情報処理学会論文誌 Vol.59 No.4 1262–1272 (Apr. 2018)

省電力プログラムの実現に向けて：実装手段の違いによる実行時間と消費電力量の比較調査

松尾 裕幸[1,a)] 松本 真佑[1,b)] 楠本 真二[1,c)]

受付日 2017年8月2日, 採録日 2018年1月15日

概要：近年，ソフトウェアの電力消費の削減のため，API やアルゴリズムなどの実装手段の違いによる電力消費の比較研究がさかんに行われている．しかしながら，我々は，プログラムの実行時間とその総消費電力量との間には，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，電力消費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，異なるソーティングアルゴリズム，および Java の異なる Collections クラスについて，その総消費電力量および実行時間の両方について調査する．実験の結果，両者の間には非常に強い正の相関が存在することが示された．また回帰分析の結果，プログラムの実行時間，単位時間あたりの消費電力量の順に，総消費電力量に与える影響が大きいことが分かった．これらの結果から，電力の計測基盤を用いずとも実行時間が短い実装手段を選ぶことが，プログラム全体の省電力化につながることが示唆された．

キーワード：消費電力，実行時間，プログラム，ソーティングアルゴリズム，Java Collections クラス

Towards Energy-Saving Program Implementation: A Comparative Study on Execution Time and Energy Consumption Based on Different Implementations

Hiroyuki Matsuo[1,a)] Shinsuke Matsumoto[1,b)] Shinji Kusumoto[1,c)]

Received: August 2, 2017, Accepted: January 15, 2018

Abstract: These days, many researchers have conducted studies which compare energy consumption based on the differences of implementation methods, such as API or algorithm, so as to reduce software energy consumption. However, we raise a question that there is a strong correlation between total energy consumption of a program and duration of its execution. If this hypothesis is correct, the problem of reducing energy consumption can be solved by shortening the duration. In this paper, we investigate the relation between total energy consumption and duration about different sorting algorithms and Java Collections classes. Experimental results reveal that there is a strong positive correlation between them. We also find that measured metrics have large impacts on program's energy consumption in the order of its duration and mean energy consumption per unit of time. As a result, we conclude that adopting an implementation with short execution time makes whole program energy-saving.

Keywords: energy consumption, execution time, program, sorting algorithm, Java Collections class

1. はじめに

昨今のスマートデバイスの普及やデータセンターの数の増加にともない，ソフトウェアに対する消費電力量への関心が高まっている．Pinto らの調査によると，StackOverflow [*1] に投稿されている消費電力量に関する質問の数はここ 5 年間でほぼ線形的に増加しており，ソフトウェア開発者の消費電力量に対する強い関心がうかがえる [11]．

1 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
a) h-matsuo@ist.osaka-u.ac.jp
b) shinsuke@ist.osaka-u.ac.jp
c) kusumoto@ist.osaka-u.ac.jp

*1 コンピュータ・プログラミングの広範囲なトピックについて扱っている Q&A サイト，https://stackoverflow.com/

このような背景にともない，ソフトウェアの省電力化を目的とした研究が多数行われている [2], [3], [7]．Bunse らは，ソーティングアルゴリズム間の消費電力量を比較し，メモリ効率の良いインサーションソートが最も電力効率が良いと結論付けている [2], [3]．また，Hasan らは Java の Collections クラスを題材として，その消費電力量を比較し，`List` インタフェースの実装クラスとしては `TIntArrayList` が最も電力効率が良いことを示している [7]．これらの研究は，同機能かつ異なる実装手段のプログラム群を題材としており，省電力化を考慮したソフトウェア開発の 1 つの指針であるといえる．

しかしながら，これらの研究に対する我々の最大の疑問は，「プログラム実行時における総消費電力量は，その実行時間の長さに強く左右されるのではないか？」という点にある．一般に，あらゆる計算機はプログラムを実行することで計算機リソースを使用し電力を消費するため，短い時間で処理を終えることができればその分の消費電力量は削減されると考えられる．また当然ながら，プログラムの実装によって計算機リソース（メモリやディスク，Wi-Fi など）の使用量には差があり，消費電力量を左右する要因となり得る．ただし，その計算機リソースが消費する電力量は計算機全体の消費電力量の一部でしかなく，実行時間以上に消費電力量を支配する要因にはなり得ないと考えられる．

この仮説が正しいとすれば，プログラムの省電力化という課題は，これまでに数多く取り組まれている速度の観点からのプログラム最適化 [4], [12] という問題に帰着させることが可能となる．また，消費電力の計測を行う際は，特殊なデバイスや環境 [8] を準備する必要がある．一方で，消費電力量と実行時間に強い相関があるならば，時間というきわめて計測容易な尺度を消費電力量の代替として用いることが可能となる．

本研究はこの仮説に基づき，同じ機能かつ異なる実装手段の様々なプログラムに対して，その総消費電力量と実行時間との関係を調査する．調査における 3 つの具体的な仮説は以下のとおりである．

H_1 総消費電力量と実行時間との間には強い相関がある．

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．

H_3 総消費電力量と平均メモリ使用量との間には相関がある．

仮説 H_1 は先の疑問と同等である．仮説 H_2 では，総消費電力量ではなく単位時間あたりの消費電力量に着目し，時間という要因を省いたうえで，実装手段の違いによって消費電力量が異なるかどうかを確かめる．仮説 H_3 では，仮説 H_2 が正しいと仮定した際に，どのような計算機リソースが総消費電力量に影響しているかを調査する．本稿では計算機リソースの 1 つとして，メモリ使用量のみに着目しその仮説を確かめる．

調査方法としては，先にあげた Bunse らの実験 [2], [3] と Hasan らの実験 [7] の一部を再現する形で行う．具体的には，7 つのソーティングアルゴリズムと 5 つの Java Collections クラスを題材として，それらの総消費電力量と実行時間を計測し，仮説の検証を行う．消費電力の計測環境としては，GreenMiner [8] を参考に開発した独自の電力消費計測システム eTracker，および，計算機リソース計測ツール procTracker を用いる．

調査の結果，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，仮説 H_1 は正しいことが示された．さらに，統計的な検定により仮説 H_2，H_3 の両方についても正しいことが分かった．また回帰分析の結果，総消費電力量に与える影響は，実行時間，単位時間あたりの消費電力量の順に大きいことが示された．これらの結果から，開発者がプログラムの総消費電力量を削減したい場合，たとえ電力計測設備を所持していなくても，実行時間の短い実装手段を優先的に選択すればよいことが分かった．

2. 準備

2.1 電気に関する諸用語の定義

直流回路について考える．ある時刻 t において計測された瞬時電流を $I(t)$，瞬時電圧を $E(t)$ とすると，その瞬時電力 $P(t)$ は次式で与えられる．

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

$P(t)$ の単位としては [W] が用いられる．

さらに，（消費）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られる．すなわち，

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

W の単位としては [J] や [W·s] が一般的に用いられる．

また一般に計算機はアイドル時，すなわち定常状態でも一定の電力を消費する．この定常電力を P_{steady} とし，t_{begin}，t_{end} をそれぞれプログラムが起動，終了した時刻と定義する．このとき，プログラム実行による総消費電力量は以下の式で表される．

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

以降，本稿では，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割った値を「単位時間あたりの消費電力量」とそれぞれ定義する．

表 1　主要なソーティングアルゴリズムの概要
Table 1　Overview of major sorting algorithms.

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	—

2.2　ソーティングアルゴリズム

表 1 に，主要な 8 つのソーティングアルゴリズムとその概要を示す [1], [15]．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　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 [2], [3]．対象としているアルゴリズムは，表 1 であげた 8 つである．Bunse らはマイクロコントローラ（いわゆるマイコン）上でプログラムを実行し，その 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
- メモリ効率の良い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良いこと

などを指摘している．

2.4　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 [7]．`List` の実装としては，Java Collections Framework（JCF）に含まれる `ArrayList` および `LinkedList`，Trove [*2] に含まれる `TIntArrayList` および `TIntLinkedList`，そして Apache Commons Collections [*3]（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，繰返し処理，ランダムアクセスを行っている．実験には，独自に作成した電力計測基盤 GreenMiner [8] 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．

表 2　`List` インタフェースの主要な各実装クラスによる総消費電力量の違い [7]
Table 2　Differences of total energy consumption between major implementation classes of `List` interface [7].

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`

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，`List` の各実装クラスについての結果の一部を示す．

3.　調査目的と仮説

3.1　関連研究における問題点

前述のとおり，Bunse らは論文 [2] で「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」と述べている．一方で，Hasan らは，論文 [7] 中の議論で「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」と述べている．

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，計算機ではプログラムの動作にともない，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あわせて発生するため，CPU のみが電力を消費するという場面は考えにくい．

Hasan らは，前述した相関関係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 Collections クラス間の総消費電力量の比較分析であり，実行時間を考

*2　http://trove.starlight-systems.com/
*3　https://commons.apache.org/proper/commons-collections/

慮した結果については定量的に述べられていない．

ここで本研究における最大の疑問は，「プログラム実行時における総消費電力量は，その実行時間の長さに強く左右されるのではないか？」という点にある．総消費電力量 [J] は [W·s]，すなわち時間成分を含むメトリクスであり，総消費電力量と実行時間に関係があるのは一見自明である．しかしながら，Bunse ら [2] はこの自明な関係を否定しており，また Hasan ら [7] は定量的な結果を示していない．

もし，この仮説が正しいとすれば，プログラムの省電力化という課題は，実行速度という観点からのプログラム最適化の問題に帰着させることが可能となる．さらに，消費電力量という特殊なデバイスを要する尺度は，実行速度というきわめて計測容易な尺度で代替が可能となる．

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，本稿では，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

仮説 H_1　総消費電力量と実行時間との間には強い相関がある．

計算機は一般に，電力を消費することで計算機リソースを使用する．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削減したいとき，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_3　総消費電力量と平均メモリ使用量との間には相関がある．

2.3 節で述べた，Bunse らの「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」という主張は，計算機全体の総消費電力量について計測した場合にもいえるのかという疑問がうまれた．また，仮説 H_2 が正しいとすれば，単位時間あたりの消費電力量に差が現れた要因は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，本研究では平均メモリ使用量に着目した．

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

本研究では，4.1 節および 4.2 節で述べる eTracker，procTracker を開発・設計し，組み合わせて，**図 1** のような実験環境を構築した．本章では，開発したこれらのツールの概要や，その他の実験環境の詳細について述べる．

4.1 総消費電力量の計測環境：eTracker

我々は，計算機の総消費電力量を計測するための独自システム eTracker [*4] 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 [8] を参考にした．**図 2** に，eTracker の回路図を示す．

eTracker は，次の 3 つのコンポーネントに大きく分けることができる：

(a) Raspberry Pi [*5] 上でプログラムを実行する TARGET

(b) TARGET が消費した電力を計測する INA219

(c) INA219 が計測したデータを収集・処理する LOGGER

INA219 は既製品の電力計測チップであり，GreenMiner で用いられているものと同一である．

TARGET では，実験とは無関係なプログラムが電力を消費することのないように注意しなければならない．可能な限り定常状態での消費電力量を削減するため，TARGET では以下のような設定を施している：

- デスクトップ GUI を利用せず，CLI コンソールでブートする．

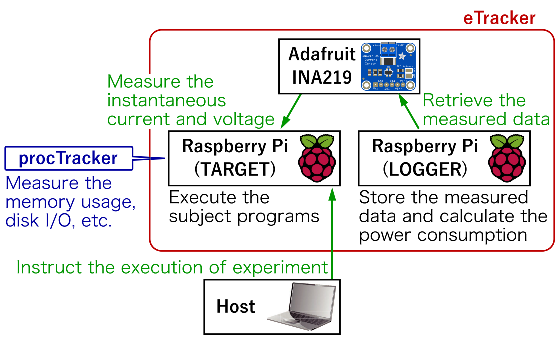


図 1　実験環境の概要

Fig. 1　Overview of experiment environment.

*4 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．

*5 Linux が動作する，小型のコンピュータ．

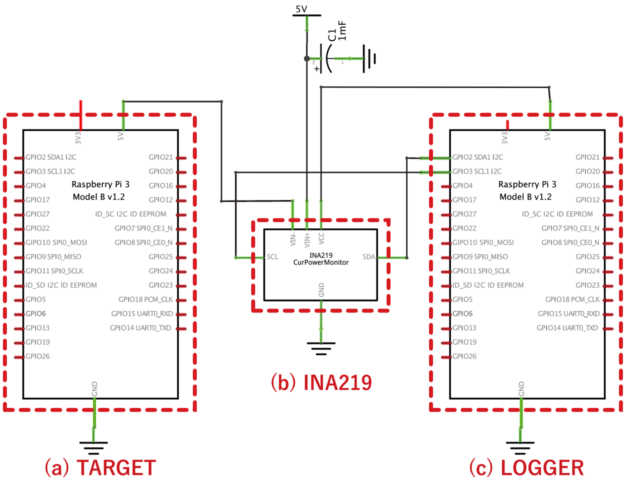


図 2　eTracker の回路図
Fig. 2　Circuit diagram of eTracker.

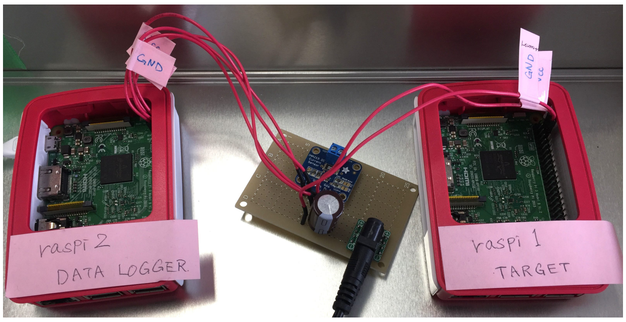


図 3　eTracker の全体図
Fig. 3　Overall view of eTracker.

- ネットワークには有線で接続し，Wi-Fi および Bluetooth は停止する．
- 実験と無関係なプログラムの実行を阻止するため，`networking`，`rsyslog`，`ssh` 以外のすべてのサービス（デーモン）を停止する．
- INA219 を除き，TARGET 本体には周辺機器などを接続しない．

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 [*6] ピンから給電する．また，INA219 で計測した瞬時電流や瞬時電圧といったデータは，$\mathrm{I^2C}$ [*7] 通信で LOGGER に送られる．

定常状態での電力 P_{steady} の値には，アイドル状態における 30 秒間の平均電力を計測し利用する．具体的な P_{steady} の値は，実験 1 では 1.27 [W]，実験 2 では 1.40 [W] であった．ここで，実験 2 では次節で述べる procTracker を利用してメモリ使用量を計測するため，電力が増している．

実際に制作した eTracker の全体図を**図 3** に示す．また，TARGET のハードウェア構成についての詳細を**表 3** に示す．

*6　組み込みデバイスなどが通信を行うための汎用的な入出力ピン．
*7　IC 間でシリアル通信を行うためのインタフェースの 1 つ．

表 3　実験環境（TARGET）のハードウェア構成
Table 3　Hardware specifications of experiment environment.

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

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の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，コマンドを数ミリ秒単位で実行するとそのプロセスの生成に多大なコストがかかり，実験で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 [*8] 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 [5]（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，これらの擬似ファイルを解析することで，メモリ使用量をはじめとした計算機リソースを計測している．

5.　実験

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．なお，各プログラムのソースコードは，Rosetta Code [*9] に記載されているものを使用した．

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ける制御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，Hasan らの研究 [7] でも用いられている 5 種類である．

*8　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.
*9　http://rosettacode.org/wiki/Rosetta_Code

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
Table 4　Controlled parameters for sorting algorithms.

パラメータ	とりうる値
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
入力サイズ n	1，250，500，750，1000，1500，2000，3000，4000，5000

表 5　実験対象：Java Collections クラスの制御パラメータ
Table 5　Controlled parameters for Java Collections classes.

パラメータ	とりうる値
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
Table 6　Versions of Java environment.

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
Table 7　Versions of thrid-party Java Collections library.

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

表 5 に，Java Collections クラスを対象とする実験における制御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．

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の純粋な比較を行うためである．

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．

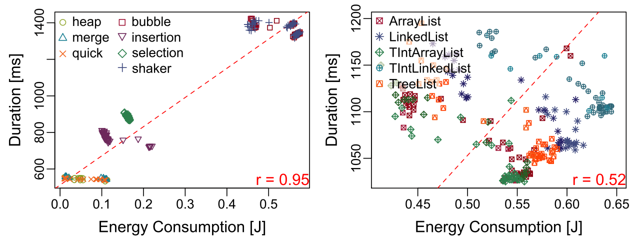
仮説 H_1 の検証としては，計測した総消費電力量と実行時間との間の相関関係を調べる．

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 $n = 5{,}000$ 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，有意差検定を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 も用いながら，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．



ソーティングアルゴリズム　　Java Collections クラス

図 4　実験 1：消費電力量と実行時間に関する結果
Fig. 4　Results of experiment 1: relation between energy consumption and duration.

5.3　実験結果

実験 1. 総消費電力量と実行時間の調査（仮説 H_1，H_2）

実験 1 の結果を図 **4** に示す．x 軸が総消費電力量，y 軸が実行時間をそれぞれ表す散布図であり，簡単のために各プログラムへの入力サイズおよびリストへ挿入する要素数が $n = 5{,}000$ の場合の結果のみを抜粋して記載している．また，各アルゴリズムにおけるすべての n についての詳細を図 **5**，図 **6** に示す．たとえば，図 5 のある一点は，アルゴリズム：クイックソート，入力サイズ：$n = 2{,}500$，試行：5 回目についての結果を示している．

なお，各図の右下に記載している r は，各標本について算出した相関係数である．また，図 5 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いる．

図 4 より，実行時間の短い実装手段はその消費電力量も小さく，両者の間には強い正の相関があることが分かる．また相関係数から，Java Collections クラスよりもソーティングアルゴリズムの方が，より顕著にこの傾向があることが示された．

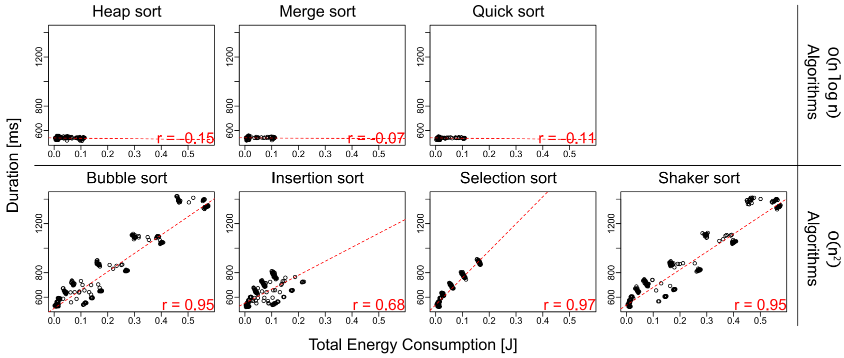


図 5　実験 1：ソーティングアルゴリズムについての詳細

Fig. 5　Detailed results of experiment 1: sorting algorothms.

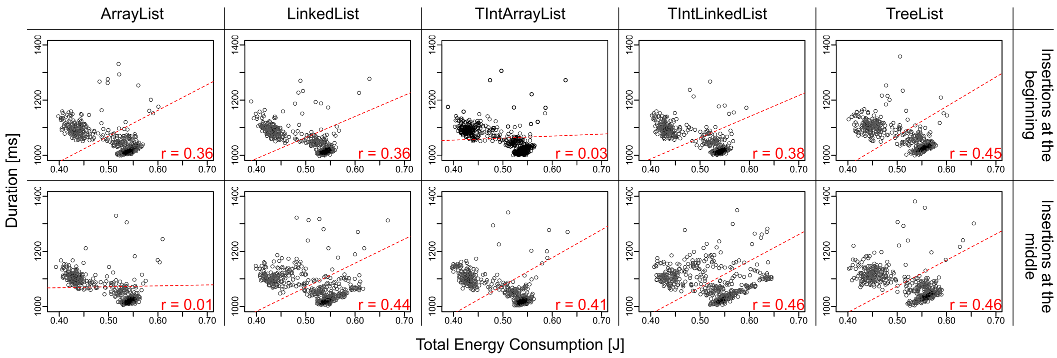


図 6　実験 1：Java Collections クラスについての詳細

Fig. 6　Detailed results of experiment 1: Java Collections classes.

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，(b) 有意差検定の結果

Table 8　(a) Difference of mean energy consumption per unit of time; (b) result of significant difference test for each set.

$O(n \log n)$			$O(n^2)$				
heap	merge	quick	bubble	insertion	selection	shaker	
—	-9.24×10^{-6}	6.84×10^{-5}*	-2.60×10^{-4}*	-3.26×10^{-5}*	-5.39×10^{-5}*	-2.56×10^{-4}*	heap
	—	7.76×10^{-5}*	-2.50×10^{-4}*	-2.34×10^{-5}*	-4.47×10^{-5}*	-2.46×10^{-4}*	merge
		—	-3.28×10^{-4}*	-1.01×10^{-4}*	-1.22×10^{-4}*	-3.24×10^{-4}*	quick
			—	2.27×10^{-4}*	2.06×10^{-4}*	4.23×10^{-6}	bubble
				—	-2.12×10^{-5}*	-2.23×10^{-4}*	insertion
					—	-2.02×10^{-4}*	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．

図 5 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分かる．たとえば，総消費電力量（x 軸）については，$O(n \log n)$ のアルゴリズムではおよそ 0.0〜0.1 [J] であるのに対し，$O(n^2)$ のアルゴリズムではおよそ 0.0〜0.6 [J] である．また，実行時間（y 軸）については，$O(n \log n)$ のアルゴリズムではおよそ 500 [ms] であるのに対し，$O(n^2)$ のアルゴリズムではおよそ 500〜1,400 [ms] である．

さらに，図 5 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，分布のしかたにも違いが見られることが分かる．$O(n \log n)$ のアルゴリズムでは相関が確認できなかったが，これは実行時間がどの入力サイズ n においても 500 [ms] 程度に収まっていることから，n の設定値が小さすぎたことが原因であると推測できる．$O(n^2)$ のアルゴリズムでは，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見られるが，いずれも強い正の相関があることが確認できる．

図 6 から，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分かる．

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

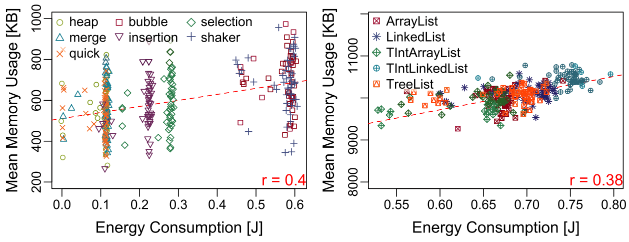
いた値），および (b) 各組の標本について有意差検定を行った結果，をそれぞれ 1 つにまとめたものである．検定により有意差があると認められたものには * 印を付してある．検定手法としては，(1) 母集団が正規分布に従うかどうか不明であり，(2) 標本間には対応があることから，ノンパラメトリックかつ対応ありの有意差検定である，ウィルコクソンの符号付き順位検定（Wilcoxon signed-rank test）[17] を採用した．有意水準 α は 0.01 とした．なお，(1) については，「標本が正規母集団に由来する」という帰無仮説を検定するシャピロ–ウィルク検定（Shapiro-Wilk test）[6] 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，それぞれのアルゴリズムに同一の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，$O(n \log n)$ のアルゴリズムどうし，また $O(n^2)$ のアルゴリズムどうし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分かる．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の調査（仮説 H_3）

実験 2 の結果を図 **7** に示す．x 軸が総消費電力量，y 軸が平均メモリ使用量を表す散布図であり，5.2 節と同様，$n = 5{,}000$ の場合の結果のみを抜粋している．また，図 **8**，図 **9** はその詳細である．

図 7 に着目すると，若干の右上がりの相関が読みとれるが，実験 1 の結果（図 4）に比べて分布のばらつきが非常に大きく，判断が難しい．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は正の



ソーティングアルゴリズム　　Java Collections クラス

図 **7**　実験 2：消費電力量と平均メモリ使用量に関する結果

Fig. 7　Results of experiment 2: relation between energy consumption and mean memory usage.

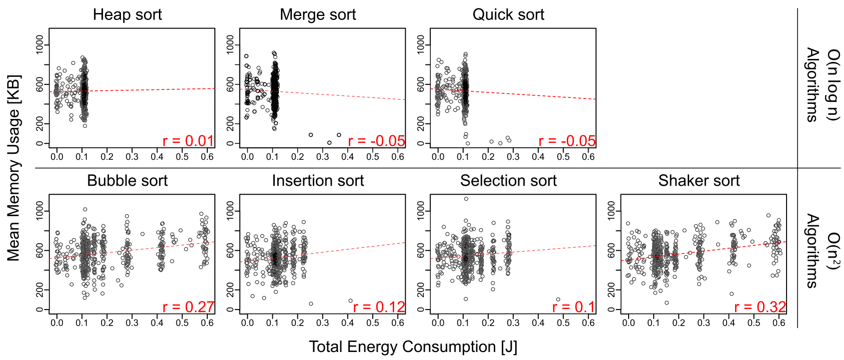


図 **8**　実験 2：ソーティングアルゴリズムについての詳細

Fig. 8　Detailed results of experiment 2: sorting algorothms.

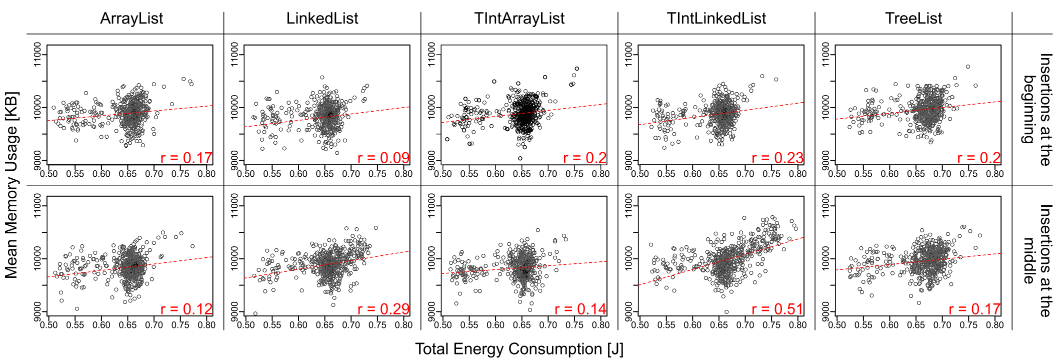


図 **9**　実験 2：Java Collections クラスについての詳細

Fig. 9　Detailed results of experiment 2: Java Collections classes.

相関があるといえる．

図 8 および図 9 から，各実装手段に着目した場合は，n の変化による総消費電力量と平均メモリ使用量の相関はあまり見られないことが分かる．アルゴリズムやクラスによっては，相関がほとんどないものや，負の相関になっているものも存在している．この傾向は，実験 1 と同様であった．

6. 議論

6.1 仮説の検証

仮説 H_1 総消費電力量と実行時間との間には強い相関がある．

結論 仮説は正しい．両者には強い正の相関がある．

実験 1 の結果より，総消費電力量と実行時間との間には強い正の相関が存在し，ソーティングアルゴリズムでは相関係数の値は 0.9 を超えるほどであった．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．実際，2.4 節で述べた Hasan らの結果は，実行時間のみから再現可能である．表 2 にも掲げたように，Hasan らはたとえばリストの中間への挿入について，`TIntLinkedList` が最も総消費電力量が大きく，`ArrayList` および `TIntArrayList` が最も総消費電力量が小さかったと述べている．ここで図 4 の時間方向（y 軸）のみに着目すると，`TIntLinkedList` は確かに最も実行時間が長く，`ArrayList` および `TIntArrayList` は短くなっていることが分かる．

また，より複雑なソフトウェアについても，本結果は適用することができる．どのようなソフトウェアも小さなプログラム構造（メソッドなど）が組み合わさって作られているため，ソフトウェア全体の実行時間を短くしようとするのではなく，このようなソフトウェア中のコード片の実行時間を短くすることにより，全体として総消費電力量も削減することができる．

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分かった．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．

仮説 H_3 総消費電力量と平均メモリ使用量との間には相関がある．

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正の相関が見られることが分かったが，その相関係数は総消費電力量と実行時間のそれよりも小さかった．よって，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．

表 9 標準化偏回帰係数
Table 9 Standardized partial regression coefficient.

説明変数	ソーティングアルゴリズム	Java Collections
実行時間	0.90	1.00
単位時間あたりの消費電力量	0.15	0.21
平均メモリ使用量	0.0001	0.01

6.2 省電力プログラムへの指針

本節では 2 つの実験，および 3 つの仮説の結果に基づいて省電力プログラム実現への指針について議論する．そのために，本稿で扱ったプログラム実行に関する 3 つの指標（実行時間，単位時間あたりの消費電力量，平均メモリ使用量）のどれを優先すれば省電力化につながるかを分析する．具体的には，総消費電力量を目的変数，上記 3 つの指標を説明変数とする重回帰分析を行った．

重回帰分析の結果得られた各指標の標準化偏回帰係数を**表 9** に示す．この指標は各説明変数が目的変数をどの程度説明しているかを表しており，すべての指標を考慮に入れたときの総消費電力量との関連の強さととらえることができる．結果より，実行時間の値がきわめて高く，次いで単位時間あたりの消費電力量の値が高い．なお，分散拡大要因はいずれの指標も 1.0 から 1.4 程度であり，多重共線性は確認されなかった．

これらの分析から，省電力プログラム実現への指針としては，実行時間が短い手法を選ぶことが最も有効であると結論づけることができる．この指針は，本研究で実験対象としたような，CPU のアイドル時間が短いアルゴリズムやプログラムにはとくに適用できると考えられる．さらに，これは電力の計測環境を必要としないという利点を持つ．また，実行時間が同程度かつ電力計測が可能であれば，単位時間あたりの消費電力量が少ないものを選ぶべきであると考えられる．

7. 妥当性への脅威

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．しかし，我々はスーパーコンピュータなどのより大きな計算機や，組み込み系などのより小さな計算機についても，本研究成果と同様の結果が得られると考えている．前者のような，CPU が大量の電力を消費する計算機の場合，単位時間あたりに消費する電力量が大きくなるため，より顕著な傾向が見られると考えている．また後者の場合，単位時間あたりの電力量は減少するため，相関はやや弱くなると考えられるが，同様の傾向は見られると考えている．

また，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・操作としている．これらの題材は，計算量が多く実行時間のほとんどが CPU の処理待ちであるという性質を持つため，同様の性質を満たす科学計算などでは類似する結果が得られると考えられる．一方，その他の計算機リソース（HDD や Wi-Fi など）を用いる題材では結果が大きく変わる可能性があり，本研究成果が適用できない可能性がある．

なお，実験にあたり TARGET 上の不要なプログラムは事前にできるだけ停止しているが，OS を搭載しているためつねに実験対象以外のプログラムも動作しており，計測結果には様々なノイズが含まれている．

8. 関連研究

Pinto らは，StackOverflow に投稿された質問をマイニングし，ソフトウェア開発者がどの程度消費電力量に関心があるのかについて調査している [11]．調査結果の中で，Pinto らは，電力に関する質問は通常の質問よりも平均して閲覧数が 68%多く，また回答数も 2.6 倍多く寄せられていたと述べている．

Zhang らは，電力計測基盤 GreenMiner [8] を用いて収集した，様々なアプリケーションやそれらの各バージョンにおける総消費電力量，CPU 使用率，I/O 状況，メモリ使用量などをデータベースとして公開している [18]．プログラムの消費電力研究における問題点の 1 つとして，Zhang らはこのような電力パフォーマンスに関するデータが容易に入手できない点をあげており，このデータベースは Zhang らのリポジトリから誰でも入手することができる．

Sahin らは，ソースコードのリファクタリングが総消費電力量に与える影響について調査している [14]．Sahin らは，リファクタリングが総消費電力量に影響を与えているだけでなく，リファクタリングによって総消費電力量が減少することもあれば，逆に増加することもあると述べている．

本研究と同様に，総消費電力量と実行時間の関係を調査したものとして，コンパイラオプションの観点から調査を行った研究も存在する [9], [10], [16]．これらの研究では，実行時間を総消費電力量の代わりのメトリクスとして使用することが提案されている．しかし，これらの研究ではコンパイラによる最適化の観点から議論している点に注意されたい．一般に，同等の機能を有するソフトウェアを複数実現する方法には様々なアプローチがあり，これらの研究における「ソースコードは変更せずにコンパイラの最適化方法を変更する」という手法は，最も低レベルからのアプローチであるといえる．一方，本研究では，コンパイラの最適化ではなく実装アルゴリズムを変更してソースコードそのものを変更しているという点で，これらの研究と大きく異なっている．また，これらの研究成果と本研究成果は排他的なものではないため，双方の手法を併用することが可能である．

本研究と動機が類似するものとして，Rashid らによる研究 [13] がある．Rashid らは，異なる 4 つのソーティングアルゴリズムを異なる 3 つのプログラミング言語でそれぞれ実装し，その電力効率を比較分析している．実験の結果から，Rashid らはアルゴリズムや言語により消費電力量が大きく異なることを指摘している．また同時に，電力消費の大部分が計算複雑度のような時間性能によって決定されると述べており，これは本研究の結論とよく似ている．一方，本研究では Rashid らが調査したソーティングアルゴリズムに加え，Java Collections クラスの違いによる消費電力量への影響も調査している．これにより実験対象となるプログラムの数を多く確保でき，外的妥当性が高いといえる．さらに，本研究では計測に必要な環境やツールを開発・公開しており，第三者による追試が比較的容易である．

9. おわりに

本研究では，プログラム実行時の実行時間と総消費電力量との相関関係およびその他計算機リソースと総消費電力量との相関関係の有無を明らかにするため，3 つの仮説をたてて調査を行った．調査の結果，プログラムの実行時間，単位時間あたりの消費電力量の順に，総消費電力量に与える影響が大きいことが分かった．この研究成果より，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，実行時間が短い実装手段を選択すればよいといえる．

今後の課題として，平均メモリ使用量以外の他の計算機リソース（ディスク I/O や Wi-Fi 通信など）について，総消費電力量との相関を同様に調べることがあげられる．また，本研究で得られた成果がイベント駆動式のアプリケーション（スマートフォンにおけるメモ帳アプリなど）についても適用されるのか，より実践的な実験を行うことが考えられる．

謝辞 本研究は，日本学術振興会科学研究費補助金基盤研究（S）（課題番号：JP25220003），および文部科学省研究費補助金若手研究（B）（課題番号：JP26730155）の助

成を得て行われた．

参考文献

[1] 浅野哲夫，和田幸一，増澤利光：アルゴリズム論，オーム社 (2003).

[2] Bunse, C., Höpfner, H., Mansour, E. and Roychoudhury, S.: Exploring the Energy Consumption of Data Sorting Algorithms in Embedded and Mobile Environments, *Proc. International Conference on Mobile Data Management*, pp.600–607 (2009).

[3] Bunse, C., Höpfner, H., Roychoudhury, S. and Mansour, E.: Choosing the "Best" Sorting Algorithm for Optimal Energy Consumption, *Proc. International Conference on Software and Data Technologies*, Vol.2, pp.199–206 (2009).

[4] Dolan, E.D. and Moré, J.J.: Benchmarking optimization software with performance profiles, *Mathematical programming*, Vol.91, No.2, pp.201–213 (2002).

[5] Faulkner, R. and Gomes, R.: The Process File System and Process Model in UNIX System V, *USENIX Winter*, pp.243–252, USENIX (1991).

[6] Ghasemi, A., Zahediasl, S., et al.: Normality tests for statistical analysis: A guide for non-statisticians, *International journal of endocrinology and metabolism*, Vol.10, No.2, pp.486–489 (2012).

[7] Hasan, S., King, Z., Hafiz, M., Sayagh, M., Adams, B. and Hindle, A.: Energy Profiles of Java Collections Classes, *Proc. International Conference on Software Engineering*, pp.225–236 (2016).

[8] Hindle, A., Wilson, A., Rasmussen, K., Barlow, E.J., Campbell, J.C. and Romansky, S: Greenminer: A hardware based mining software repositories software energy consumption framework, *Proc. Working Conference on Mining Software Repositories*, pp.12–21 (2014).

[9] Ibrahim, M.E., Rupp, M. and Habib, S.-D.: Compiler-based optimizations impact on embedded software power consumption, *Proc. Joint IEEE North-East Workshop on Circuits and Systems and TAISA Conference*, pp.1–4 (2009).

[10] Pallister, J., Hollis, S.J. and Bennett, J.: Identifying compiler options to minimize energy consumption for embedded platforms, *The Computer Journal*, Vol.58, No.1, pp.95–109 (2013).

[11] Pinto, G., Castor, F. and Liu, Y.D.: Mining questions about software energy consumption, *Proc. Working Conference on Mining Software Repositories*, pp.22–31 (2014).

[12] Platt, J.: Sequential minimal optimization: A fast algorithm for training support vector machines, Technical Report (1998).

[13] Rashid, M., Ardito, L. and Torchiano, M.: Energy Consumption Analysis of Algorithms Implementations, *Proc. ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp.1–4 (2015).

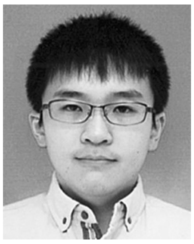
[14] Sahin, C., Pollock, L. and Clause, J.: How do code refactorings affect energy usage?, *Proc. ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, p.36 (2014).

[15] Sedgewick, R.：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).

[16] Seng, J.S. and Tullsen, D.M.: The effect of compiler optimizations on Pentium 4 power consumption, *Proc. Interaction Between Compilers and Computer Architectures*, pp.51–56 (2003).

[17] Teetor, P.：R クックブック，オライリー・ジャパン (2011).

[18] Zhang, C. and Hindle, A.: A green miner's dataset: Mining the impact of software change on energy consumption, *Proc. Working Conference on Mining Software Repositories*, pp.400–403 (2014).



松尾 裕幸

平成 29 年大阪大学基礎工学部情報科学科卒業．同年より同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在学中．

松本 真佑 （正会員）

平成 22 年奈良先端科学技術大学院大学博士後期課程修了．同年神戸大学大学院システム情報学研究科特命助教．平成 28 年大阪大学大学院情報科学研究科助教．博士（工学）．エンピリカルソフトウェア工学の研究に従事．

楠本 真二 （正会員）

昭和 63 年大阪大学基礎工学部情報工学科卒業．平成 3 年同大学大学院博士課程中退．同年同大学基礎工学部助手．平成 8 年同講師．平成 11 年同助教授．平成 14 年同大学大学院情報科学研究科助教授．平成 17 年同教授．博士（工学）．ソフトウェアの生産性や品質の定量的評価，プロジェクト管理に関する研究に従事．IEICE，JSSST，IEEE，JFPUG，PM 学会，SEA，各会員．