Vol.2017-SE-195 No.16
2017/3/12

実装手段の異なるプログラムの実行時間と消費電力量に関する調査

松尾 裕幸[†1,a)]　柗本 真佑[†2,b)]　楠本 真二[†2,c)]

概要：近年，ソフトウェアの電力消費の削減のため，API やアルゴリズム等の実装手段の違いによる電力消費の比較研究が盛んに行われている．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，異なるソートアルゴリズム，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，相関係数が 0.9 を超える非常に強い正の相関が存在することが示された．また，総消費電力量と平均メモリ使用量の間にも弱い相関があることがわかった．これらの研究成果から，開発者がプログラムの総消費電力量を削減したいとき，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．

キーワード：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

1. はじめに

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，メモリ効率のよいインサーションソートが最も電力効率がよ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，その総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を開発する際の一つの指針として活用可能である．

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，あらゆる計算機は駆動しているだけで一定の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（メモリやディスク等）の使用量に差はあるものの，そのリソース使用の差異は，実行時間以上の支配的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], [9] 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のであれば，時間という極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の関係を調査する．調査対象における 3 つの具体的な仮説は以下の通りである．

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上で，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

†1 現在，大阪大学基礎工学部情報科学科
†2 現在，大阪大学大学院情報科学研究科
a) h-matsuo@ist.osaka-u.ac.jp
b) shinsuke@ist.osaka-u.ac.jp
c) kusumoto@ist.osaka-u.ac.jp

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一つとして，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，先に挙げた 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わかった．また，平均メモリ使用量と総消費電力量の相関は，実行時間と総消費電力量のそれに比べると弱いことが示された．これらの研究成果から，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といえる．

2. 準備

2.1 電気に関する用語の定義

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力を $I(t)$，瞬時電圧を $E(t)$ とすると，その瞬時電力は $P(t) = E(t) \cdot I(t)$ と計算できる．一般に，計算機上であるプログラムを実行したときの消費電力量とは，t_{begin}，t_{end} をそれぞれプログラムが起動，終了した時刻として，次式で計算される値のことを指す．

$$\int_{t_{\mathrm{begin}}}^{t_{\mathrm{end}}} P(t)\,dt \tag{1}$$

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 8 つのソーティングアルゴリズムとその概要を示す [1], [10]．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．

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	—

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 [2], [3]．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．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
- メモリ効率のよ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よいこと

などを指摘している．

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 [7]．`List` の実装としては，Java Collections Framework（JCF）に含まれる `ArrayList` および `LinkedList`，Trove[*1] に含まれる `TIntArrayList` および `TIntLinkedList`，そして Apache Commons Collections[*2]（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[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．

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

[*1] `http://trove.starlight-systems.com/`
[*2] `https://commons.apache.org/proper/commons-collections/`

表 2 List インター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

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

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

前述の通り，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，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，実行時間を考慮した結果については定量的に述べられていない．

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

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，プログラムの総消費電力量の〈最適化〉について考える際には，実行時間だけではなく実際の総消費電力量も考慮して議論を行う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相関がある．

2.3.1 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，平均消費電力量に差があらわ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

本研究では，次節および次々節で述べる 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[*3] を設計・開発した．設計にあたっては，

[*3] 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.

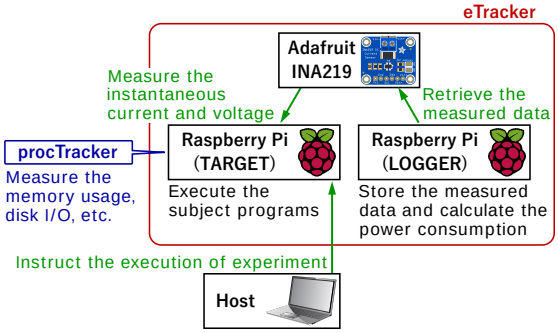

図 1　実験環境の概要

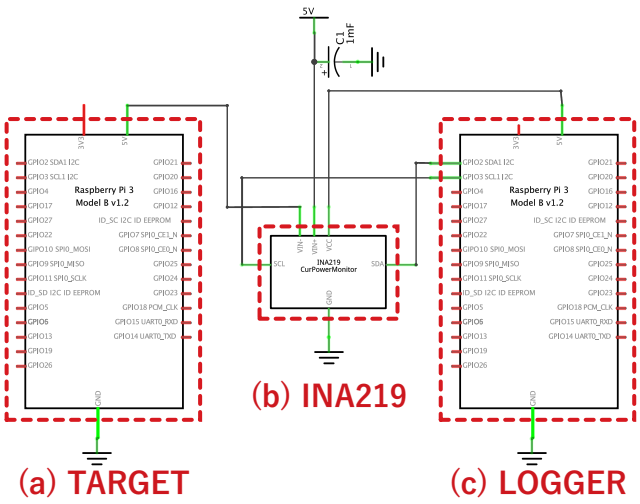

図 2　eTracker の回路図

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れられる：

(a) Raspberry Pi[*4] 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処理・保管する 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．

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[*5] 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，I^{2}C[*6] 通信で LOGGER に送られる．

本節の終わりに，実際に制作した eTracker の全体図を**図 3** に示す．

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[*7] 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している．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．

*4　Linux が動作する，小型のコンピュータ．
*5　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．
*6　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．

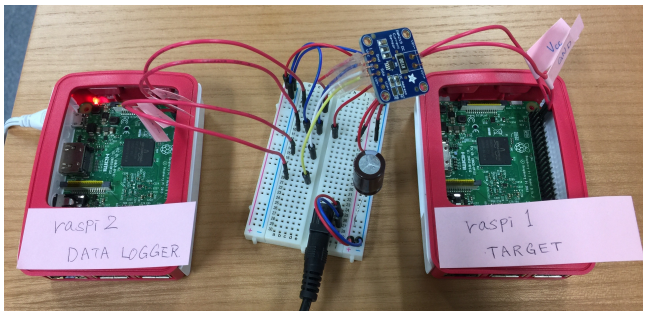

図 3　eTracker の全体図

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．

表 3 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，Hasan らの研究 [7] でも用いられている 5 種類である．

表 4 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 5** は実験で用いた Java 環境のバージョン，**表 6** は用いたサードパー

*7　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`.

表 3 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
入力サイズ n	1, 250, 500, 750, ..., 5000

表 4 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 5 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 6 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

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に関する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 $n = 5000$ 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，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の調査（仮説 H_1, H_2）

実験 1 の結果を，**図 4**，**図 5** に示す．なお，図 4 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 4 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 4 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 5 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 7 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）[11]

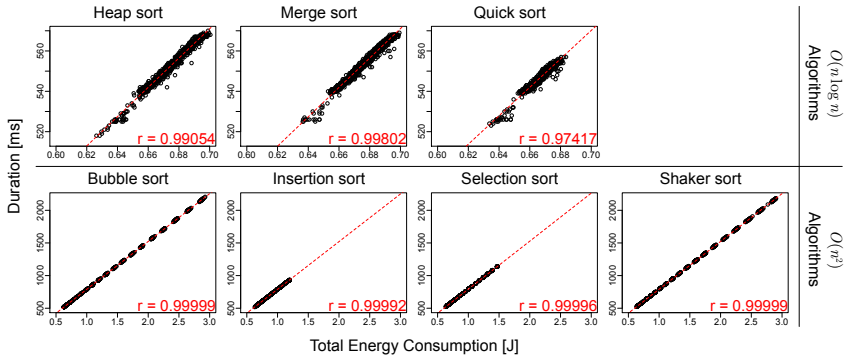

図 4 実験 1：ソーティングアルゴリズムについての結果

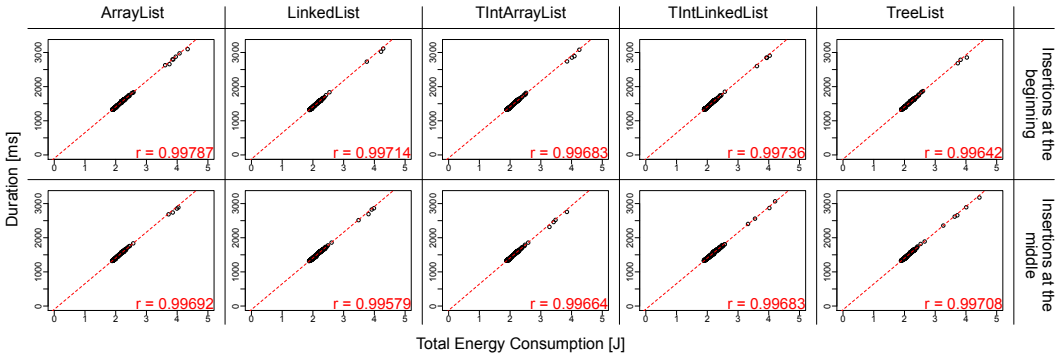

図 5 実験 1：Java Collections クラスについての結果

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 7 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の調査（仮説 H_3）

実験 2 の結果を，**図 6**，**図 7** に示す．実験 1 と同様，図 6 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 6 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 7 では，図 6 に比べるとばらつきが少ないが，図の形状からは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 6 から，実行時間（図 4）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向の分布が多くみえる．

実験 1 と同様，図 7 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．

6. 議論

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．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 の結果より，ソーティングアルゴリズムおよび

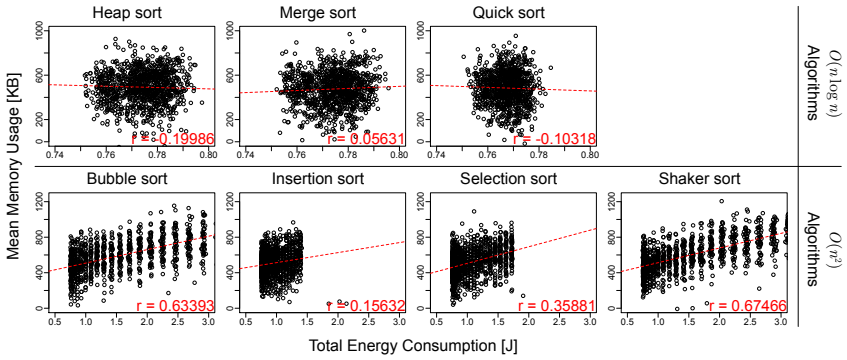

図 6 実験 2：ソーティングアルゴリズムについての結果

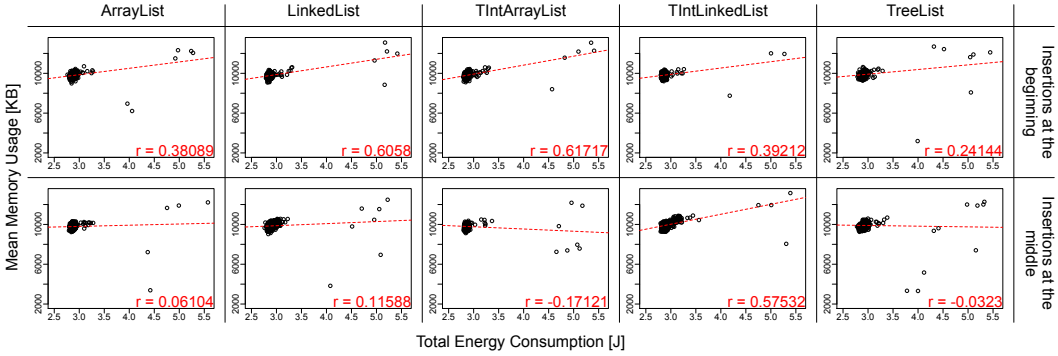

図 7 実験 2：Java Collections クラスについての結果

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，単位時間あたりの総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．

仮説 H_1 が真であることより，総消費電力量の〈削減〉のみを考える場合は，実行時間の短縮を考えればよいことが示された．しかし，仮説 H_2 が真であることより，総消費電力量の〈最適化〉を考える場合は，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ことを示している．

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，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 プログラムを起動するとき，JVM は自身の上で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，最も優先して考慮するべきなの

表 7 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

$O(n \log n)$			$O(n^2)$				
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．

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，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 を搭載している性質上，常に実験対象以外のプログラムも電力を消費しており，計測結果には様々なノイズが含まれている．

本研究では，実験対象を表 3 に示す 7 つのソーティングアルゴリズム，および表 4 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. 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や，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，実行時間と総消費電力量，平均メモリ使用量と総消費電力量の双方に相関がみられた．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について，総消費電力量との相関を同様に調べることが挙げられる．また，本研究で得られた成果が，実際に広く用いられているライブラリやアプリケーション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, *Proceedings of the 10th International Conference on Mobile Data Management*, IEEE, pp. 600–607 (2009).
[3] Bunse, C., Höpfner, H., Roychoudhury, S. and Mansour, E.: Choosing the "Best" Sorting Algorithm for Optimal Energy Consumption, *Proceedings of the 4th 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*, USENIX, pp. 243–252 (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, *Proceedings of the 38th International Conference on Software Engineering*, ACM, 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, *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, pp. 12–21 (2014).
[9] Platt, J.: Sequential minimal optimization: A fast algorithm for training support vector machines (1998).
[10] 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). 野下 浩平，星 守，佐藤 創，田口 東 訳．原著：Algorithms in C, Parts 1-4.
[11] Teetor, P.: 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.