

# 組込みLinuxの起動高速化

株式会社富士通コンピュータテクノロジーズ

亀山英司

## ■ 組込みLinuxの起動時間短縮について依頼あり。

### ■ スペック

- ・ CPU : Cortex-A9 ( 800MB - single)
- ・ RAM : 500MB程度

## ■ 要件

### ■ 起動時間

- ・ 画出し . . . . . 5秒
- ・ 音出し . . . . . 3秒

### ■ 終了時間

- ・ 数msで電源断

## ■ 問題点

- 起動まで13秒 (“ login:” 表示まで)
- 機能の折込みは段階的に実施されるため、その度に再度チューニングが必要

## ■ 方針

boot loaderやkernel、アプリなどを個々に高速化するのではなく、段階的に提供される各モジュールの処理時間に影響されない高速起動の仕組みを作る。



**ユビキタス社の起動高速ソリューション  
「QuickBoot」を採用**



Ubiquitous

## Quick Boot

### ■ QuickBootとは

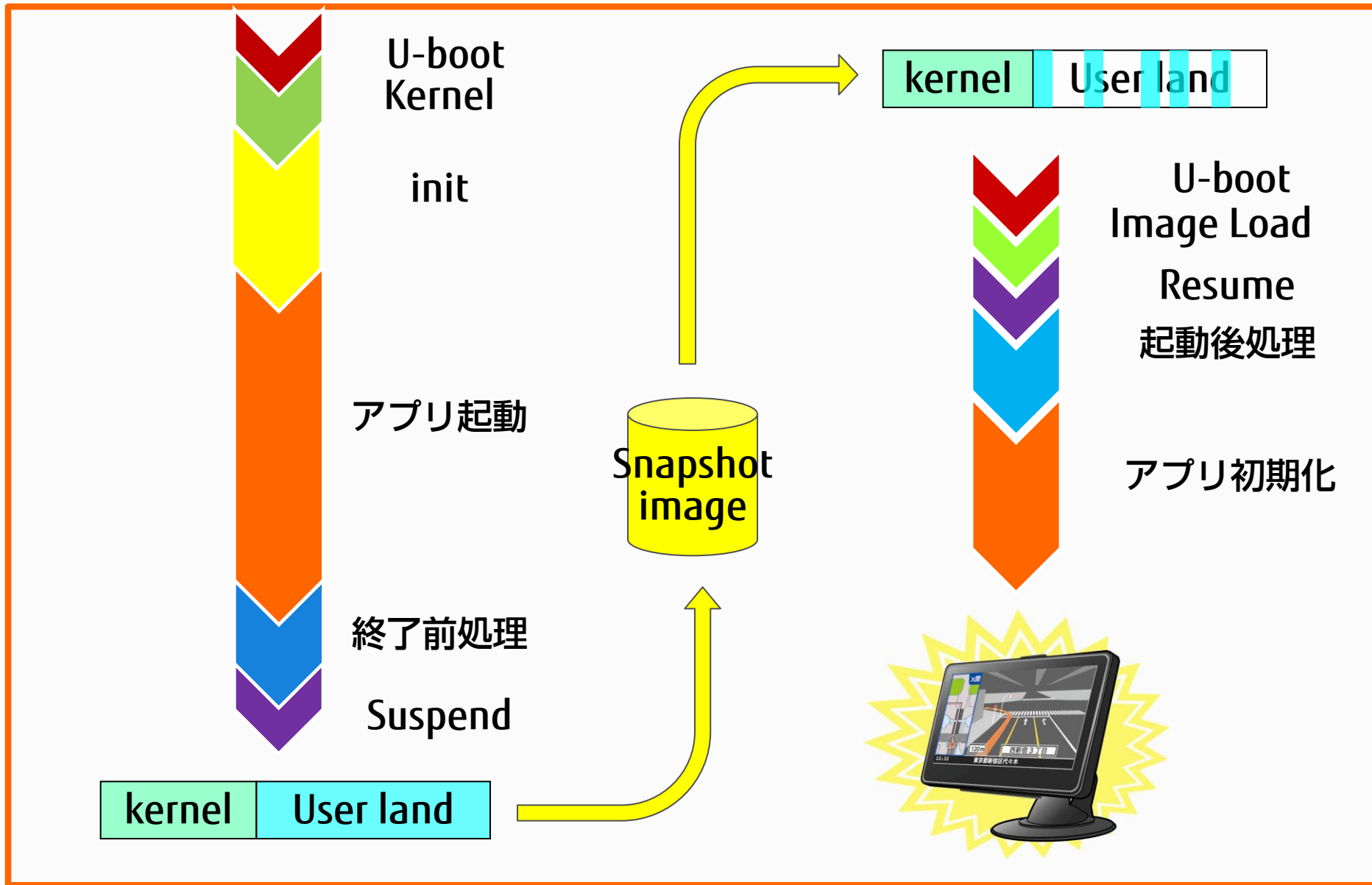
ハイバネーション技術をベースに株式会社ユビキタスが提供するAndroid/Linux向けの高速起動ソリューション。

高速起動時はスナップショット採取時の状態に復帰するため、各モジュールの初期化処理に依存されない。

システムの起動に必要なメモリ領域を優先的に不揮発性メモリからRAMに復元することで、他の方式と比べて圧倒的な速度の起動を実現。

アプリケーション側で使用しているメモリ量に依存せず常に高速起動が可能であり、残りのメモリ領域は起動後に順次読み込みを行うため、ユーザーの操作にほとんど影響を与えない。

# QuickBoot (2)

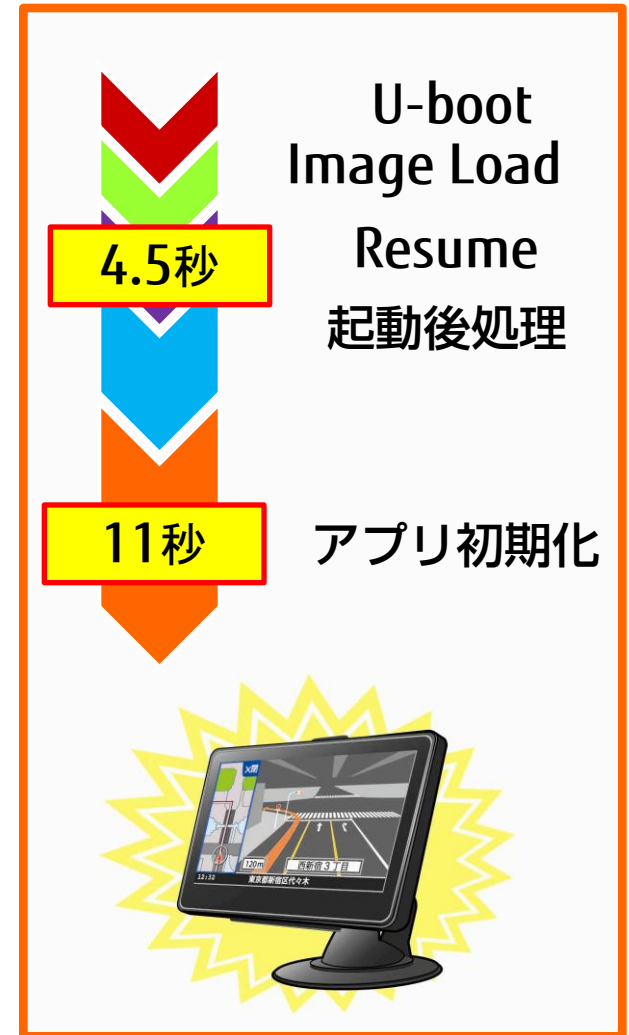


## ■ 効果

- アプリ起動までは13秒から4.5秒に短縮。
- しかしアプリ起動から画面出しまで11秒。



**更なる改善が必要**



- Snapshot Image
- Device Driver
- Task Priority
- File I/O

# 施策 1 Snapshot Image (1)

「各モジュールの処理に影響されない高速起動の仕組み」を実現するためには出来るだけ起動した状態に近い時点のスナップショットを採取することが望ましい。

## ■ 改善のポイント

- 採取のタイミング
- サイズの削減





# 施策 1 Snapshot Image (2)

## ■ スナップショット採取のタイミング

### ■ 課題

- ・スナップショットブート後に動作するアプリの初期化処理が多い。

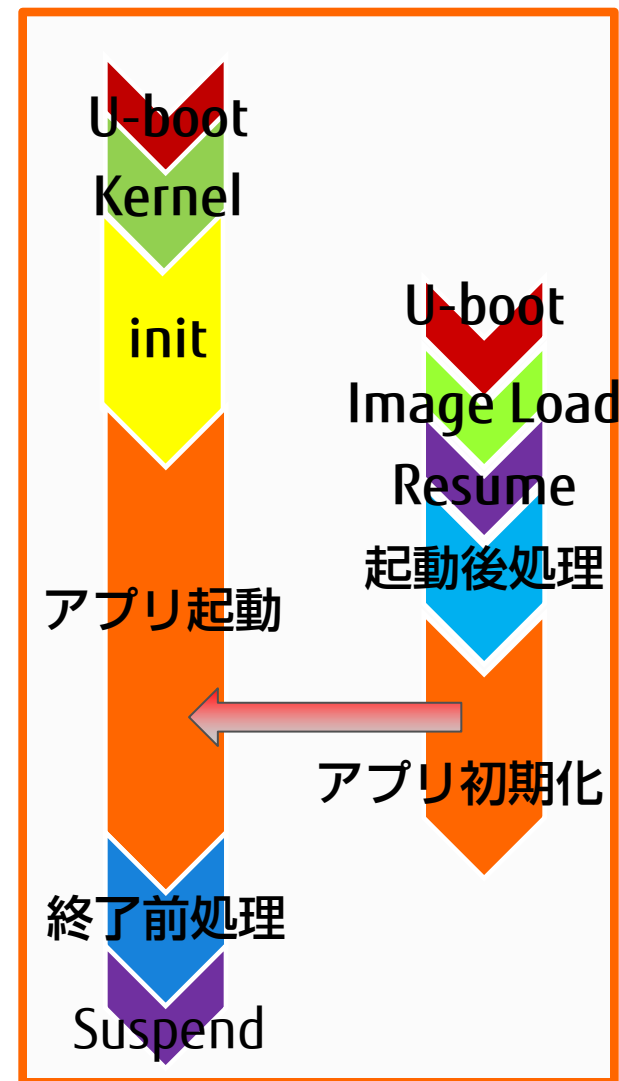
### ■ 施策

- ・アプリ側の初期化処理を見直し、スナップショット採取前に移動できるものを精査。



**スナップショットサイズが増大**

**100BM程度**



## ■ イメージサイズの削減

### ■ 課題

- ・領域不足。
- ・整合性確認時間の増大。
- ・ロード時間の増大。

### ■ 施策

- ・イメージ圧縮。

圧縮方式	圧縮率	伸長時間
LZF	約60%	○
ZLIB	約40%	△
<b>LZ4</b>	<b>約50%</b>	<b>◎</b>

- ・ページキャッシュの解放。
  - ・スナップショット起動後の処理が遅くなる場合あり。

## ■ Suspend/Resume対応可能なドライバ

### ■ Suspend/Resume対応

- ・ Suspend/Resume対応を行うことにより起動後処理にてinsmodするより高速。

### ■ 並列化

Suspend/Resume処理を並列動作させることにより短縮化。

- ・ SDホストインタフェースなどResumeに時間がかかるドライバを非同期で実施。
  - ・ `device_enable_async_suspend()`



## ■ Suspend/Resume非対応ドライバ

ロードダブルモジュール化して起動後処理にてロード。

### ■ Device node

- ・デバイスIDを固定とし、Suspend前に作成

### ■ Suspend時にロードだけ実施

- ・ロード時はなにもしない。
- ・Resume後従来の初期化処理を実施。



# 施策3 Task Priority (1)

## ■ 課題

- 高優先度で動作するタスクが多く、起動に必要な処理を阻害。

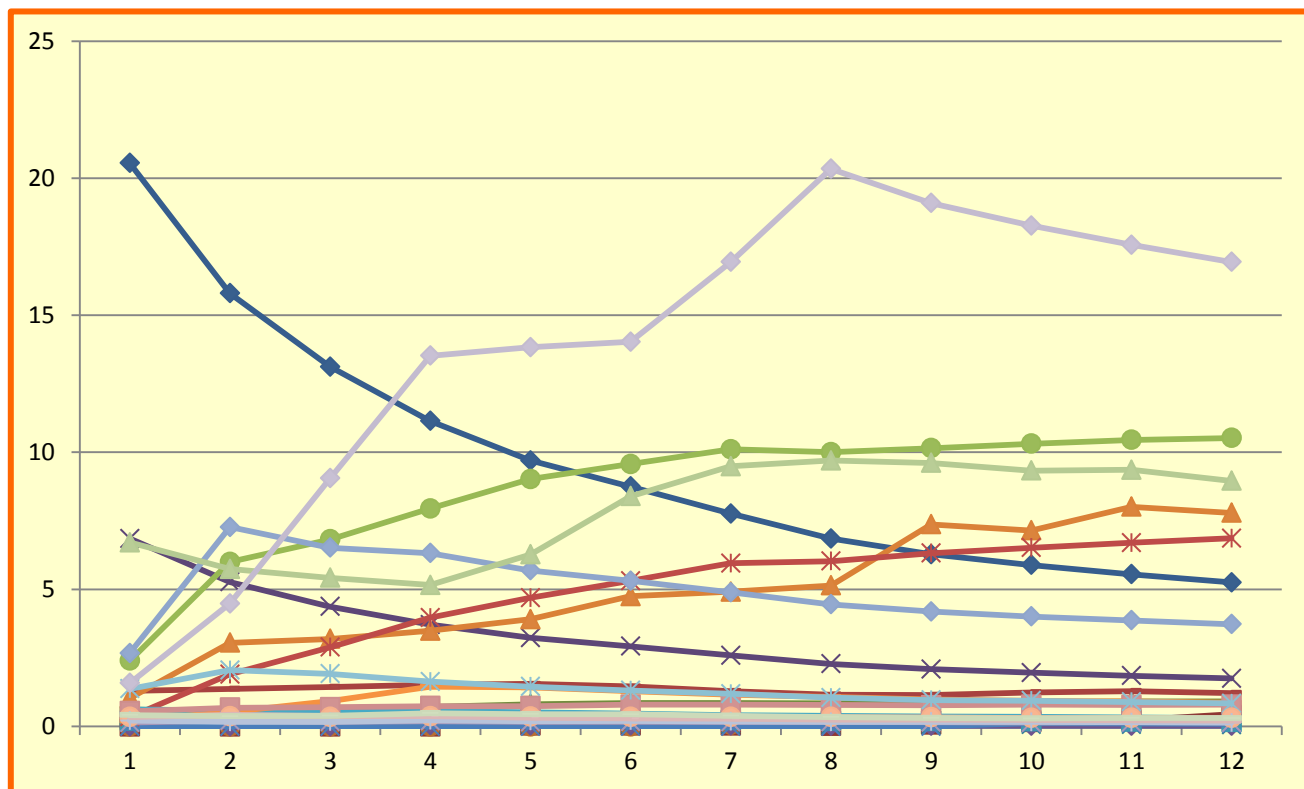


# 施策3 Task Priority (2)

## ■ 解析方法

### ■ Profile

- ・ JTAGやPerftools、FtraceなどのProfile機能を使用。
- ・ 単位時間当たりのプロセス・スレッドのCPU使用率をグラフ化。



## ■ タスク優先度の調整

### ■ 施策

- ・各タスクのプライオリティを調査。起動に必要なタスクを優先的に動作させる様に優先度を調整。
- ・Kernelのタスクスケジューラの設定を調整し、起動時間が最も短くなる設定を検証。

## ■ ファイルI/O負荷の集中

### ■ 課題

- ・ 高速起動後各プロセスが一斉に動作する為、プログラムロードによるファイルI/Oが集中。
- ・ プロセスの初期化時に画面データやDB参照などファイルの読み込みが多発。



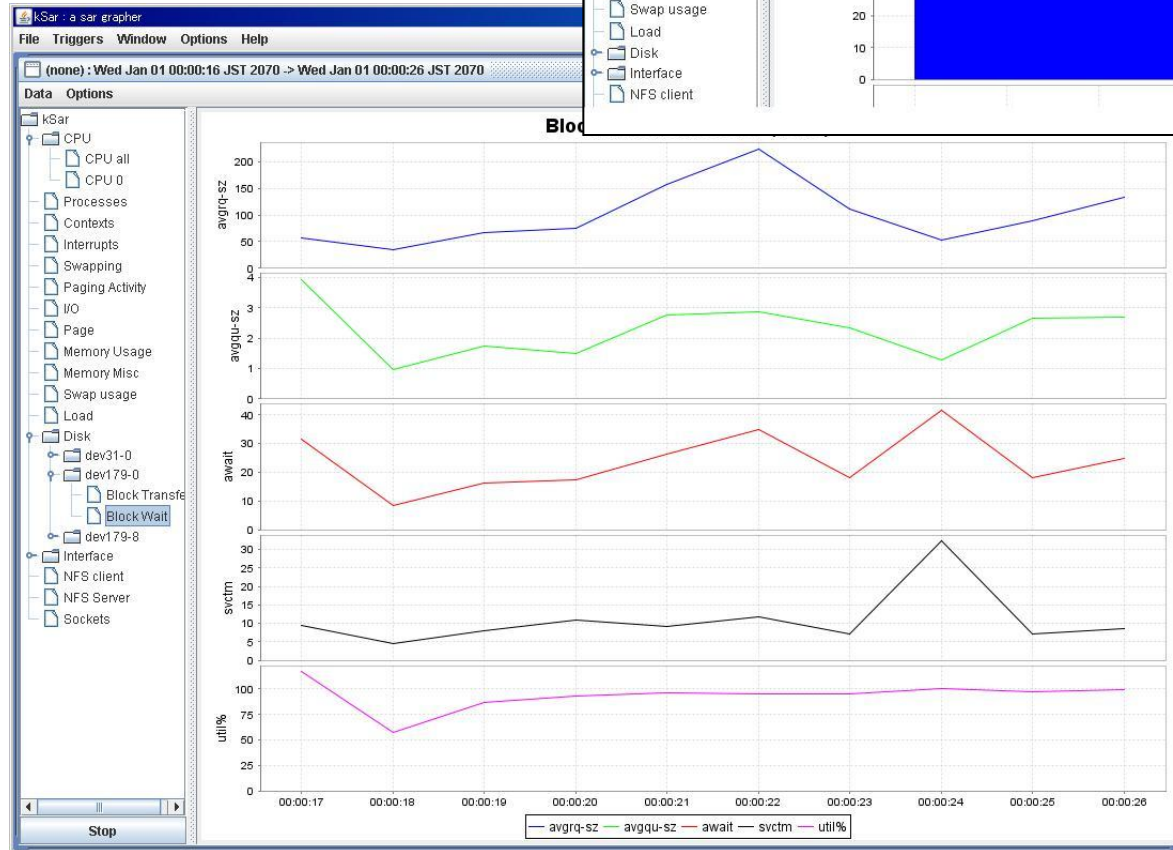


# 施策4 File I/O (2)

## ■ 解析方法

### ■ Sarを使用

- ・ I/O Waitがある。
- ・ Utilが100%を示す。



## ■ ファイルI/O負荷の集中

### ■ 対策

- ・ 起動に最低限必要なプロセスのみ先に動作させ、Page faultによるプログラムロードを低減させる。
- ・ プロセスがアクセスするファイルの読み出しタイミングを、プロセスの初期化処理で一括で行うのではなく、そのデータを最初に使用するタイミングに変更。

## ■ ファイルI/O性能

### ■ 課題

- ・ファイルI/O自体が遅い。

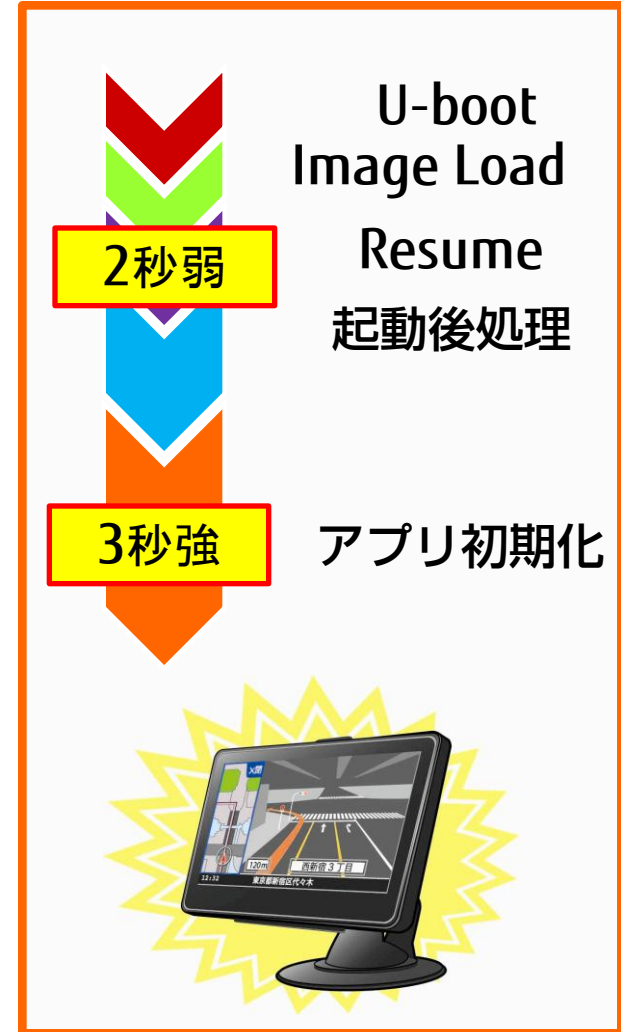
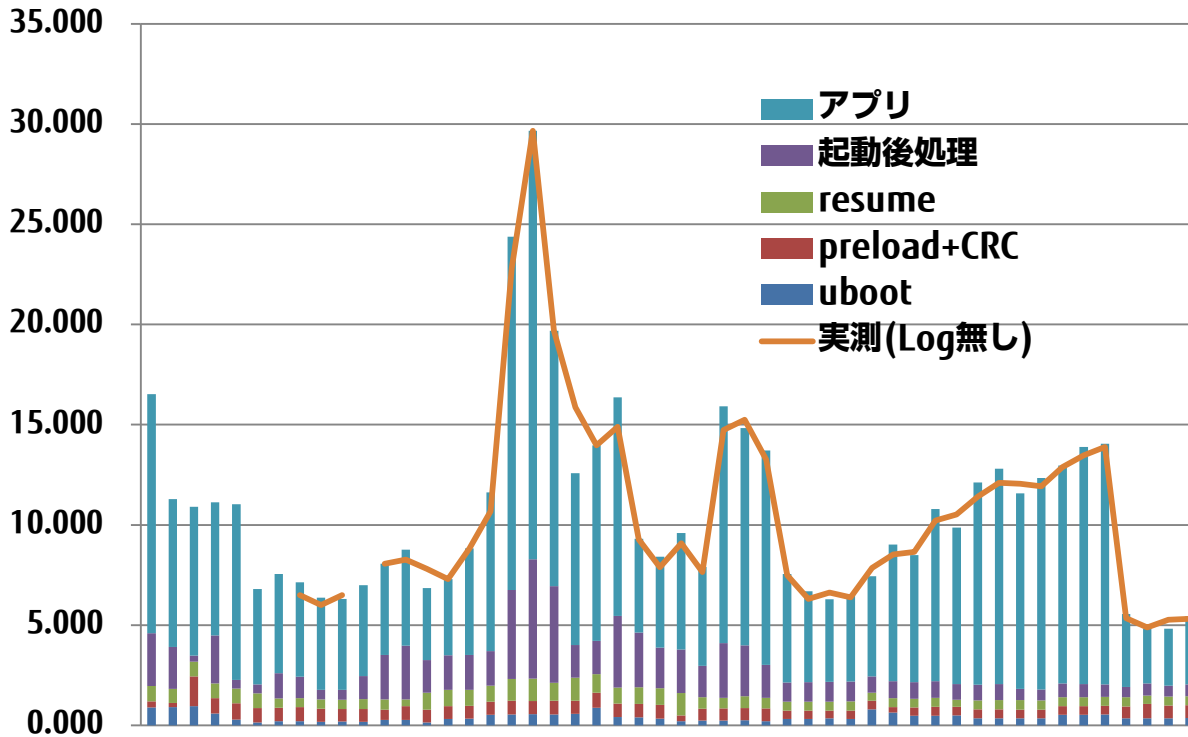
### ■ 施策


- ・ファイルを事前にキャッシュ。
  - ・スナップショット採取前に不揮発なメモリへコピー。
- ・I/Oスケジューラの選定
  - ・Kernelで選択可能な複数のI/Oスケジューラ(CFQ/deadline/NOOP)の検証とチューニング。
- ・I/Oプライオリティの見直し
  - ・タスクごとにI/Oプライオリティをチューニング (CFQ使用時)。

## ■ 効果

### ■ 起動時間

- ・ 画出し . . . . . 5秒強
- ・ 音だし . . . . . 4秒強





**FUJITSU**

shaping tomorrow with you