td>
<td><em>SPM size</em></td>
<td>Regions</td>
</tr>
</tbody>
</table>
## Evaluation (2)

<table>
<thead>
<tr>
<th>Proposed</th>
<th>Length</th>
<th>Footprint</th>
<th>Compilation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal</td>
<td>$l_{\text{max}}$</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Optimal</td>
<td>$l_{\text{max}}$</td>
<td>SPM size</td>
<td>Regions</td>
</tr>
<tr>
<td>Heuristic</td>
<td>Fixed $l_{\text{max}}$</td>
<td>SPM size</td>
<td>Regions</td>
</tr>
<tr>
<td>Greedy</td>
<td>None</td>
<td>SPM size</td>
<td>Regions</td>
</tr>
</tbody>
</table>
Evaluation (3)
Conclusion & Future Work

Conclusion

- The paper proposes a segmentation framework based on LLVM compiler to automatically generate PREM-compatible code for sequential programs running on a general purpose processor.
- An optimal task set segmentation algorithm is derived under fixed-priority scheduling for fixed-size DMA time.
- The evaluation shows that the proposed algorithm outperforms both greedy and heuristic algorithms.

Future Work

- The framework can be extended to other PREM-based scheduling schemes.
- The framework can also consider other task and platform models, especially parallel tasks.
For questions, please contact the authors:

mrefaat@uwaterloo.ca
rpellizz@uwaterloo.ca

Thank you