特別研究報告

題目

実装方法の違いによるプログラムの消費電力量および実行時間への影響に関する調査

指導教員

楠本 真二 教授

報告者

松尾 裕幸

平成 29 年 2 月 14 日

大阪大学 基礎工学部 情報科学科

平成 28 年度 特別研究報告

実装方法の違いによるプログラムの消費電力量および実行時間への影響に関する調査

松尾 裕幸

内容梗概

近年，モバイル端末やデータセンターの数は増加の一途をたどっており，それに伴いプログラムやソフトウェアの消費電力量を削減する必要性がますます高まってきている．消費電力量の削減へのアプローチとしてはソフトウェア面やハードウェア面が挙げらるが，本研究では，ソフトウェア面から議論する．

ソフトウェア面からの消費電力量削減の研究は盛んに行われている．しかしながら，我々は，プログラムの総消費電力量と実行時間との間には，強い相関があるのではないかという仮説をたてた．この仮説が正しければ，消費電力量の削減という問題は，実行時間をいかに短縮させるかという問題に帰着させることができる．

本研究では，独自の電力消費計測基盤 eTracker を開発し，主要なソーティングアルゴリズム，および主要な Java Collections クラスについて，それぞれの総消費電力量および実行時間を計測し，両者の相関を調べた．また，消費電力量に与える影響についてのほかの要因として，平均メモリ使用量を取りあげ，これと総消費電力量との間の相関も同様に調べた．

実験の結果，総消費電力量と実行時間との間には強い正の相関がみられた．また，総消費電力量と平均メモリ使用量との間にも正の相関がみられたが，総消費電力量と実行時間の相関ほど強くはなかった．この結果から，開発者がプログラムの総消費電力量を削減したい場合，第一に実行時間，第二に平均メモリ使用量がそれぞれ少ない実装手段を選択すればよいことが示された．

主な用語

消費電力，ソフトウェア，プログラム，ソーティングアルゴリズム，Java Collections クラス

目次

図目次

表目次

1　まえがき

近年，社会の情報化が進み，モバイル端末やデータセンターは増加・複雑化・大規模化の一途をたどっている．モバイル端末は一般的にバッテリー駆動であり，端末上で動作するプログラムやアプリケーションの消費電力量が，端末の稼働時間を大きく左右している．ソフトウェアの消費電力量を削減し，端末の稼働時間を長くすることは，端末を使用する我々ユーザにとって大きな問題である．また，ある調査によれば，平均的なデータセンターは 25,000 世帯分の電力を消費するとされ [16]，全世界で発電される総電力のうち 1.5 〜 3 % がデータセンターで消費されているという [8]．24 時間稼働しなければならないその性質上，データセンターは莫大な電力を消費しており，それに伴う電気料金や地球資源の利用が大きな問題となっている．このように，プログラムやアプリケーションの消費電力の効率を考える必要性がますます高くなっているといえる．

消費電力への問題に対するアプローチとしては，ハードウェアの省電力化や，搭載ソフトウェアの省電力化，またモバイル端末においてはバッテリーの大容量化および小型化の両立など，いくつかの方法が考えられる．このうち，搭載ソフトウェアの省電力化については，ハードウェアの再設計や交換の必要がなく，既存のユーザについてもアップデートプログラムを配布することで適用でき，比較的実施・普及させやすい手法といえる．本研究では，この搭載ソフトウェアの省電力化について着目する．

搭載ソフトウェアの省電力化についての研究は盛んに行われている [9, 10, 14]．Bunse らは，ソーティングアルゴリズム間の総消費電力量を比較し，メモリ効率のよいインサーションソートがもっとも総消費電力量が少ないことを示した [9, 10]．Hasan らは，主要な Java の Collections クラスにおける総消費電力量を比較・分析し，また実際にライブラリやアプリケーション中の実装を変更することで，消費電力量に最大 300 % の差がうまれたと述べている [14]．これらの研究成果は，ソフトウェアの観点から電力の消費について考えることが妥当であることを示している．しかし，Bunse らと Hasan らの主張には食い違いが見られた．Bunse らが実行時間と総消費電力量との間には相関がないと主張しているのに対して，Hasan らは両者の間に相関がある傾向が見られたと述べているのである．なお，Hasan らは相関の有無を結論づけるにはさらなる調査が必要だとし，定量的には述べていない．我々は，この食い違いの理由が，Bunse らと Hasan らの実験環境の違いにあるのではないかと考えた．Bunse らは CPU のみの消費電力量を計測しているのに対して，Hasan らは計算機全体の消費電力量を計測しているためである．

以上の考察から，我々は，計算機全体の消費電力量について計測した場合，実行時間と総消費電力量の間には相関があるのではないかという仮説をたてた．この根拠としては，計算機において，電源を入れている間，CPU やメモリ，その他の計算機リソースが常に電力を消費し続けることが挙げられる．また，実際に消費電力量を計測することは一般的に困難であり，専用の計測機器が必要となる．前述した

Bunse らや Hasan らも，それぞれ独自の消費電力量計測システムや基盤を開発し，研究を行っている．このように，総消費電力量について考える場合，問題の本質に迫る前に，実際の総消費電力量をいかに計測するかという問題が発生する．このことは，特にスタートアップのような小さな企業や，個人で開発している開発者など，開発環境に乏しいユーザにとって大きな問題である．前述した仮説が正しい場合，プログラムの消費電力量の削減という問題は，これまでにも数多く取り組まれている実行速度の最適化 [11, 17, 20] という問題に帰着できる．すなわち，総消費電力量を削減したい開発者は，たんに実行時間が短いような実装手段を選べばよいといえる．これは多くの開発者にとって有用な事実となる．

本研究では，この仮説をもとに，同じ出力だが実装手段が異なるような様々なアルゴリズム・実装について，それぞれの総消費電力量と実行時間の相関を調べた．具体的には，主要なソーティングアルゴリズムや，主要な Java Collections クラスの実装クラスについて，それぞれを実装・利用した簡単なプログラムを用意した．また，独自の電力消費計測基盤 eTracker を用いて，各プログラムの総消費電力量を計測した．さらに，実行時間以外にも総消費電力量と相関があるのかを調べるため，平均メモリ使用量を取りあげ，同様に総消費電力量との相関を調べた．

実験の結果，実行時間と総消費電力量の相関係数 r は 0.9 を超えており，前述した仮説は正しいことが示された．また，平均メモリ使用量と総消費電力量との間にも相関がみられた．しかし，実行時間と総消費電力量のそれと比べると，平均メモリ使用量と総消費電力量の相関は弱いことがわかった．この結果から，開発者が総消費電力量を削減したい場合，まず実行時間がもっとも短い実装手段を採用し，次に平均メモリ使用量が少ない実装手段を選択すればよいことが示された．

以降，2 章では，本論文において用いる知識や用語，および関連研究について述べる．3 章では，本研究における調査の目的を示し，また実験にあたってたてた仮説について述べる．4 章で実験環境について説明し，5 章で実験題材および実験手法を述べ，実験結果を示す．6 章では，実験結果をもとに考察を行い，仮説の真偽を述べる．7 章では，妥当性の脅威について述べる．最後に，8 章で，本研究のまとめおよび今後の課題について述べる．

2　準備

本章では，本論文で扱う知識や用語，および関連研究について述べる．

2.1　電気に関する諸用語

2.1.1　瞬時電力

直流回路について考える．電気回路を流れる電流を I，電圧を E，電力を P とすると，P は

$$P = EI$$

と計算できる．P の単位としては [W] が用いられる．

I や E の値が変化しない定常状態においては，時間に関わらず P の値は一定となる．しかし，一般にコンピュータシステムでは，回路中の I, E の値はその瞬間にかかる負荷によって目まぐるしく変化する．このような環境においては，I, E をそれぞれ時間によって変化する関数として考える必要があり，したがって P も関数となる．すなわち，時間 t における瞬時電流を $I(t)$，瞬時電圧を $E(t)$ としたとき，その瞬時電力 $P(t)$ は

$$P(t) = E(t) \cdot I(t) \tag{1}$$

となる．

2.1.2　電力量

電力量を W とすると，W は瞬時電力 $P(t)$ をすべての時間 t について積算することで求めることができる．すなわち，

$$W = \int P(t)\, dt \tag{2}$$

である．W の単位としては [J] や [W·s] が用いられる．

一般に，コンピュータシステムにおいて「あるプログラムを動かしたときの消費電力量」とは，「あるプログラムが起動してから終了するまでの電力量」つまり

$$\int_{t_{\mathrm{begin}}}^{t_{\mathrm{end}}} P(t)\, dt \tag{3}$$

のことを指す．ここで，t_{begin} はプログラムを起動した時刻，t_{end} は終了した時刻を表す．

以降，本論文では，この時刻 t_{begin} から時刻 t_{end} までの間に消費した電力量のことを「総消費電力量」と定義する．また，$t_{\mathrm{end}} - t_{\mathrm{begin}}$ を「実行時間」，総消費電力量を実行時間でわった値を「単位時間あたりの消費電力量」とそれぞれ定義する．

2.2 ソーティングアルゴリズム

表 1 に，主要な 7 つのソーティングアルゴリズムとその概要を示す [7, 18]．ここでは，ソート対象となる入力データのサイズ（数）を n とした．なお，シェルソートは，適当な間隔 h をあけてデータ列を取り出し，各データ列に対してインサーションソートを実行するアルゴリズムである．この h によって，平均の時間計算量は変化する．また，シェーカーソートはバブルソートを変形したものである．最悪の計算量は $O(n^2)$ であるが，入力データが整列されていればいるほど，計算量は線形（$O(n)$）に近づく．

2.3 関連研究

2.3.1 異なるソーティングアルゴリズム間での総消費電力量の比較

Bunse らは，主要なソーティングアルゴリズム間の総消費電力量を計測・比較している [9, 10]．対象としているアルゴリズムは，表 1 で挙げた 8 つである．各ソーティングアルゴリズムを実装したプログラムは，Atmel 社製のマイクロコントローラ（いわゆるマイコン）上で実行している．Bunse らは，この CPU のみの総消費電力量を直接計測している．

実験の結果をもとに，Bunse らは，

- プログラムの実行時間と総消費電力量との間には直接的な相関はなく，メモリ消費量が総消費電力量に決定的な影響を与えていること
- 入力データの格納以外に追加でメモリを消費することのない（in-place な）アルゴリズムである，インサーションソートの電力効率がよいこと

表 1: 主要なソーティングアルゴリズムの概要

アルゴリズム	平均時間計算量	安定性
ヒープソート	$O(n \log n)$	—
マージソート	$O(n \log n)$	✓
クイックソート	$O(n \log n)$	—
バブルソート	$O(n^2)$	✓
インサーションソート	$O(n^2)$	✓
セレクションソート	$O(n^2)$	—
シェーカーソート	$O(n^2)$	✓
シェルソート	実装に依存	—

などを指摘するとともに，各ソーティングアルゴリズムの総消費電力量の傾向を図表にまとめている．

2.3.2 Java における異なる Collections クラス間での総消費電力量の比較

Hasan らは，Java の Collections クラスにおける，`List`, `Map` および `Set` インターフェースのそれぞれの主要な実装クラスについて，各クラスのインスタンスに共通の操作を行う際の総消費電力量を計測し，詳細に分析している [14]．`List` の実装としては，Java Collections Framework [3]（JCF）に含まれる `ArrayList` および `LinkedList`，Trove [6] に含まれる `TIntArrayList` および `TIntLinkedList`，そして Apache Commons Collections [1]（ACC）に含まれる `TreeList` を選んでいる．同様に，`Map` および `Set` についても，それぞれ JCF, Trove, ACC から主要な実装を選んでいる．さらに，インスタンスへの操作としては，リストの冒頭・中間・末尾への挿入や，繰り返し処理，ランダムアクセスを行なっている．実験には，独自に作成した電力計測基盤 GreenMiner [15] を用いている．この基盤では，Java プログラムを Android 端末（Samsung Galaxy Nexus）上で動作させ，その端末全体の総消費電力量を計測している．

Hasan らは実験の結果から，とくにリストのサイズが 500 以上のとき，実装クラス間の総消費電力量の差が顕著になったと述べている．表 2 に示すのは，Hasan らが述べている実験結果の一部である，`List` の各実装クラスについての結果である．また，Hasan らは，実際のライブラリやアプリケーションにおける，Collections クラスの実装の変更が総消費電力量に与える影響についても調査している．実験によれば，適切でないクラスを選択した場合，最も効率のよいクラスに比べて，総消費電力量が 300 % 大きくなったと述べている．

表 2: Hasan らの実験結果の一部より，Java の主要な各実装クラスによる総消費電力量の違い

操作	総消費電力量が大きいクラス	総消費電力量が小さいクラス
リストの冒頭への挿入	`TreeList`	`LinkedList`, `TIntLinkedList`
リストの中間への挿入	`TIntLinkedList`	`ArrayList`, `TIntArrayList`
リストの末尾への挿入	`TreeList`	`TreeList` 以外で大きな差はなし
繰り返し処理	リスト間で大きな差はなし	リスト間で大きな差はなし
ランダムアクセス	`TIntLinkedList`	`ArrayList`, `TIntArrayList`, `TreeList`

3　調査目的と仮説

本章では，本研究を行う動機および仮説について述べる．

3.1　関連研究における問題点

前述の通り，Bunse らは論文 [9] で次のように述べている：

> プログラムの実行時間と総消費電力量との間には，直接的な相関関係は存在しない．

一方で，Hasan らは，論文 [14] 中の議論で次のように述べている：

> プログラム中の Collections クラスの実装を変更し，実行時間が長くなったとき，総消費電力量も応じて大きくなるという傾向に気がついた．

Hasan らのこの主張は，次のように言いかえることができる：

> プログラムの実行時間と総消費電力量との間には，相関関係が存在する傾向にある．

我々は，Bunse らの主張と Hasan らの主張の間にこのような食い違いが生まれた原因を，両者の実験環境の違いにあると考えた．Bunse らの研究では，CPU のみにおける総消費電力量を計測しているが，Hasan らの研究では，Android 端末全体における総消費電力量を計測しているのである．一般に，計算機上でプログラムを実行する場合，CPU のみが電力を消費するという場面は考えにくい．なぜなら，ほとんどの場合，プログラムの動作に伴い，主記憶装置（メモリ）やそのほかの入出力装置などによる電力の消費が併せて発生するためである．

Hasan らは，前述した傾向が実験で多くみられたと述べている．しかし，Hasan らの研究の主眼は Java クラス間の総消費電力量の比較分析であるため，実行時間と総消費電力量との間に相関があると結論づけるにはさらなる調査が必要だとし，定量的には述べていない．

以上の考察より，本研究では，Bunse らや Hasan らが行った実験の一部について，次の 2 点に注意して再実験を行うこととした：

1. CPU のみではなく，計算機全体の総消費電力量について計測する．
2. プログラムの総消費電力量と同時に，その実行時間についても計測する．

以下，本論文では，とくに断りのない限り，プログラムを実行する計算機全体の総消費電力量について議論するものとする．

3.2 仮説

前節に基づき，以下に示す仮説 H_1 〜 H_3 をたてた．

仮説 H_1　総消費電力量と実行時間との間には正の相関がある．

計算機は一般に，電源がついている間，常に電力を消費し続けている．よって，プログラムの総消費電力量の大きさは，その実行時間に大きく依存しているはずであり，両者には強い正の相関があるはずだと考えた．

仮説 H_1 が正しい場合，プログラムの開発において総消費電力量を小さくしたいとき，開発者は実行時間が短くなるように実装すればよいことになる．一般に，実行時間に比べて総消費電力量の計測や見積もりは難しいため，このことは多くの開発者にとって有用な事実となる．

仮説 H_2　実装手段により，単位時間あたりの消費電力量は異なる．

アルゴリズムやその実装により，メモリなどの計算機リソースの扱いが変わる．こういった実装方法の違いで，単位時間あたりの消費電力量に差が現れるはずであると考えた．

仮説 H_2 が正しい場合，消費電力量に影響を与える要因は実行時間だけではなく，他の要因が存在するといえる．

仮説 H_3　総消費電力量と平均メモリ使用量との間には正の相関がある．

先の仮説 H_2 が正しいとすれば，実行時間以外の要因が何なのかという疑問もうまれる．この要因として，計算機リソースが消費する電力が考えられる．本研究では，計算機リソースとして，平均メモリ使用量を取りあげた．

仮説 H_3 が正しい場合，プログラムの総消費電力量を削減したいとき，より平均メモリ使用量が少ない実装を選べばよいといえる．仮説 H_1 の場合と同様に，平均メモリ使用量は総消費電力量よりも見積もりやすいため，開発者にとって有用な情報となる．

4　実験環境

前章で挙げた仮説を検証するため，実験を行う必要がある．本章では，この実験を実施する環境について，詳細に述べる．

4.1　総消費電力量の計測環境：eTracker

4.1.1　eTracker の設計

我々は，計算機の総消費電力量を計測するための独自基盤 eTracker[*1] を設計・開発した．設計にあたっては，同様の電力計測基盤である GreenMiner [15] を参考にした．図 1 に，eTracker の回路図を示す．また，図 2 は，eTracker をブレットボード上に実装したイメージである．

eTracker は，次の 3 つのコンポーネントに大きく分けれられる：

(a) Raspberry Pi 上でプログラムを実行する TARGET（図 1 の左側）
(b) TARGET が消費した電力を計測する INA219（図 1 の中央）
(c) INA219 が計測したデータを処理・保管する LOGGER（図 1 の右側）

INA219 [2] は Adafruit 社製の電力計測チップ INA219 であり，GreenMiner で用いられているものと同一である．また，GreenMiner では計測対象機器（TARGET にあたる）が Android 端末であったのに対し，eTracker では将来の拡張性などを見据え，TARGET を Raspberry Pi[*2] 3 Model B [5] とした．

LOGGER へはコンセントから microUSB ケーブルを用いて直接給電する（通常の給電方法）．一方で，TARGET が消費する電力を計測するため，TARGET へは計測部である INA219 を経由させて GPIO[*3] ピンから給電する．

INA219 で直接計測できるのは，計測した瞬間の時刻 t における瞬時電流 $I(t)$ および瞬時電圧 $E(t)$ である．LOGGER は，I^2C[*4] 通信でこれらを INA219 に問い合わせる．瞬時電流と瞬時電圧がわかれば，このときの瞬時電力 $P(t)$ は，式 (1) により計算できる．こうして得られた瞬時電力 $P(t)$ は，図 3 のような離散値となる（得られた離散値同士を線でつないである）．

このとき，たとえば時刻 t_1 に電力を取得し，時刻 t_2 に再び取得したとする．$t_2 - t_1$ が十分に小さけ

[*1] オープンソースソフトウェアとして公開している：`https://github.com/h-matsuo/eTracker`.

[*2] イギリスの Raspberry Pi 財団によって開発されている，小型のコンピュータ．図 4 の左右に見えるのが Raspberry Pi である．eTracker では，Raspberry Pi の OS として，Linux ディストリビューションの一つである Raspbian を採用している．

[*3] General Purpose I/O の略．マイクロコントローラなどの組み込みデバイスが外界と通信するための汎用的な入出力ピンとして広く用いられている．

[*4] Inter-IC の略．オランダの Philips 社が開発した，IC 間でシリアル通信を行うためのインターフェースの一つ．データ（SDA）およびクロック（SCL）から成る 2 本の通信線でデータをやり取りできる．

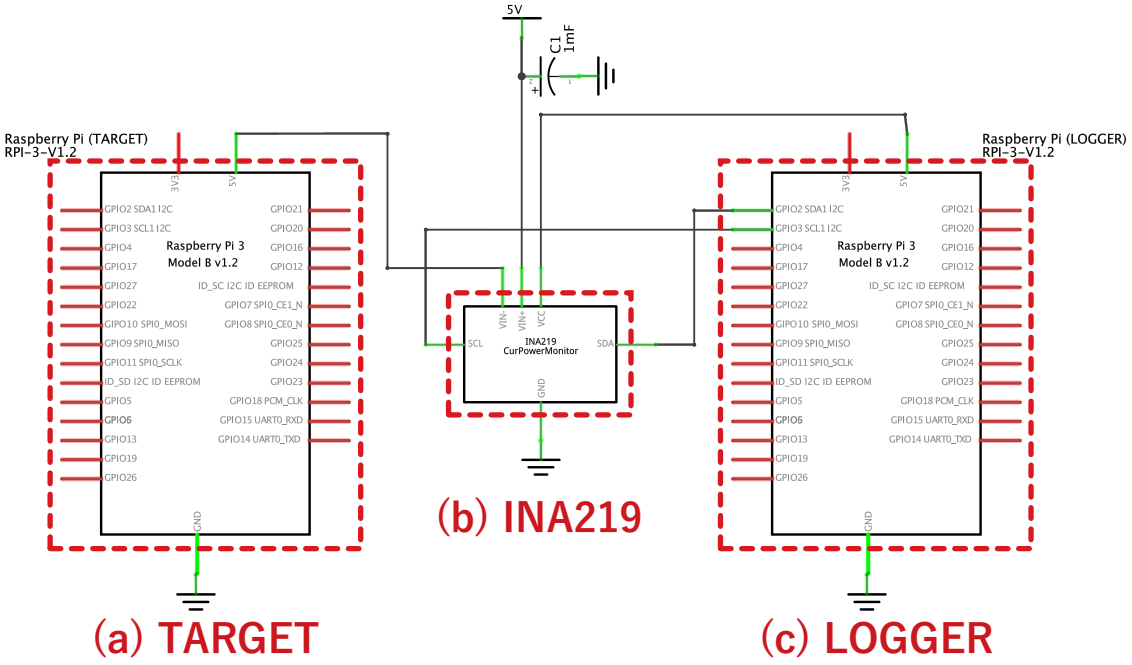


図 1: eTracker の回路図

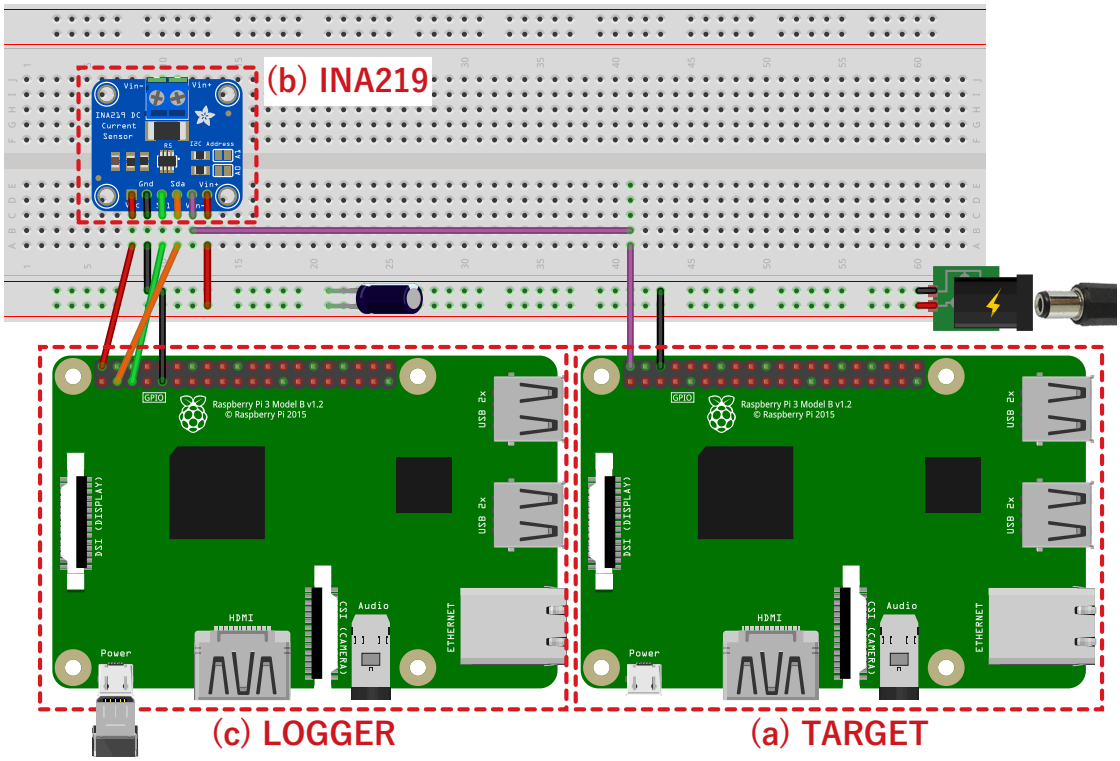


図 2: eTracker のブレッドボード上への実装イメージ

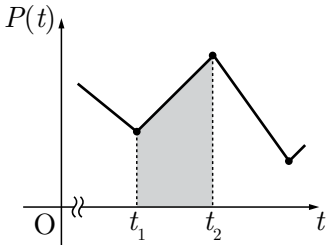


図 3: eTracker で計測した電力データの例

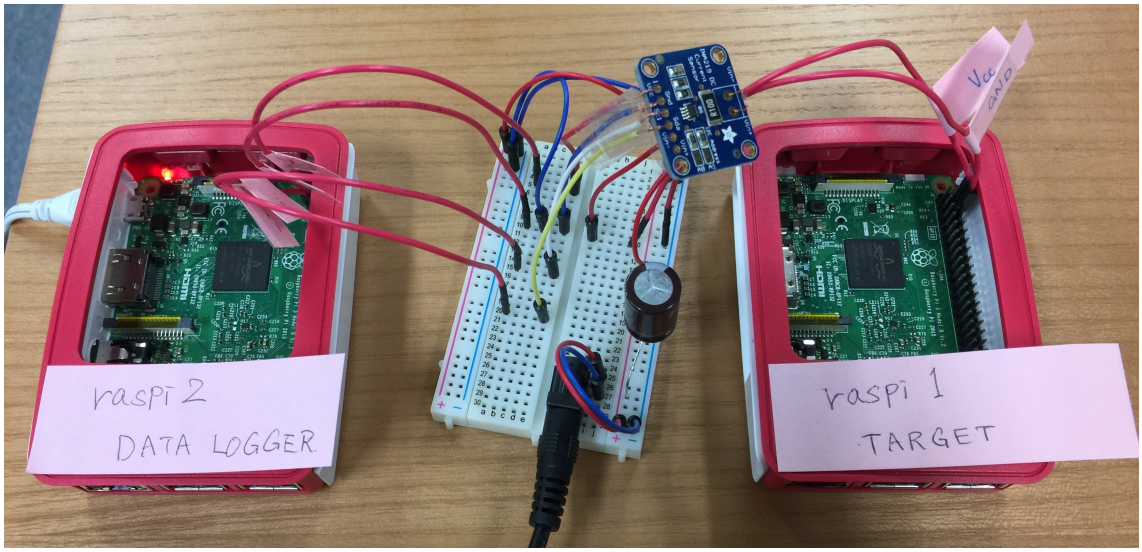


図 4: eTracker の全体図

れば，この区間における総消費電力量は，図 3 の影のついた部分の面積

$$\frac{1}{2}\{P(t_1)+P(t_2)\}(t_2-t_1) \tag{4}$$

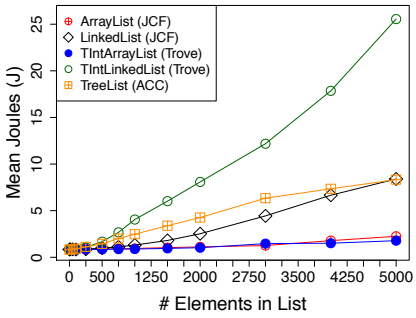
で近似できる．実際に計測する際は，t_2-t_1 が約 1 [ms] となるように設定した．

式 (4) を，プログラムの実行を開始した時刻 t_{begin} から終了した時刻 t_{end} の間に得られたすべての区間について適用し，それらを足し合わせた値

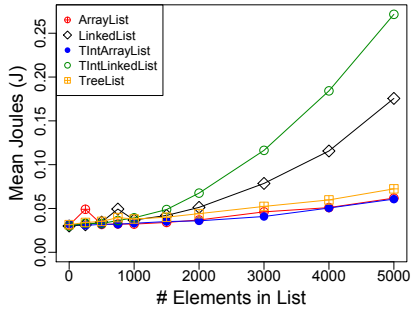
$$\frac{1}{2}\sum_{k=1}^{n-1}[\{P(t_k)+P(t_{k+1})\}(t_{k+1}-t_k)]\ (t_1=t_{\text{begin}}, t_n=t_{\text{end}}) \tag{5}$$

が，プログラムの総消費電力量となる．

本節の終わりに，実際に制作した eTracker の全体図を図 4 に示す．



(a) Hasan らの実験結果[*5]



(b) 再現実験の eTracker での計測結果

図 5: Hasan らの実験の再現結果

4.1.2 eTracker の妥当性の検証

eTracker の妥当性を確かめるため，Hasan らの研究 [14] で行われた実験の一部を再現して eTracker で計測し，Hasan らと同様の結果が得られるかどうかを確かめた．この実験は次のようなものである．Java の `List` インターフェースにおける主要な 5 つの実装について，n 個の要素（$1 \leq n \leq 5000$）をリストの中間に挿入したときの総消費電力量を計測・比較する．ここで対象とする実装クラスは，2.3.2 節でも挙げた，`ArrayList`, `LinkedList`, `TIntArrayList`, `TIntLinkedList` および `TreeList` の 5 つである．図 5 に実験の結果を示す．図 5(a) は Hasan らが示したグラフ，図 5(b) は eTracker を用いた再現実験で得られた結果である．各図における横軸はリストに挿入する要素の数 n [個]，縦軸は総消費電力量 [J] であり，各クラスについて，n の数を変化させたときの総消費電力量の変化の様子を示している．なお，ここで示している総消費電力量は，20 回試行したときの平均値である．

図 5 からもわかる通り，eTracker により，Hasan らグラフとほぼ同じ傾向のグラフを得ることができた．この結果から，eTracker の計測結果には妥当性があると判断した．

念のため，Hasan らのグラフと，我々のグラフとの差異について考察する．両者の相違点として，縦軸（総消費電力量）の範囲およびスケールが異なる点や，`TreeList` クラスの傾向などが挙げられる．このような違いがうまれた原因として，Hasan らとの実験環境の違いが考えられる．前述したように，Hasan らの用いた GreenMiner では TARGET が Android 端末であり，eTracker では TARGET が Raspberry Pi である．また，Hasan らは用いた Java のバージョンや，サードパーティ製のクラスである `TIntArrayList`, `TIntLinkedList` および `TreeList` のバージョンを明記していない．したがって，今回の実験にあたり，Hasan らのときとは異なるバージョンのものを用いた可能性がある．

[*5] 出典：S. Hasan et al. Energy Profiles of Java Collections Classes [14].

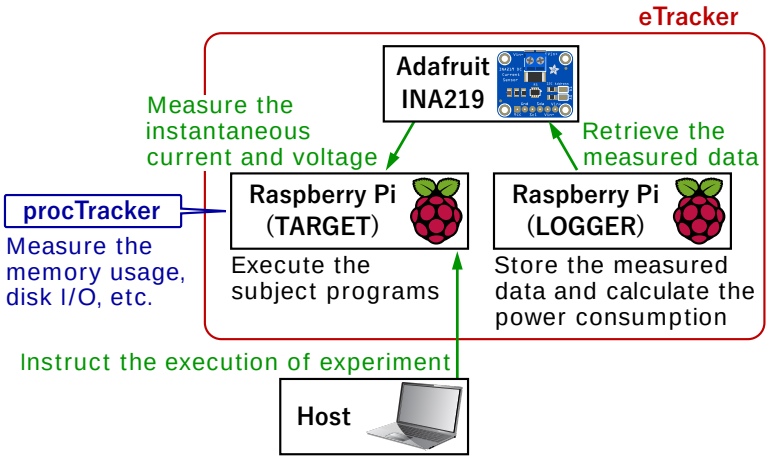


図 6: 実験環境の概要

4.2 計算機リソースの計測ツール：procTracker

Unix システムのメモリ使用量や空き容量などを取得するため既存のコマンドとしては，`free` コマンドなどが有名である．しかし，こういったコマンドを数ミリ秒単位で実行するとなると，そのプロセスの生成に多大なコストがかかり，実験にて電力を計測する際に大きなノイズとなることが予想される．そこで我々は，Raspberry Pi 全体のメモリ使用量などを同一プロセスで定期的に計測することができる，procTracker[*6] を開発した．

Raspberry Pi が搭載する Raspbian をはじめ，Unix 系列の多くのシステムでは procfs [12]（process filesystem）を搭載している．procfs は，`/proc` ディレクトリ下に，プロセスに関するカーネル情報をまとめた擬似ファイル群を提供する．procTracker は，これらを解析することで，前述のメモリ使用量を計測している．

なお，procTracker の妥当性については，前述した `free` コマンドなどが同じく`/proc` ディレクトリ下のファイルを解析している [4] ことから，自明とした．

4.3 実験の流れ

実際に実験を行う際は，前節，および前々節で述べた eTracker, procTracker を組み合わせて，図 6 のような環境を構築した．なお，procTracker については，必要のないときは稼働させないことで，電力の計測へのノイズをできるだけ軽減する．また，表 3 は，TARGET の環境についての詳細である．

[*6] オープンソースソフトウェアとして公開している：`https://github.com/h-matsuo/procTracker`.

5　実験

本章では，前章で述べた実験環境を用いて，何の題材について，どのように実験を行い，その結果がどうなったのかについて述べる．

5.1　実験対象

対象 1．ソーティングアルゴリズム

2.2 節の表 1 に示した 8 つのアルゴリズムのうち，シェルソートを除いた 7 つのアルゴリズムについてそれぞれ C 言語で実装し，プログラムの実行を開始してから終了するまでの間について計測する．ここでシェルソートを省いた理由は，アルゴリズムを $O(n \log n)$ および $O(n^2)$ の 2 つのグループに分けるためである．

表 4 に，ソーティングアルゴリズムを対象とする実験において制御するパラメータの一覧を示す．

対象 2．Java Collections クラス

Java の Collections クラスのうち，とりわけ `List` インターフェースの主要な実装クラスについて，特定の操作を行うプログラムをそれぞれ実装し，プログラムの実行を開始してから終了するまでの間について計測する．ここで対象としたクラスは，4.1.2 節で eTracker の妥当性を検証する実験を行った際にも用いた 5 種類であり，Hasan らの研究 [14] でも用いられている．なお，`TIntArrayList` および `TIntLinkedList` はサードパーティ製ライブラリの Trove [6] に定義されており，それぞれパッケージ `gnu.trove.list.array` および `gnu.trove.list.linked` に含まれる．同様に，`TreeList` はサードパーティ製ライブラリ Apache Commons Collections [1]（ACC）に定義されており，パッケージ `org.apache.commons.collections4.list` に含まれる．

表 5 に，Java Collections クラスを対象とする実験において制御するパラメータの一覧を示す．また，表 6 は実験で用いた Java 環境のバージョン，表 7 は用いたサードパーティ製ライブラリのバージョンの一覧である．

表 3: 実験環境（TARGET）についての詳細

項目	説明
OS	Raspbian 8.0 (Jessie)
CPU	1.2GHz 64-bit quad-core ARMv8
メモリ	1GB RAM

なお，各試行において，実際に操作を行うクラスの種類にかかわらず，プログラムの開始直後にすべてのクラスをインスタンス化することとする．たとえば，実際には `ArrayList` に対してのみ操作を行う試行の場合でも，残りの 4 つのクラスをすべてインスタンス化する．これは，各クラスのインスタンス化にかかるノイズを排除し，リストへの操作のみに関する純粋な比較を行うためである．

表 4: 実験対象：ソーティングアルゴリズムの制御パラメータ

パラメータ	取りうる値
アルゴリズム	ヒープソート，マージソート，クイックソート，バブルソート， インサーションソート，セレクションソート，シェーカーソート
入力データサイズ n	1, 250, 500, 750, ..., 5000

表 5: 実験対象：Java Collections クラスの制御パラメータ

パラメータ	取りうる値
実装クラス	`ArrayList`, `LinkedList`, `TIntArrayList`, `TIntLinkedList`, `TreeList`
操作	リストの冒頭への挿入，リストの中間への挿入
リストに挿入する要素数 n	1, 250, 500, 750, 1000, 1500, 2000, 3000, 4000, 5000

表 6: 実験で用いた Java 環境のバージョン

項目	バージョン
Java SE Runtime Environment	1.8.0_65-b17
Java HotSpot Client VM	25.65-b01, mixed mode

表 7: サードパーティ製 Java Collections ライブラリのバージョン

ライブラリ	バージョン
Trove（`TIntArrayList`, `TIntLinkedList`）	3.0.3
Apache Commons Collections（`TreeList`）	4.1

5.2 実験方法

3.2 節でたてた仮説 H_1 〜 H_3 を検証するため，以下の実験を行う．

実験 1．総消費電力量と実行時間についての調査（仮説 H_1, H_2）

各実験対象（ソーティングアルゴリズムおよび Java Collections クラス）について，各プログラムを実行したときの総消費電力量および実行時間を計測する．試行回数は 50 回とする．なお，無用なノイズが実験データに含まれるのを回避するため，procTracker は稼働させない．

仮説 H_1 の検証としては，計測した総消費電力量と実行時間との間の相関関係を調べる．

仮説 H_2 の検証としては，とくにソーティングアルゴリズムの入力データ数 $n = 5000$ の場合において，各アルゴリズムの組についてその単位時間あたりの消費電力量に差があるのかどうかを，統計的な手法（有意差検定）も用いながら調査する．

実験 2．総消費電力量と平均メモリ使用量についての調査（仮説 H_3）

各実験対象（ソーティングアルゴリズムおよび Java Collections クラス）について，procTracker も用いて，各プログラムを実行したときの総消費電力量および平均メモリ使用量を計測する．試行回数は 50 回とする．

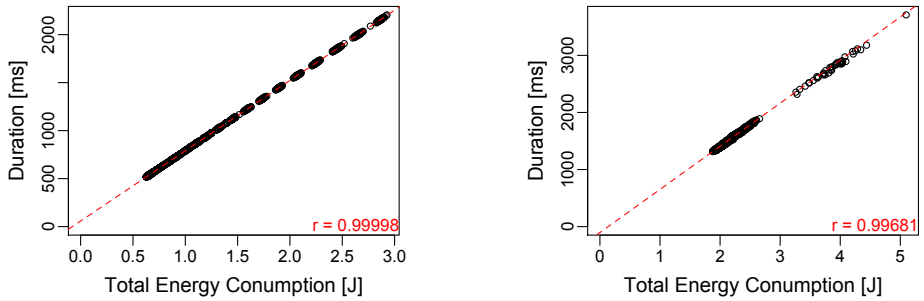
仮説 H_3 の検証としては，計測した総消費電力量と平均メモリ使用量との間の相関関係を調べる．

5.3 実験結果

実験 1．総消費電力量と実行時間についての調査（仮説 H_1, H_2）

図 7 は実験 1 の結果の概要を散布図として示したものである．図 7(a) はソーティングアルゴリズムについて，図 7(b) は Java Collections クラスについての結果をそれぞれ表している．各図において，横軸は総消費電力量 [J]，縦軸は実行時間 [ms] である．各点は各試行における計測データを示しており，たとえば図 7(a) 中のある一点は，アルゴリズム：クイックソート，入力データサイズ：500，試行：20 回目についてのデータを表す．また，図 7(b) 中のある一点は，クラス：`ArrayList`，操作：リストの中間への挿入，リストに挿入する要素数：500，試行：20 回目についてのデータを表す．なお，各散布図中に示した赤色の破線は，各標本における回帰直線を表す．また，算出した相関係数 r を右下に赤字で示している．破線および右下の値の意味については，以降の散布図でも同様である．

図 7 より，ソーティングアルゴリズムおよび Java Collections クラスのどちらについても，総消費電力量と実行時間との間に，右上がりの明確な相関がみられる．また，相関係数 r はどちらの実験対象でも 0.9 を超えており，総消費電力量と実行時間との間には強い正の相関があるといえる．



(a) 実験対象：ソーティングアルゴリズム　　(b) 実験対象：Java Collections クラス

図 7: 実験 1 の結果（概要）

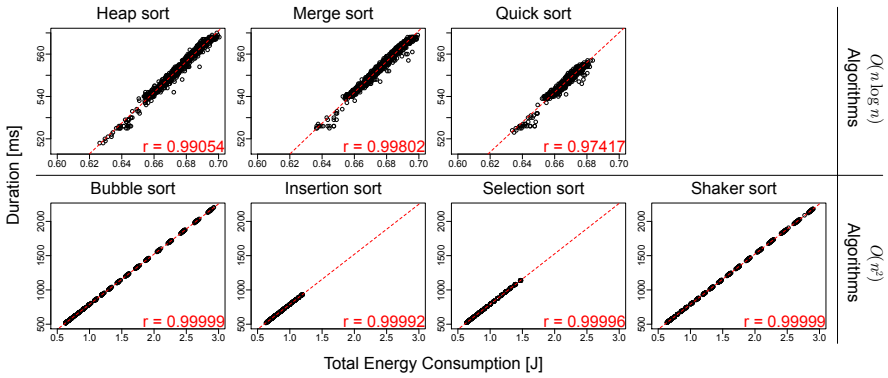


図 8: 実験 1：ソーティングアルゴリズムの結果の詳細

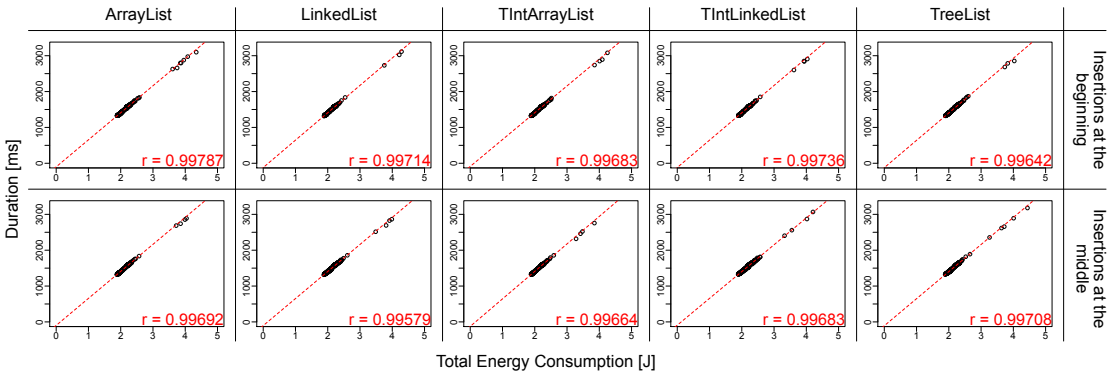


図 9: 実験 1：Java Collections クラスの結果の詳細

また，図 8 は図 7(a) をアルゴリズムごとに，図 9 は図 7(b) をクラスおよび操作ごとに，それぞれ分類してプロットしなおしたものである．なお，図 8 については，上段に $O(n \log n)$ のアルゴリズム，下段に $O(n^2)$ のアルゴリズムをそれぞれまとめており，上段と下段で縦軸・横軸の範囲およびスケールがそれぞれ異なることに注意されたい．

図 8 より，$O(n \log n)$ のアルゴリズム（上段）と $O(n^2)$ のアルゴリズム（下段）では，分布の範囲に大きな違いがあることがわかる．たとえば，総消費電力量（横軸）については，$O(n \log n)$ のアルゴリズムではおよそ 0.6 〜 0.7 [J] であるのに対し，$O(n^2)$ のアルゴリズムではおよそ 0.6 〜 3 [J] である．また，実行時間（縦軸）については，$O(n \log n)$ のアルゴリズムではおよそ 520 〜 570 [ms] であるのに対し，$O(n^2)$ のアルゴリズムではおよそ 520 〜 2,200 [ms] である．

さらに，図 8 より，$O(n \log n)$ のアルゴリズムと $O(n^2)$ のアルゴリズムでは，分布のしかたにも若干の違いがみられることがわかる．$O(n \log n)$ のアルゴリズムは完全な右上がりではなく，若干のばらつきがみられるが，$O(n^2)$ のアルゴリズムはほとんどばらつきが見られず，相関係数もほぼ 1 となっている．また，$O(n^2)$ のアルゴリズムでも，バブルソートおよびシェーカーソートと，インサーションソートおよびセレクションソートとの間で，分布の範囲に違いがみられる．

図 9 からは，Java のクラス間では，分布の範囲やしかたやばらつきについて，ソーティングアルゴリズムほどの大きな差が見られないことがわかる．

表 8 は，各ソーティングアルゴリズムの組ごとに，(a) 単位時間あたりの消費電力量の平均値の差（表の上側のアルゴリズムのものから，表の右側のアルゴリズムのものを引いた値），(b) 各組について有意差検定を行った結果，のそれぞれを一つにまとめたものである．検定により有意差があると認められたものには * 印を付してある．検定手法としては，(1) 母集団が正規分布に従うかどうか不明であり，(2) 標本間には対応があることから，ノンパラメトリックかつ対応ありの有意差検定である，ウィルコクソンの符号付き順位検定（Wilcoxon signed-rank test）[19] を採用した．有意水準 α は 0.01 とした．な

表 8: 各組ごとの (a) 単位時間あたりの消費電力量の平均値の差，および (b) 有意差検定の結果

$O(n \log n)$			$O(n^2)$				
heap	merge	quick	bubble	insertion	selection	shaker	
—	4.01×10^{-7}	3.78×10^{-6}*	-1.03×10^{-4}*	-5.51×10^{-5}*	-6.65×10^{-5}*	-1.03×10^{-4}*	heap
	—	3.38×10^{-6}*	-1.03×10^{-4}*	-5.55×10^{-5}*	-6.70×10^{-5}*	-1.03×10^{-4}*	merge
		—	-1.07×10^{-4}*	-5.89×10^{-5}*	-7.03×10^{-5}*	-1.07×10^{-4}*	quick
			—	4.79×10^{-5}*	3.65×10^{-5}*	-5.49×10^{-8}	bubble
				—	-1.14×10^{-5}*	-4.79×10^{-5}*	insertion
					—	-3.65×10^{-5}*	selection
						—	shaker

*：有意差があると認められたもの．有意水準 $\alpha = 0.01$ とした．

お，(1) については，「標本が正規母集団に由来する」という帰無仮説を検定するシャピロ-ウィルク検定（Shapiro-Wilk test）[13] を各標本について実施したところ，バブルソート，インサーションソートおよびセレクションソートの各標本について，それぞれ帰無仮説が棄却されたためである（有意水準 $\alpha = 0.01$）．(2) については，50 回の全試行のうち，各試行においては，それぞれのアルゴリズムにおいてまったく同じデータを入力として与えているためである．

表 8 より，$O(n \log n)$ のアルゴリズムと $O(n^2)$ のアルゴリズムとを比較した場合に比べて，$O(n \log n)$ のアルゴリズム同士，また $O(n^2)$ のアルゴリズム同士をそれぞれ比較した場合のほうが，単位時間あたりの消費電力量の差が小さい傾向にあることがわかる．また，有意差検定の結果，マージソート-ヒープソートの組，シェーカーソート-バブルソートの組についてはそれぞれ有意差があるとは認められなかったが，それ以外のアルゴリズムの組については，すべて有意差があると認められた．

実験 2．消費電力量と計算機リソースについての調査（仮説 H_3）

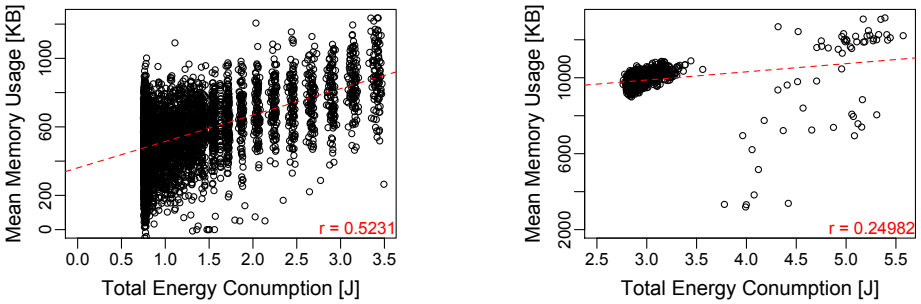
図 10 は実験 2 の結果の概要を散布図として示したものである．図 10(a) はソーティングアルゴリズムについて，図 10(b) は Java Collections クラスについての結果をそれぞれ表している．各図において，横軸は総消費電力量 [J]，縦軸は平均メモリ使用量 [KB] である．

図 10(a) からは若干の右上がりの相関が読みとれるが，ばらつきが非常に大きいことが見てとれる．また，図 10(b) では，図 10(a) に比べるとばらつきが少ないが，図の形状，および相関係数 r のどちらからも，相関があるかどうかを判定することが難しい．そこで，それぞれの標本について無相関検定を行った（有意水準 $\alpha = 0.01$）．検定の結果，ソーティングアルゴリズムおよび Java Collections クラスの両方について，相関係数が有意であることが認められた．よって，総消費電力量と平均メモリ使用量との間について，ソーティングアルゴリズムでは正の相関が，Java Collections クラスでは弱い正の相関があるといえる．

図 11 は図 10(a) をアルゴリズムごとに，図 12 は図 10(b) をクラスおよび操作ごとに，それぞれ分類してプロットしなおしたものである．実験 1 と同様，図 11 については，上段と下段で縦軸・横軸の範囲およびスケールがそれぞれ異なることに注意されたい．

図 11 から，実行時間（図 8）と比べて，平均メモリ使用量については $O(n \log n)$ のアルゴリズムと $O(n^2)$ のアルゴリズムとで分布の範囲にあまり違いはないことがわかる．ただし，$O(n \log n)$ のアルゴリズムと $O(n^2)$ のアルゴリズムとで分布のしかたは大きく異なっており，$O(n \log n)$ のアルゴリズムでは円状に広く分布しているのに対し，$O(n^2)$ のアルゴリズムでは縦軸方法の分布が多くみえる．

実験 1 と同様，図 12 から，Java のクラス間では分布の範囲やしかた，ばらつきについて，ソーティングアルゴリズムほどの大きな差が見られないことがわかる．



(a) 実験対象：ソーティングアルゴリズム　　(b) 実験対象：Java Collections クラス

図 10: 実験 2 の結果（概要）

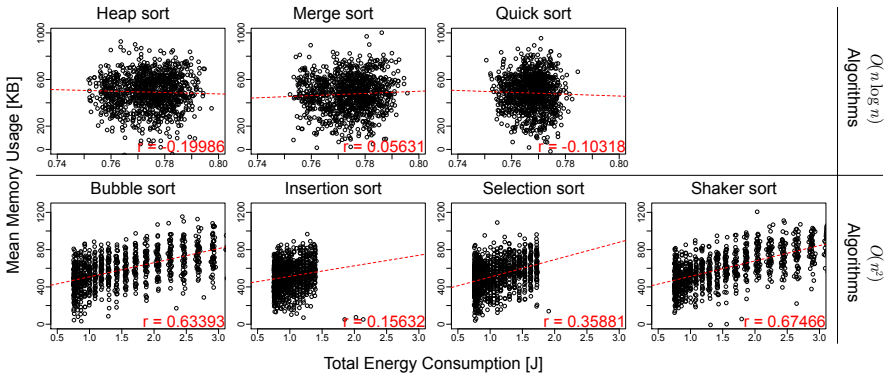


図 11: 実験 2：ソーティングアルゴリズムの結果の詳細

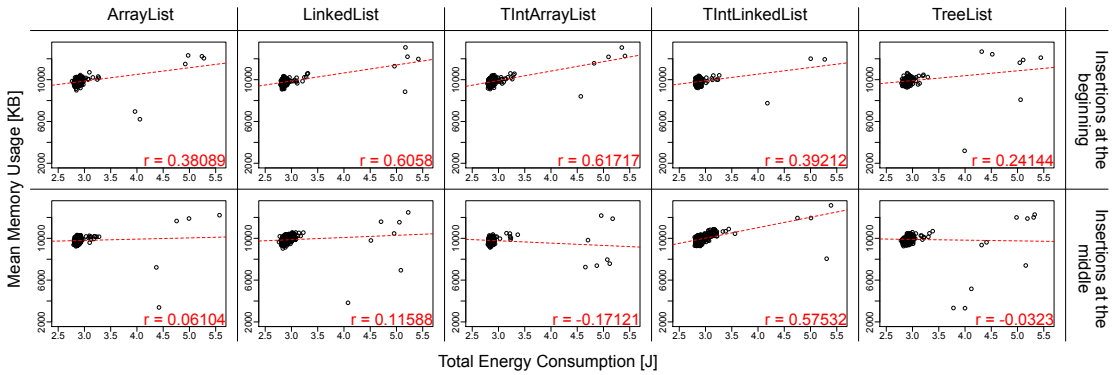


図 12: 実験 2：Java Collections クラスの結果の詳細

6　議論

本章では，前章で示した実験結果を考察し，3.2 節で提示したそれぞれ仮説の真偽を述べる．

仮説 H_1　総消費電力量と実行時間との間には正の相関がある．

結論　仮説は正しい．相関係数 r は 0.9 を超えており，両者には強い正の相関がある．

実験 1 の結果，とくに図 7 〜 9 より，ソーティングアルゴリズムおよび Java Collections クラスのいずれの場合においても，総消費電力量と実行時間との相関係数 r は 0.9 を超えており，強い正の相関が存在することがわかった．よって，仮説 H_1 は真であるといえる．

このことから，開発者がプログラムの総消費電力量を削減したい場合，実行時間が短くなるような実装手段を選択すればよいといえる．

仮説 H_2　実装手段により，単位時間あたりの消費電力量は異なる．

結論　仮説は正しい．ほぼすべてのソーティングアルゴリズムの間に有意差が認められ，とくに異なる時間計算量の間では顕著である．

実験 1 の結果，とくに表 8 より，有意差検定の結果，ほぼすべてのソーティングアルゴリズム間に有意差があることがわかった．なお，前述したように，有意差が認められなかったアルゴリズムの組は，マージソート-ヒープソートおよびシェーカーソート-バブルソートの 2 組のみである．これらはどちらも同じオーダーのアルゴリズムの組であるため，有意差が検出されなかったと考えられる．よって，仮説 H_2 は真であり，とくに異なる時間計算量の実装手段の間では顕著といえる．

このことは，実行時間だけが消費電力量を支配する要因ではないことを示している．また，総消費電力量と実行時間との間に完全な比例関係が存在するわけではないため，総消費電力量の〈最適化〉を考える場合には，Bunse らや Hindle らの研究のように，実装ごとの総消費電力量の違いやその理由を考える必要があるといえる．

仮説 H_3　総消費電力量と平均メモリ使用量との間には正の相関がある．

結論　仮説は正しい．ただし，仮説 H_1（総消費電力量と実行時間との相関）よりは強くない．

実験 2 の結果より，総消費電力量と平均メモリ使用量との間には，ソーティングアルゴリズムでは正の相関が，Java Collections クラスでは弱い正の相関がみられることがわかった．ソーティングアルゴリズムでの相関係数 $r \approx 0.52$ に比べて，Java Collections クラスでは相関係数が $r \approx 0.25$ と小さかっ

た理由として，Java 仮想マシン（JVM）の存在が挙げられる．Java では，プログラムはすべて JVM 上で実行されるため，プログラムの実行とともに JVM も起動する．このとき，JVM は実行するプログラム用のヒープ領域として，一定量のメモリを常にシステムから確保する．このように，Java Collections クラスを対象とした実験では，実装内容に関係なく常に一定量のメモリが確保されるため，相関係数の値が小さくなったと考える．以上の考察より，仮説 H_3 は真であるが，仮説 H_1 の総消費電力量と実行時間との相関に比べると，総消費電力量と平均メモリ使用量との相関は強くないといえる．

このことから，開発者がプログラムの総消費電力量を削減したい場合，平均メモリ使用量が小さくなるような実装手段を選択することも有効であるといえる．ただし，仮説 H_1 が真であることより，もっとも優先するべきなのは実行時間の短さであり，実行時間がほぼ同じような実装手段が複数存在する場合に，より平均メモリ使用量が少ないものを選べばよいといえる．

7　妥当性への脅威

本章では，本研究における妥当性への脅威について述べる．

本研究では，総消費電力量や平均メモリ使用量を計測するプラットフォーム（TARGET）として，Raspberry Pi 3 Model B を選択している．したがって，TARGET を変更すると，調査結果が変わる可能性がある．また，TARGET 上で実行されているプログラムのうち，実験に関係のないプログラムは計測前にできるだけ停止しているが，計測ツール eTracker および procTracker やシステムの動作に必要な各種デーモンなど，実験対象以外のプログラムも電力を消費している．さらに，実験の都合上，開発した procTracker は TARGET のシステム全体における平均メモリ使用量を計測しており，実験対象のプロセスのみの平均メモリ使用量については計測できていない．以上より，計測結果には様々なノイズが含まれており，より厳密な議論にはさらなるノイズの削減が必要である．

本研究では，実験対象を表 4 に示す 7 つのソーティングアルゴリズム，および表 5 に示す 5 つの `List` クラスおよび操作としている．その他のアルゴリズムやクラス，プログラミング言語については議論していないため，これらを変更すると，調査結果が変わる可能性がある．

8　あとがき

本研究では，実行時間と総消費電力量との間には強い相関があるという仮説のもと，独自の電力消費計測基盤 eTracker を用いて，実験対象のプログラムを実行している間の実行時間および総消費電力量をそれぞれ計測し，両者の相関を調べた．また，同様に，平均メモリ使用量と総消費電力量との間の相関についても調査した．実験の結果，実行時間と総消費電力量との間には強い正の相関が存在し，平均メモリ使用量と総消費電力量との間にも正の相関が存在した．この結果から，プログラムの開発者がその総消費電力量を削減したい場合，実行時間の短さ，平均メモリ使用量の少なさの順に考慮し，実装手段を選択すればよいことがわかった．

今後の課題として，より実践的なプログラムにおける，実行時間と総消費電力量との相関関係の調査が挙げられる．本研究では，ソーティングアルゴリズムや Java の実装クラスといった，実践的なソフトウェアと比べてアトミックな要素を実験対象とした．本研究で得られた成果が，実際に広く使われているライブラリやアプリケーションにおいても適用されるのか，調査したい．また，本研究では，計算機リソースとして平均メモリ使用量を取りあげた．しかし，物理ディスク I/O やネットワーク通信など，その他の計算機リソースについてのそれぞれと総消費電力量との相関がわかれば，開発者に対して，総消費電力量を削減するためのより詳細な指針を示すことができる．さらに，計算機リソースの使い方の違いが総消費電力量に与える影響の調査も考えられる．たとえば，ソフトウェアにおいて，実行時のログをストレージに書き出すことは広く行われているが，これを毎回書き出すのか，またはメモリにある程度蓄えてから一度に書き出すのかで，それぞれ総消費電力量がどれほど異なるのかを調べることができる．

本研究成果の実用化例として，コンパイラへの総消費電力量を削減するオプションの追加が挙げられる．コンパイル時，出力が同じだが実装手段が複数存在する場合において，コンパイラがより実行時間や平均メモリ使用量が少ない実装を適切に選ぶことができれば，総消費電力量を削減することが可能となる．

謝辞

本研究を行うにあたり，理解あるご指導を賜り，常に励ましていただきました楠本真二教授に，心より感謝申し上げます.

本研究に関して，議論の中で貴重なご助言をいただきました肥後芳樹准教授に，深く感謝申し上げます.

本研究の全過程を通して，研究に対する考え方や方向性など，丁寧かつ熱心なご指導を賜りました柗本真佑助教に，心より感謝申し上げます.

本研究に関して，独自の電力消費計測ツール eTracker の開発にあたり，部品選定や電気回路の設計などで多大なご助力を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程 2 年生の鷲見創一氏，および同大学基礎工学部情報科学科 4 年生の有馬諒氏に，深く感謝申し上げます.

本研究を進めるにあたり，様々な形で励まし，ご助言を頂きましたその他の楠本研究室の皆様のご協力に深く感謝致します.

最後に，本研究に至るまでに，講義，演習，実験等でお世話になりました大阪大学基礎工学部情報科学科の諸先生方に，この場を借りて心から御礼申し上げます.

参考文献

[1] Apache commons collections. https://commons.apache.org/proper/commons-collections/.

[2] Ina219 high side dc current sensor breakout. https://www.adafruit.com/product/904.

[3] Java collections framework. https://docs.oracle.com/javase/8/docs/technotes/guides/collections/.

[4] Linux filesystem hierarchy. http://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html.

[5] Raspberry pi 3 model b. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[6] Trove. http://trove.starlight-systems.com/.

[7] 浅野哲夫, 和田幸一, 増澤利光. アルゴリズム論. IT text / 情報処理学会編. オーム社, 2003.

[8] Kenneth G. Brill. Data center energy efficiency and productivity. Technical report, The Uptime Institute, Santa Fe, NM, USA, 2007.

[9] Christian Bunse, Hagen Höpfner, Essam Mansour, and Suman Roychoudhury. Exploring the energy consumption of data sorting algorithms in embedded and mobile environments. In *Proceedings of the 10th International Conference on Mobile Data Management*, pp. 600–607. IEEE, 2009.

[10] Christian Bunse, Hagen Höpfner, Suman Roychoudhury, and Essam Mansour. Choosing the "best" sorting algorithm for optimal energy consumption. In *Proceedings of the 4th International Conference on Software and Data Technologies*, Vol. 2, pp. 199–206, 2009.

[11] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, Vol. 91, No. 2, pp. 201–213, 2002.

[12] Roger Faulkner and Ron Gomes. The process file system and process model in unix system v. In *USENIX Winter*, pp. 243–252. USENIX, 1991.

[13] Asghar Ghasemi and Saleh Zahediasl. Normality tests for statistical analysis: a guide for non-statisticians. *International journal of endocrinology and metabolism*, Vol. 10, No. 2, pp. 486–489, 2012.

[14] Samir Hasan, Zachary King, Munawar Hafiz, Mohammed Sayagh, Bram Adams, and Abram Hindle. Energy profiles of java collections classes. In *Proceedings of the 38th International Conference on Software Engineering*, pp. 225–236, 2016.

[15] Abram Hindle, Alex Wilson, Kent Rasmussen, E. Jed Barlow, Joshua Charles Campbell, and Stephen Romansky. Greenminer: A hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 12–21. ACM, 2014.

[16] James M. Kaplan, William Forrest, and Noah Kindler. Revolutionizing data center energy efficiency. Technical report, McKinsey & Company, 2008.

[17] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, 1998.

[18] Robert Sedgewick. アルゴリズムC 新版 基礎 データ構造 整列 探索. 近代科学社, 2004. 野下 浩平, 星 守, 佐藤 創, 田口 東 訳. 原著：Algorithms in C, Parts 1-4.

[19] Paul Teetor. R クックブック. オライリー・ジャパン, 2011. 大橋 真也 監訳, 木下 哲也 訳. 原著：R Cookbook.

[20] TY Zhang and Ching Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, Vol. 27, No. 3, pp. 236–239, 1984.