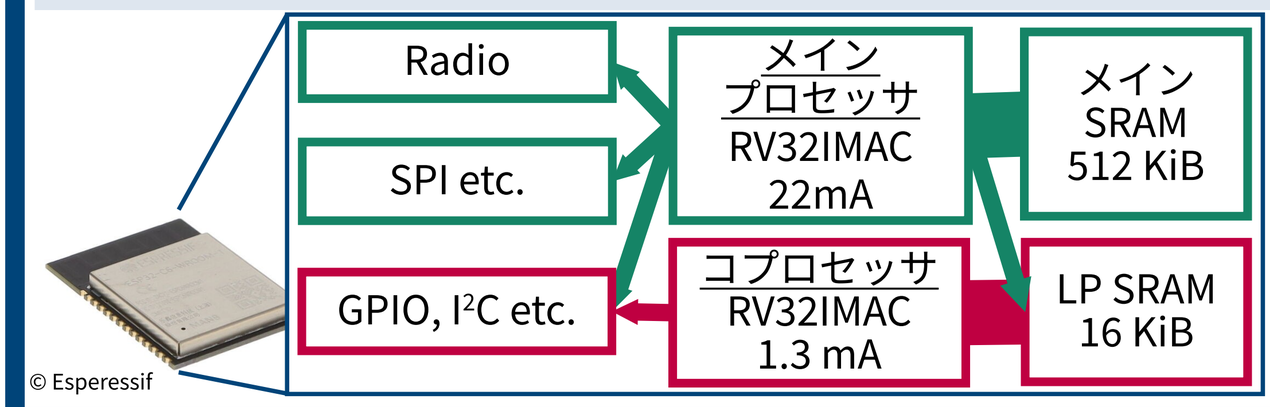
SWEST 27 インタラクティブセッション

8KiB程度の省電力コプロセッサで動く mruby/copro

鈴木豪，渡部卓雄，森口草介（東京科学大学情報理工学院）

1.組込みデバイスにおける ヘテロジニアスメモリアーキテクチャ

メインプロセッサの他に省電力コプロセッサを持つ **ESP32-C6**がターゲット



2.コプロセッサプログラミングの問題

1 コプロセッサとメインプロセッサでプログラムが分断される

- ある関数を両プロセッサで使うには2箇所に書く
- main関数から始まり，制御フローが分かりにくい
- ISA（命令ｾｯﾄｱｰｷﾃｸﾁｬ）が異なる場合がある

```
sensor_t sensor;
int sensor_read(sensor_t & sensor) { … }
int main(void) {
  sensor = sensor_new();
  send_to_copro(&sensor);
  val = sensor_read(&sensor);
  run_copro();
}
```

メインプロセッサ用

```
int sensor_read(sensor_t & sensor) { … }
int main(void) {
  val = sensor_read(&sensor);
  send_to_main(&val);
}
```

コプロセッサ用

2 プログラマが手動でデータを転送する

- コプロセッサからメインメモリにアクセスできない
- 利用可能なメモリ空間が非常に小さい（16 KiB）
- MPU(メモリ保護機構)が無い場合がある
- メモリマップが異なる場合がある

3. 提案パラダイム

コプロセッサで30回計測するIoTセンサの例

```
sensor = SHT30.new(I2C.new())
result = Array.new(30)
Copro.run do
 i = 0
 while i < 30 do
  v = sensor.read()
  break if v.nil?
  result[i] = v; i += 1
  Copro.delayMs(60*1000)
end; end
Network.send(result)
```

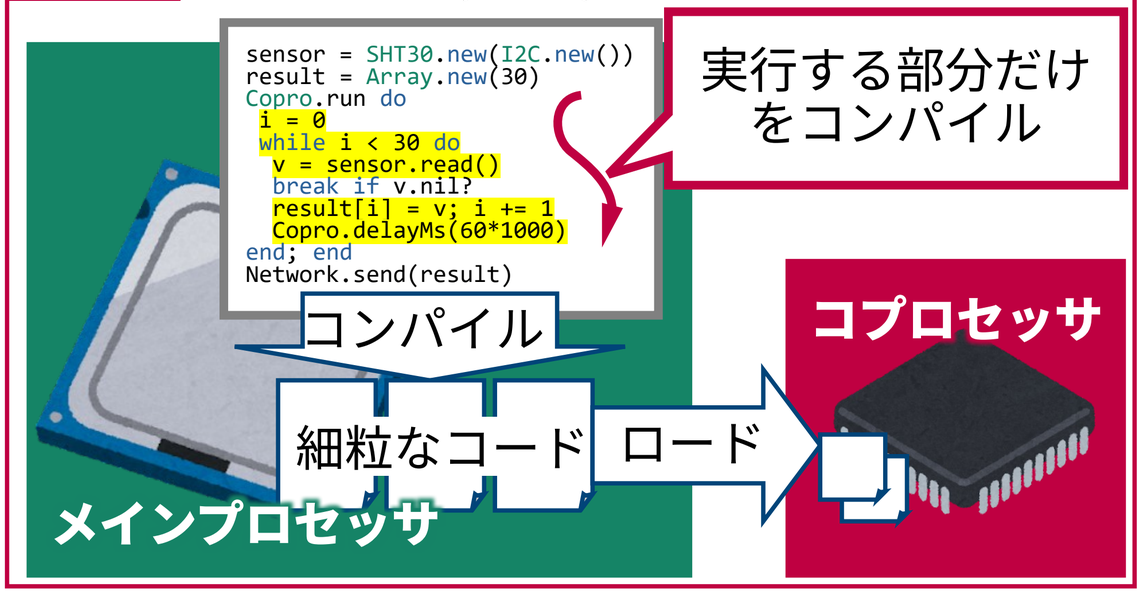
1つの定義で両プロセッサで使える

Copro.runによってコプロセッサで実行

自動転送

4. 提案手法

1 実行する部分だけをコンパイルする動的(JIT)コンパイル手法



2 コンパイラと協調するソフトウェア分散メモリ

- 不変条件を決めることで，チェックを削減
- 似た研究HeraJVM[R. M. OOPSLA'10, HotOS'09]と異なり，FIFOキャッシュを採用
- 変換結果のライトバック

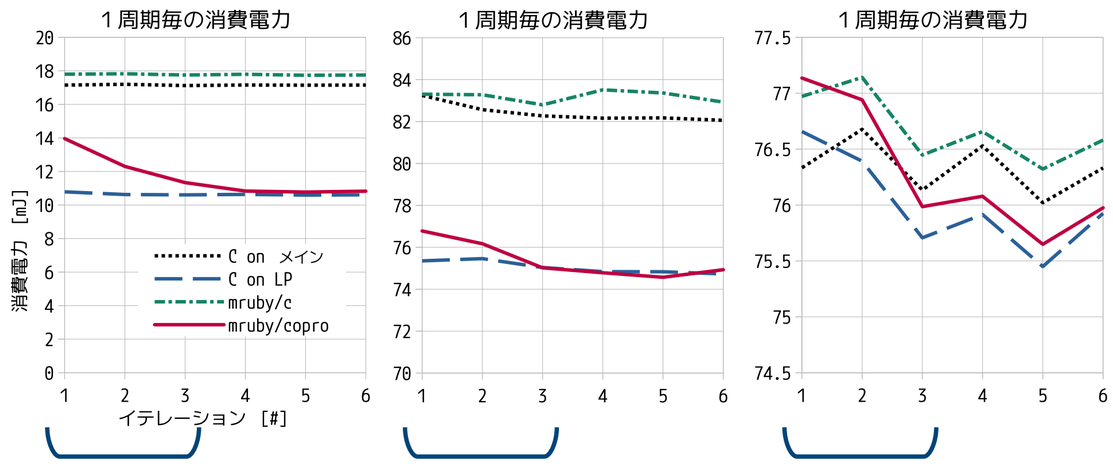
5.実験

加速度読取
12Hz・12.5%*
平均処理

水位計
1/3 Hz
ソート・平均

温度湿度計
1 sample/min
処理なし

* デューティー比



最初の3回は消費電力オーバヘッドが大きい
理由: JITコンパイル・オブジェクトの転送

コードサイズ (C言語と比較して)

6.6x　5.2x　4.8x

処理がある方がオーバヘッドが大きい

MPLR2025で発表予定
Co-operative JIT Compilation for Resource-Constrained Low-Power Coprocessors

今後の予定

- mruby/copro
  - クロージャの実装
  - GraalVM/Truffle[A. W. PPPJ'14]に似たオブジェクトの型推論による型チェック削減
- 静的型付け言語（Rust? C++? Java?）