A Study on
C-group controlled big.LITTLE Architecture

Renesas Electronics Corporation
New Solutions Platform Business Division
Renesas Solutions Corporation
Advanced Software Platform Development Department

Tetsuya Nakagawa
Magnus Damm

2013/5/30      Rev. 1.00

© 2013 Renesas Electronics Corporation. All rights reserved.
Content

- Introduction
- big.LITTLE Architecture
- Approach 1: A cluster migration using C-group
- Approach 2: A Scalable virtual processor using C-group
- Evaluation
- Conclusion
Introduction

- Renesas has developed big.LITTLE architecture based SoCs and been working on the software solution
  - Existing Renesas SoCs: APE6 (Shown in MWC2013), R-CarH2
  - Both of them have CA15 x4 + CA7 x 4, Oct cores

- big.LITTLE is ARM architecture and they proposed 3 use models. But there is no established software solution although ARM and many partners are making much effort
  - Kernel approaches for 2 of 3 ARM use models
  - Some proposal based on existing techniques

- Renesas as an ARM partner propose one powerful solution exploiting existing techniques and give its initial evaluation result on real silicon in this presentation
big.LITTLE Architecture and Solutions
big.LITTLE Architecture and Solutions

- **big.LITTLE Architecture**
  - Heterogeneous Multi core architecture with performance oriented “big Core” and energy conscious “LITTLE Core” proposed by ARM.

![Diagram of big.LITTLE Architecture](image-url)
big.LITTLE Architecture and Solutions

- ARM proposes 3 Use models

**Cluster migration**
Either one of big (CA15×4) cluster or LITTLE (CA4×4) cluster is active

- Cluster “big”
- Cluster “LITTLE”

**Pros:** Easy to control
**Cons:** Always only the half of all physical cores is active

**In-Kernel Switcher**
Switching from big to LITTLE or LITTLE to big in CPU pair-wise (big×1 + LITTLE×1)

- big
- LITTLE

**Pros:** Product quality Linux solution exists
**Cons:** Always only the half of all physical cores is active

**big.LITTLE MP**
Kernel takes care of heterogeneous multi processors

- big
- LITTLE

**Pros:** All the existing physical cores are active if required
**Cons:** Takes time to develop the kernel
Three challenging issues in big.LITTLE MP

- big.LITTLE MP is the most powerful use model which is expected to be the final solution

- Three challenging issues in big.LITTLE MP
  
  **Issue 1: Optimal process placement**
  Dynamically place computationally intensive processes on big cores and less intensive ones on LITTLE cores.

  **Issue 2: Exploitation of additional input parameters**
  Kernel needs to take care of additional input parameters such as chip temperature and Performance Index (Performance oriented or power conscious) in addition to CPU load.

  **Issue 3: Consolidation with existing Power management framework**
  Apply optimal Dynamic Voltage and Frequency Scaling on all the big cores and LITTLE cores.
Use Model comparison

- Solving all the 3 Issues at a time is a difficult “multi-dimensional optimization problem” particularly when all physical cores are active.

<table>
<thead>
<tr>
<th></th>
<th>Cluster Migration (Original)</th>
<th>In-Kernel Switcher</th>
<th>big.LITTLE MP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue 1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Issue 2</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Issue 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>All physical cores are active?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Status</td>
<td>Not maintained</td>
<td>Used in a product</td>
<td>Work In Progress (Not in 3.10)</td>
</tr>
</tbody>
</table>
Here we propose two C-group based approaches which overcome all the three issues.

Approach 1:
- A Cluster migration using C-group
- Enhanced comparing to the original Cluster migration use model by exploiting parameters such as Performance Index and Temperature

Approach 2:
- Based on Approach 1
- Introduce “a scalable virtual processor” in place of “Cluster migration” to enable the use of all physical cores at the same time
Approach 1: A cluster migration using C-group
Approach 1: Optimal Process Placement (Issue1)

- C-group assigns “process groups” to pre-defined CPU sets
  - “Slow Process group” is statically assigned to LITTLE cluster
  - User space C-group governor migrates the other processes in “Dynamic Process group” between big and LITTLE clusters

Dynamic Process group on LITTLE cluster

[Diagram showing process placement on LITTLE cluster]

Dynamic Process group on big cluster

[Diagram showing process placement on big cluster]
Approach 1: A cluster migration using C-group

- Standard kernel interfaces are used for monitoring and control
  - /proc/cpuinfo is used to detect CA15 and CA7
  - /proc/stat is used to determine per-CPU usage
  - /sys/class/thermal/.. provides temperature information
  - /sys/devices/system/cpu/... can be used with CPU Hotplug

- A “dynamic” C-group cpuset switches between CA15 and CA7
  - /dev/cpuset/dynamic/cpuset.cpus is defined to switch cluster
  - The number of CPU cores is scaled depending on Performance Index
  - Any number of CA15 or any number of CA7 can be used

- C-group governor monitors Thermal sensor state and reduces CA15 usage
  - /dev/cpuset/dynamic/cpuset.cpus is also used stay on CA7
Approach 1: A cluster migration using C-group

- Dynamically assign `cpuset.cpus` on “Dynamic Process Group” to a cluster

```
User Space
Dynamic Process Group
/dev/cpuset/dynamic/cpuset.cpus

Kernel
Scheduler
Device Driver
Power Management

Firmware

HW
Big CPUs
CPU#0
CPU#1
CPU#2
CPU#3

Little CPUs
CPU#0
CPU#1
CPU#2
CPU#3

big cluster
LITTLE cluster
1 x CA7
```
Approach 1: C-group governor algorithm (Issue2)

- C-group governor exploits “Temperature and Performance Index”
- “Temperature and Performance Index” determine “Dynamic process placement” and the number of core

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Performance Index</th>
<th>Dynamic Process</th>
<th>Slow Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>less than 60 deg C</td>
<td>0% - 20%</td>
<td>CA7 x 1</td>
<td>CA7 x 1</td>
</tr>
<tr>
<td></td>
<td>20% - 30%</td>
<td>CA7 x 2</td>
<td>CA7 x 2</td>
</tr>
<tr>
<td></td>
<td>30% - 40%</td>
<td>CA7 x 3</td>
<td>CA7 x 3</td>
</tr>
<tr>
<td></td>
<td>40% - 50%</td>
<td>CA7 x 4</td>
<td>CA7 x 4</td>
</tr>
<tr>
<td></td>
<td>50% - 60%</td>
<td>CA15 x 1</td>
<td>CA7 x 4</td>
</tr>
<tr>
<td></td>
<td>60% - 70%</td>
<td>CA15 x 2</td>
<td>CA7 x 4</td>
</tr>
<tr>
<td></td>
<td>70% - 80%</td>
<td>CA15 x 3</td>
<td>CA7 x 4</td>
</tr>
<tr>
<td></td>
<td>80% - 100%</td>
<td>CA15 x 4</td>
<td>CA7 x 4</td>
</tr>
<tr>
<td>Larger than or equal to 60 deg C</td>
<td>0% - 100%</td>
<td>CA7 x 1</td>
<td>CA7 x 1</td>
</tr>
</tbody>
</table>
Approach 1: Per cluster CPUFreq Scaling(Issue3)

- In current SoCs, CPUs in one cluster share the same clock and voltage control and DVFS can be applied in each cluster independently.
Approach 1 solves all the three issues,

- **Issue1**: Optimal process placement is taken care of by cluster switch of "Dynamic Process Group"
- **Issue2**: Additional input parameters such as temperature and Performance Index are exploited by C-Group Governor.
- **Issue3** is solved per cluster base.

---

<table>
<thead>
<tr>
<th></th>
<th>Cluster Migration (Original)</th>
<th>In-Kernel Swither</th>
<th>big.LITTLE MP</th>
<th>Approach 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Issue 1</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Issue 2</strong></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Issue 3</strong></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>All physical cores are active?</strong></td>
<td></td>
<td></td>
<td>✓</td>
<td>Partially Yes</td>
</tr>
<tr>
<td><strong>Status</strong></td>
<td>Not maintained</td>
<td>Used in a product</td>
<td>Work In Progress (Not in 3.10)</td>
<td>Available with the current kernel</td>
</tr>
</tbody>
</table>

**But for "Dynamic Process Group", all the physical cores, 8 in this case, can not be assigned.**
Approach 2:  
A scalable virtual processor using C-group
Approach 2: A scalable virtual processor (Issue1)

- Introduce a scalable virtual processor and map the multi dimensional optimization problem onto one dimensional problem.
  1. Heterogeneous multi core -> a scalable virtual processor
  2. Consolidation with one dimensional CPUfreq scaling

- One example of scalable virtual processor (Vi: i=1-12)

\[
\begin{align*}
\begin{bmatrix}
V_1 \\
V_2 \\
V_3 \\
V_4 \\
V_5 \\
V_6 \\
V_7 \\
V_8 \\
V_9 \\
V_{10} \\
V_{11} \\
V_{12}
\end{bmatrix}
&=
\begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\times
\begin{bmatrix}
B_1 \\
B_2 \\
B_3 \\
B_4 \\
L_1 \\
L_2 \\
L_3 \\
L_4 \\
B_1 + B_2 + B_3 + L_1 + L_2 + L_3 + L_4 \\
B_1 + B_2 + B_3 + L_1 + L_2 + L_3 + L_4 \\
B_1 + B_2 + B_3 + B_4 + L_1 + L_2 + L_3 + L_4 \\
B_1 + B_2 + B_3 + B_4 + L_1 + L_2 + L_3 + L_4 \\
B_1 + B_2 + B_3 + B_4 + L_1 + L_2 + L_3 + L_4 \\
B_1 + B_2 + B_3 + B_4 + L_1 + L_2 + L_3 + L_4 \\
B_1 + B_2 + B_3 + B_4 + L_1 + L_2 + L_3 + L_4 \\
B_1 + B_2 + B_3 + B_4 + L_1 + L_2 + L_3 + L_4 \\
\end{bmatrix}
\end{align*}
\]
Approach 2: A scalable virtual processor (Issue1)

Another example of scalable virtual processor (Vi: i=1-8)

\[
\begin{pmatrix}
V_1 \\
V_2 \\
V_3 \\
V_4 \\
V_5 \\
V_6 \\
V_7 \\
V_8
\end{pmatrix}
= 
\begin{pmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
B_1 \\
B_2 \\
B_3 \\
B_4 \\
L_1 \\
L_2 \\
L_3 \\
L_4
\end{pmatrix}
= 
\begin{pmatrix}
L_1 \\
L_1 + L_2 \\
L_1 + L_2 + L_3 \\
L_1 + L_2 + L_3 + L_4 \\
B_1 + L_1 + L_2 + L_3 + L_4 \\
B_1 + B_2 + L_1 + L_2 + L_3 + L_4 \\
B_1 + B_2 + B_3 + L_1 + L_2 + L_3 + L_4 \\
B_1 + B_2 + B_3 + B_4 + L_1 + L_2 + L_3 + L_4
\end{pmatrix}
Scalable virtual processor using C-group

- Dynamically assign `cpuset cpus` on “Dynamic Process Group” to an adequate scalable virtual processor state $V_i$ according to its load (hereafter we call “system load”)

```
V_1
V_2
V_3
V_4
V_5
V_6
V_7
V_8
V_9
V_{10}
V_{11}
V_{12}
```

$\text{Cgrp0up Governor}$

$\text{Slow Process Group}$

/\text{dev/cpuset/dynamic/cpuset.cpus}

$\text{Scheduler}$

$\text{Device Driver}$

$\text{Power Management}$

$\text{Boot}$

$\text{Big CPUs}$

- CPU#0
- CPU#1
- CPU#2
- CPU#3

$\text{Little CPUs}$

- CPU#0
- CPU#1
- CPU#2
- CPU#3
Approach 2: C-group governor algorithm (Issue 2)

- CPU number scaling is done by selecting a scalable virtual processor state.
- Dynamic process placement is done based on all of temperature, Performance Index and System load.

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Performance Index</th>
<th>System Load</th>
<th>Scaling Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;=60 deg C</td>
<td>-</td>
<td>-</td>
<td>Choose V&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td>&lt;60 deg C</td>
<td>&gt;=50%</td>
<td>&gt;=70%</td>
<td>V&lt;sub&gt;i&lt;/sub&gt; -&gt; V&lt;sub&gt;i+1&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30% &lt;= or &lt;70%</td>
<td>NOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;30%</td>
<td>V&lt;sub&gt;i&lt;/sub&gt; -&gt; V&lt;sub&gt;i-1&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;50%</td>
<td>30% &gt;=</td>
<td>NOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;30%</td>
<td>V&lt;sub&gt;i&lt;/sub&gt; -&gt; V&lt;sub&gt;i-1&lt;/sub&gt;</td>
</tr>
</tbody>
</table>
CPUfreq consolidation in one dimension (Issue3)

- CPUfreq consolidation is realized using C-group governor also as “CPUfreq User Space Governor”.

- CPUfreq scaling is applied to “Virtual Frequency” of “Scalable Virtual Processor” where its state does not change.

- Standard kernel interfaces can be used for CPUfreq scaling.

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Performance Index</th>
<th>System Load</th>
<th>Scaling Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;=60 deg C</td>
<td>-</td>
<td>-</td>
<td>Choose V&lt;sub&gt;1&lt;/sub&gt;</td>
</tr>
<tr>
<td>&lt;60 deg C</td>
<td>&gt;=50%</td>
<td>&gt;70%</td>
<td>V&lt;sub&gt;i&lt;/sub&gt; -&gt; V&lt;sub&gt;i+1&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30% &lt;= or &lt; 70%</td>
<td>CPUfreq scaling on V&lt;sub&gt;i&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;30%</td>
<td>V&lt;sub&gt;i&lt;/sub&gt; -&gt; V&lt;sub&gt;i-1&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;50%</td>
<td>CPUfreq scaling on V&lt;sub&gt;i&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30% &gt;=</td>
<td>CPUfreq scaling on V&lt;sub&gt;i&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;30%</td>
<td>V&lt;sub&gt;i&lt;/sub&gt; -&gt; V&lt;sub&gt;i-1&lt;/sub&gt;</td>
</tr>
</tbody>
</table>
Virtual Frequency (Issue3)

Scalable Virtual Processor OPP is mapped to 8 Physical CPU OPPs

\[ V_{Freq} = \sum_{i=1}^{4} (Freq(B_i) + Freq(L_i)) \]

- \( \text{Min}(V_{Freq}) = \text{Min}(Freq(L_1)) = 300\text{MHz} \)
- \( \text{Max}(V_{Freq}) = \text{Max} \left( \sum_{i=1}^{4} (Freq(B_i) + Freq(L_i)) \right) = 2\text{GHz} \times 4 + 1\text{GHz} \times 4 = 12\text{GHz} \)

Super wide performance dynamic range
300MHz to 12GHz with DVFS (Theoretical Value)
866MHz to 7.4GHz without Frequency scaling
Approach 2 Summary

Approach 2 solves all the three issues, **while All physical cores are active.**

- **Issue 1:** Optimum process placement is taken care of by changing the state of “Virtual Scalable Processor”.
- **Issue 2:** Additional input parameters such as temperature and Performance Index are exploited by C-Group Governor.
- **Issue 3:** CPUfreq Governor is consolidated with C-Group Governor and Established one dimensional Scaling scheme can be applied.

<table>
<thead>
<tr>
<th></th>
<th>Cluster Migration (Original)</th>
<th>In-Kernel Swither</th>
<th>big.LITTLE MP</th>
<th>Approach 1</th>
<th>Approach 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue 1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Issue 2</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Issue 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>All physical cores are active?</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Partially Yes</td>
<td>✓</td>
</tr>
<tr>
<td>Status</td>
<td>Not maintained</td>
<td>Used in a product</td>
<td>Work In Progress (Not in 3.10)</td>
<td>Available with the current kernel</td>
<td>Available with the current kernel</td>
</tr>
</tbody>
</table>
Evaluation
Evaluation on Renesas APE6 test board

- For evaluation a demo is integrated on AOSP Android 4.1.2+Linaro 12.10 kernel 3.6 on APE6 test board.
Evaluation Demo on APE6 test board

- Evaluation Demo consists of 3 components:
  - Input slider bars
  - CPU load windows
  - Application for Heavy CPU load
Evaluation Demo on APE6 test board

- 2 input slider bars
  - Performance Index and Temperature

Performance Index (0%-100%)

Full Performance (100%)

Power Conscious (0%)

Temperature
Evaluation Demo on APE6 test board

- CPU load window is prepared for each core
- 8 Sub windows for 8 cores
Evaluation Demo on APE6 test board

- 2D colliding n-body simulation for Heavy CPU load
  - Runs only on the cores which 2 slider bar inputs enable

The larger the number of CPU, the faster the speed of balls
Approach 1 Evaluation

- We evaluate, on demo with Approach 1 C-group governor,
  - Cluster is switched at Performance Index boundary (50%)
  - Number of core is determined by Performance Index
  - Falling back to 1 x CA7 over 60 deg C

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Performance Index</th>
<th>Dynamic Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>less than 60 deg C</td>
<td>0% - 20%</td>
<td>CA7 x 1</td>
</tr>
<tr>
<td></td>
<td>20% - 30%</td>
<td>CA7 x 2</td>
</tr>
<tr>
<td></td>
<td>30% - 40%</td>
<td>CA7 x 3</td>
</tr>
<tr>
<td></td>
<td>40% - 50%</td>
<td>CA7 x 4</td>
</tr>
<tr>
<td></td>
<td>50% - 60%</td>
<td>CA15 x 1</td>
</tr>
<tr>
<td></td>
<td>60% - 70%</td>
<td>CA15 x 2</td>
</tr>
<tr>
<td></td>
<td>70% - 80%</td>
<td>CA15 x 3</td>
</tr>
<tr>
<td></td>
<td>80% - 100%</td>
<td>CA15 x 4</td>
</tr>
<tr>
<td>lager than or equal to 60 deg C</td>
<td>0% - 100%</td>
<td>CA7 x 1</td>
</tr>
</tbody>
</table>
Approach 1 Evaluation Result

- We confirmed, on demo with Approach 1 C-group governor,
  - Cluster is switched at Performance Index boundary (50%)
  - Number of core is determined by Performance Index
  - Falling back to 1 x CA7 when over 60 deg C
We confirmed, on demo with Approach 1 C-group governor,
- Cluster is switched at Performance Index boundary (50%)
- Number of core is determined by Performance Index
- Falling back to 1 x CA7 when over 60 deg C
Approach 2 Evaluation

We evaluate, on demo with Approach 2 C-group governor,

- Scalable virtual processor is controlled in one dimension by Performance Index.
- The scalable virtual processor (Vi) evaluated in demo. (This is the second example Vi: i=1-8)

<table>
<thead>
<tr>
<th>Performance Index</th>
<th>Dynamic Process</th>
<th>Virtual Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0% - 10%</td>
<td>CA7 x 1</td>
<td>V1</td>
</tr>
<tr>
<td>10% - 20%</td>
<td>CA7 x 2</td>
<td>V2</td>
</tr>
<tr>
<td>20% - 30%</td>
<td>CA7 x 3</td>
<td>V3</td>
</tr>
<tr>
<td>30% - 40%</td>
<td>CA7 x 4</td>
<td>V4</td>
</tr>
<tr>
<td>40% - 55%</td>
<td>CA15 x 1 + CA7 x 4</td>
<td>V5</td>
</tr>
<tr>
<td>55% - 70%</td>
<td>CA15 x 2 + CA7 x 4</td>
<td>V6</td>
</tr>
<tr>
<td>70% - 85%</td>
<td>CA15 x 3 + CA7 x 4</td>
<td>V7</td>
</tr>
<tr>
<td>85% - 100%</td>
<td>CA15 x 4 + CA7 x 4</td>
<td>V8</td>
</tr>
</tbody>
</table>
Approach 2 Evaluation Result

- We confirmed, on demo with Approach 2 C-group governor,
  - V1(CA7 x 1) to V8(CA15 x 4 + CA7 x 4) scaling is controlled in one dimension by Performance Index slider bar.
  - At the highest end, this approach enables the use of all physical cores (CA15 x 4 + CA7 x 4) at the same time.

Performance Index = 100%

20% < Performance Index < 30%
Not evaluated yet

- CPUfreq consolidation
- Overhead measurement
- CPU hotplug and CPUIdle integration
Conclusion
Conclusion

- Two C-group based big.LITTLE solutions are proposed.
- Both Approaches solves all three challenging issues in big.LITTLE MP and can go with the current latest upstream kernel (3.9)

  - **Issue 1: Optimal process placement**
    Optimal process placement is taken care of by "Dynamic Process Group" assigned on “CPU cluster” or “Virtual Scalable Processor”

  - **Issue 2: Exploitation of additional input parameters**
    Additional input parameters such as chip temperature and Performance Index are exploited in C-Group governor.

  - **Issue 3: Consolidation with existing Power management framework**
    Established one dimensional Dynamic Voltage and Frequency Scaling scheme can be applied as is.
Conclusion

- In addition, Approach 2
  - Enables the use of all physical cores at the same time
  - Provides super wide performance dynamic range
    - 300MHz to 12GHz with DVFS (Theoretical Value)
    - 866MHz to 7.4GHz without Frequency scaling (This demo)
Next Step

- Further evaluation on Approach 2
  - CPUfreq consolidation
  - C-group governor performance overhead measurement
  - CPU hotplug and CPUidle integration

- Study on complementary solution with big.LITTLE MP kernel

- Investigation on other C-group based solutions
Thanks!
Trademarks

All trademarks and registered trademarks are the property of their respective owners.

big.LITTLE and its based trademarks are trademarks of ARM Holdings.
Android and its based trademarks are trademarks of Google Inc.
Linaro® is a registered trademark of Linaro in the U.S. and other countries
Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries