Vol.2016-HPC-156 No.8
2016/9/16

# イベントベースサンプリングによるソフトウェア消費電力の計測手法の評価

小野美由紀[†1]　山本昌生[†1]　中島耕太[†1]

**概要**：HPC システムやデータセンターの大規模化と高性能化が進んでおり，これに伴う消費電力の増大が深刻な課題となっている．この問題を解決するために，ハードウェアに関する省電力化技術は進んでいる．例えば，マイクロアーキテクチャ，冷却機構，電源ユニットといった各部分の省電力化が進んでいる．一方でこれを利用するソフトウェアの観点での省電力化は進んでいない．ソフトウェアの観点で電力を論じるためには，まず，プログラムのホットスポットを明らかにする機構が必要である．そこで，我々は，関数単位での電力ホットスポットを明らかにする消費電力プロファイル分析について研究を行っている．これまでに，電力データ採取のための専用計測器やソフトウェアの改変を不要とする電力データ採取方法として，電力消費に関係する CPU の性能イベントをベースとしたサンプリングを行うことにより，従来のタイムベースサンプリングよりも精度の高い電力サンプリングを可能とする手法を提案した．本稿では，提案手法を評価するため，電力消費に関係する 4 つの性能イベントをベースとしたサンプリングを同時に行う複数イベントベースサンプリングを実装し，採取したサンプルデータの電力値を一定消費電力量毎に集計することにより，疑似的電力サンプリングを試作した．特に，短時間しか実行されないアプリケーションの消費電力の計測を行った場合において，サンプリング間隔が大きい場合にはイベントベースサンプリングのほうがタイムベースサンプリングよりも正確に計測できることを示した．

## 1. はじめに

ICT システムにおける省電力化の要求は，コストや資源枯渇の面だけでなく地球環境問題の観点から高まっている．これに伴い，計算機でも，CPU のマルチコア化以降，電力が設計要件や性能指標の一部として重要視されるようになっている．

HPC システムやデータセンターの大規模化と高性能化が進んでおり，これに伴う消費電力の増大が深刻な課題となっている．例えば，スーパーコンピュータシステムに対する性能ベンチマーク Linpack で 93PFlop/s を達成した Sunway TaihuLight では約 15.48MW という莫大な電力を必要とする [1]. したがって, 省電力化が非常に重要である．また，HPC システムの電力効率のランキングである Green 500 [2] によると，2016 年 6 月時点では，最高の電力効率は 6.674GFlops/W であり，この電力効率で 1EFlops を達成しようとすると約 150MW が必要となる．このような膨大な消費電力は現実的ではなく，今後計算速度を向上させていくためにも，省電力化は非常に重要である．

この問題を解決するために，ハードウェアに関する省電力化技術は進んでいる. 例えば, マイクロアーキテクチャ，冷却機構，電源ユニットといった各部分の省電力化が進んでいる．一方でこれを利用するソフトウェアの観点での省電力化は進んでいない．ソフトウェアの観点で電力を論じるためには，まず，プログラムのホットスポットを明らかにする機構が必要である．そこで，我々はプログラムの電力最適化を行うことを目標とし，関数単位での電力ホットスポットを明らかにする消費電力プロファイル分析について研究を行っている．これまでに，電力データ採取のための専用計測器やソフトウェアの改変を不要とする電力データ採取方法として，タイムベースサンプリングと同時に CPU が備える消費電力監視機能 RAPL の消費電力値を採取する方法を提案した [9]．しかし，タイムベースサンプリングでは時間がかかる箇所がサンプリングされ，必ずしも電力消費が多い箇所をサンプリングできるとは限らない．そこで, 電力消費に関係する CPU の性能イベントをベースとしたサンプリングを行うことにより，従来のタイムベースサンプリングよりも精度の高い電力サンプリングを可能とする手法を提案した [10]．本稿では，提案手法の有効性を評価するため，電力消費に関係する 4 つの性能イベントをベースとしたサンプリングを同時に行う複数イベントベースサンプリングを実装し，採取したサンプルデータの電力値を一定消費電力量毎に集計することにより，疑似的電力サンプリングを試作した．特に，本報告ではあらたに、短時間しか実行されないアプリケーションの消費電力の計測を行った場合において，サンプリング間隔が大きい場合にはイベントベースサンプリングのほうがタイムベースサンプリングよりも正確に計測できることを示した．

本稿では，2 章でタイムベースサンプリングと同時に CPU が備える消費電力監視機能 RAPL(Running Average Power Limit)の消費電力値を採取する従来手法とその課題について述べる．3 章では電力消費に関係する CPU の性能イベントをベースとしたサンプリングによる電力計測の提案手法について述べ，4 章では短時間しか実行されないアプリケーションの消費電力の計測に関してイベントベースとタイムベースを比較する．5 章では関連研究，6 章でまとめと今後について述べる．

†1 (株) 富士通研究所
Fujitsu Laboratories Ltd.

## 2. 従来手法の課題

### 2.1 RAPL

RAPL は，ソフトウェアから消費電力を監視制御機能するための機能で，Sandy Bridge 以降の Intel CPU に搭載されている．本機能により，CPU やメモリの消費電力を監視し，予め設定した電力に達した場合に消費電力を制限することができる [3] ．

図 1 のように，RAPL では CPU 全体（青い部分）や CPU 内のコア全体（Core：オレンジ色の部分）の消費電力を監視できる．さらに，サーバ機ではメモリ（DRAM：緑色の部分），クライアント機では，グラフィクス（赤い部分）の消費電力が監視対象となる[4]．これらの監視単位毎に，電力制御や消費電力記録などのためのレジスタが用意されており，消費電力値（Joule：以降 J と表記）は約 1ms 毎に更新される．

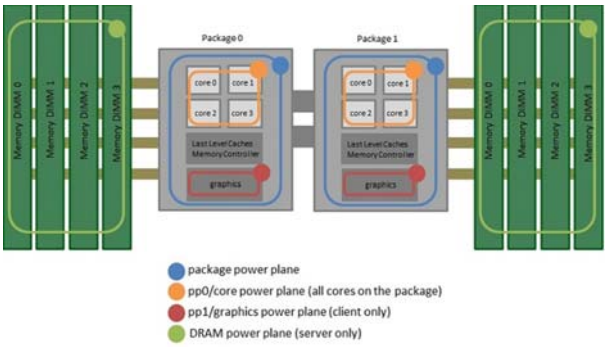


図 1　RAPL の監視対象（出典[4]）

電力プロファイルの実現において，RAPL には十分な精度と利便性があると考えている．RAPL で得られる消費電力は計算値である[5]が，RAPL と外部電力計の消費電力傾向は非常に良く一致するという報告がある[6]．これは，電力プロファイルで消費電力のホットスポットを発見するために，消費電力の傾向を捉える用途には十分な精度である．また，専用ハードウェアなしに，CPU に組み込まれた機能を利用して，電力情報を採取可能であるという利点もある．

RAPL はあくまで積算電力のカウンタであり，これと走行するプログラムの電力とを関連付けるための機能はない．したがって，たとえば測定対象のプログラム内部に RAPL によって計測される電力を記録する仕組みを埋め込む機構や，定期的に RAPL の電力を計測し，その瞬間に動作していたプログラムと関連付ける機構のような仕組みが必要である．また，RAPL で採取できるのは CPU 全体の消費電力であり，CPU 内の各コアが消費した電力はわからない．CPU 内の各コア上で別々のプログラムが走行する場合，個々のプログラムの消費電力を把握することができない．

### 2.2 タイムベースサンプリングによる電力計測

RAPL では関数レベルの電力プロファイルの実現が難しいという課題に対し，図 2 のような電力サンプリング機構による電力データ採取と，CPU 単位の電力値を CPU 内の各コアに分配する手法を提案した [9].

RAPL では積算電力ベースのサンプリングが実現できないため，プロファイラで一定間隔毎にタイムベースサンプリングを行う際に，RAPL 機能を利用して消費電力量を採取することにより電力サンプリングを実現する．このとき，コアの性能情報も採取する．この性能情報をもとにコアに消費電力を分配する．さらに，各サンプルデータの動作プログラム情報を利用して，コアに分配した消費電力を関数に分配する．各サンプルデータには，通常のプロファイラと同様に動作プロセスや関数を特定する情報が採取される．

CPU 全体の消費電力を各コアに分配することにより，各コアで動作したアプリケーションの消費電力が把握可能となる．

なお，本プロファイラでは時刻に相当するイベント CPU_CYCLES を使用して一定時間間隔のサンプリングを実現している．従って，イベントベースサンプリングの一種であるが，他のイベントベースサンプリングと区別するために，タイムベースサンプリングと記述する．

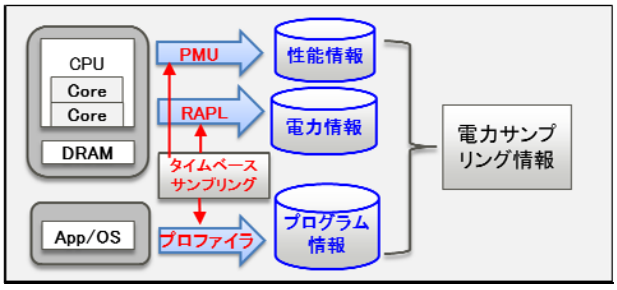


図 2　タイムベースの電力サンプリング機構

### 2.3 タイムベースサンプリングの課題

タイムベースサンプリングでは一定時間間隔毎にサンプリングを行うため，時間がかかる部分が多くサンプリングされる傾向がある．時間はかからないが，電力は多く消費するような部分があった場合，少なくサンプリングされることになる．このような部分は実際よりも消費電力が少なく見えてしまうという問題が発生する．

図 3 を用いて，タイムベースサンプリングと積算電力ベースサンプリングの違いを説明する．図 3 の 2 つのグラフは横軸が経過時間，縦軸が消費電力量であり，各時間にどの関数がどのくらい電力を消費したかを表すものである．ここでは関数 A（水色）と B（ピンク色）がそれぞれ 12J ずつ電力を消費している．この状況で，タイムベース（左のグラフ）と積算電力ベース（右のグラフ）でどうサンプリングされるかをみる．タイムベースでは実行時間の長い

関数 A が 3 回, 実行時間の短い関数 B が 1 回サンプリングされ, 消費電力量は関数 A が合計 18J, 関数 B は 6J である．一方，積算電力ベースでは 6J 毎にサンプリングを行い，関数 A と B がそれぞれ 2 回サンプリングされ, 消費電力量も 12J ずつとなる．この例では，積算電力ベースでは正しく消費電力をサンプリングできるが，タイムベースでは正しく消費電力をサンプリングできていない．

このように，タイムベースサンプリングによる電力計測では，サンプリングされる電力量は一定とは限らず，必ずしも電力消費が多い箇所をサンプリングできるとは限らない．そのため，より正確に電力を計測できるサンプリング方式が必要である．

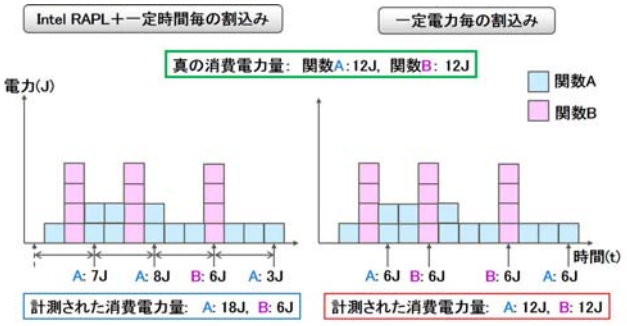


図 3　サンプリングの比較

## 3. 提案手法

タイムベースサンプリングでは正確な電力サンプリングを行えない場合があるという課題に対して, 時間ではなく，電力消費に関係する CPU の性能イベントをベースとしたサンプリングを行う手法を提案した [10]．これにより，従来のタイムベースサンプリングよりも精度の高い電力サンプリングを可能とすることを目指す．

サンプリングのベースとするイベントとしては，CPU が備える PMU(Performance Monitoring Unit)という性能監視機構で測定監視できる CPU の性能(Performance Monitoring Counter: PMC)イベントを利用する．PMU のカウンタオーバーフロー割込みをサンプリング契機に利用して行うサンプリングをイベントベースサンプリングと呼ぶ．性能イベントとしては実行命令数やキャッシュミスなどがあり，これらの性能に関係するイベントの発生回数をカウントできる．しかし，消費電力を直接監視しているイベントは提供されていない．そこで，電力消費と相関関係のある 4 つの性能イベントをベースとしたサンプリングを同時に行う複数イベントベースサンプリングを実装し，採取したサンプルデータの電力値を一定消費電力量毎に集計することにより，疑似的電力サンプリングを行う．サンプリングは，3.1 で述べる消費電力モデルで使用するイベントをベースとする．

### 3.1 消費電力モデル

サンプリングのベースとなるイベントを決定するために, 消費電力量と相関の高いイベントや CPU 情報をベースに，消費電力を表すモデルを作成した．まず，いくつかのアプリケーション実行時に消費電力量，PMC イベント値，CPU 情報を採取する．採取したデータから各消費電力値を目的変数，消費電力と相関が高い PMC イベント値やコア情報を説明変数として，重回帰分析により消費電力モデルを作成する．一度に採取可能な PMC イベント数には制限があり，例えば Sandy Bridge や Haswell では 4 つである．従って，電力測定を 1 度で済ませるためには，モデルに使用するイベント数を 4 つ以下に絞り込むことが必要である．

表 1 の実験環境で，アイドル状態，CPU 負荷をかけた状態やメモリ負荷をかけた状態における消費電力と PMC イベント値，CPU 情報を採取した．CPU 負荷をかけるアプリケーションは stress コマンドと AVX プログラム，メモリ負荷をかけるアプリケーションは stream ベンチマークを使用した．stress コマンドでは，CPU 負荷をかける worker を 1 つ動作させた．AVX プログラムは AVX 命令用のパイプラインが埋まるようにアセンブリ記述した自作のプログラムである．stream ベンチマークはアプリケーションで使用するメモリ量を変え，L1 キャッシュサイズ内，L2 キャッシュサイズ内，L3 キャッシュサイズ内，メモリを使用する 4 つのパターンで実行した. また，1 ソケット 8 コア構成で，1 コアから 8 コアまでアプリケーションを実行するコア数を 1 つずつ増やして 8 パターンを採取した．なお，今回は 2 ソケットのうちの 1 ソケットのみを使用し，残りの 1 ソケットはほぼアイドル状態だった．また，周波数は固定とした．

表 1　実験環境

| | |
|---|---|
| CPU | Sandy Bridge (XeonE5) |
| CPU 周波数 | 2.90GHz |
| CPU ソケット数 | 2 |
| コア数／ソケット | 8 |
| キャッシュサイズ | L1d 32K, L1i 32K, L2 256K, L3(LLC) 20MB |

これらのデータをベースに重回帰分析を行い，消費電力モデルを作成した. 消費電力モデルの重決定係数 R2 は 0.99 である．重決定係数は，目的変数の変動のうち説明変数によって説明される割合を表すものであり, 0≦R2≦1 の値を取り，1 に近いほどよい．従って，作成したモデルは消費電力をうまく表せていると言える．

今回は Core の消費電力のみを対象とするため, 上記に今回採用した Core のモデルを示す．Core のモデルは，CPU の使用状況を表す C0 と実行命令数 INST_RETIRED.ANY_P

（INST），リソースに関係したストールサイクル数 RESOURCE_STALLS.RS（RESOURCE），L2 キャッシュアクセスに関するイベント L2_TRANS.ALL_PF（L2）を使用することにした．

Core の消費電力[W]

$12.41+6.288\times C0$
$+2.06\times10^{-11}\times$INST_RETIRED.ANY_P
$+3.161\times10^{-10}\times$RESOURCE_STALLS.RS
$+6.058\times10^{-10}\times$L2_TRANS.ALL_PF

### 3.2 イベントのサンプリング間隔

次に，消費電力モデルに使用するイベントに対するサンプリング間隔を決定する．

今回は，Core の消費電力を対象とし，積算電力サンプリングする電力量 P にモデル式の各イベント $E_i$ の係数 Ci の逆数を掛けて求めた回数をサンプリング間隔 $R_i$ とする．

$$R_i = P / C_i$$

関数プロファイルを目的とした電力サンプリングのために，1 J 毎にサンプリングすることを想定する．表 2 に各イベントのサンプリング間隔を示す．例えば，INST イベントでは，1 秒間で約 4,854,000,000 回に 1 回サンプリングすることになる．関数プロファイルを考えると 1 秒間では荒いので，10ms 間相当とし，48,540,000 回に 1 回サンプリングする．なお，モデル式の C0 はレジスタから計算することを想定していたが，Halt でない状態でのサイクル数 CPU_CLK_UNHALTED.THREAD_P（CLK）から計算することができるため，イベントとしてこれを使用する．モデル式の C0 は各コアの値の合計から求めているため，係数をコア数で割った値を使用する．なお，今回使用した CPU の周波数から CLK イベントは約 12.7ms 間隔にサンプリングされることになる．従って，10ms 間隔のタイムベースサンプリングよりも荒いサンプリングとなる．

表 2　各イベントのサンプリング間隔

| イベント | 回数／秒 | 回数／10ms |
|---|---|---|
| INST | 4,854,368,932 | 48,540,000 |
| RESOURCE | 316,355,584 | 3,160,000 |
| L2 | 165,070,981 | 1,650,000 |
| CLK | 3,689,567,430 | 36,900,000 |

### 3.3 複数イベントによるサンプリング

今回使用する消費電力モデルは 4 つのイベントの組み合わせで消費電力を表すものであり，サンプリングも複数イベントで行うことが理想的である．そこで，複数のイベントで同時にサンプリングを行う複数イベントベースサンプリングを試作した．なお，複数イベントでサンプリングを行ったとしても，一定電力量毎に割込みを発生させられるわけではない．そのため，電力サンプリングを実現するために，採取したサンプルデータを集計する．

サンプルデータとして， CPU（コア）識別子，動作プロセス，実行アドレス，時刻情報，RAPL 値，各イベントの値を採取する．動作プロセスと実行アドレスから動作した関数，時刻情報から測定開始からの経過時間を求めることができる．CPU 毎に，測定開始から経過時間順にサンプルデータの電力値を見ていき，集計単位の消費電力量になるサンプルデータに集計単位の消費電力量をカウントする．サンプルデータの電力量に端数が生じた場合には次のサンプルデータの値として繰越す．このようにして，集計により疑似的に電力ベースサンプリングを実現する．例えば，図 3 のデータで 6J 毎にサンプリングする場合，1 つ目と 2 つ目のデータの電力値 1J と 4J を繰り越し，3 つのデータで 6J となり，関数 A に 6J を加算する．3 つのデータの電力値の残り 1J を繰り越す．5 つ目のデータで 6J となり，関数 B に 6J を加算する．

## 4. 評価

### 4.1 電力プロファイル

計測したサンプルデータの電力値をある一定消費電力量毎にカウントしてプロセスや関数毎に集計し，電力プロファイルを作成することにより，電力を多く消費する部分を発見できる．

```
Socket1 Total_power=222.5 (J)
 CPUID:8 POWER=28.0
 CPUID:9 POWER=28.5
 CPUID:10 POWER=27.5
 CPUID:11 POWER=28.5
 CPUID:12 POWER=27.5
 CPUID:13 POWER=27.5
 CPUID:14 POWER=27.5
 CPUID:15 POWER=27.5

Function Information
Power CPU8 CPU9 CPU10 ... CPU15 Funcname
 97.0  13.0  12.0  10.5  ...  12.0  libc-2.12.so::random_r
 43.0   4.5   5.0   9.0  ...   4.0  libc-2.12.so::random
 37.5   5.0   1.5   5.0  ...   4.5  stress::hogcpu
 23.0   2.5   3.0   2.5  ...   3.0  cpu_test::main.omp_fn.0
 16.0   2.0   1.5   2.5  ...   3.5  libc-2.12.so::rand
  5.0   0.0   0.5   1.5  ...   0.5  stress::.plt
  1.0   1.0   0.0   0.0  ...   0.0  update_curr

Process Information
31786 : "stress"  TOTAL =8.0(J)
  Funcname                Power
  libc-2.12.so::random_r  4.0
  libc-2.12.so::rand      2.0
  stress::hogcpu          1.5
  libc-2.12.so::random    0.5
                :
```

図 4　電力プロファイルの出力例

図4にCore消費電力プロファイルの出力イメージを示す．ここでは，”CPU”は論理コア（スレッド）を表す．最初にソケット内の各コアのCore消費電力量を集計した結果，次にソケット内の関数ランキング，最後にプロセス情報を出力している．プロセス情報としては，プロセス毎に総消費電力量とプロセス内で実行された関数の消費電力量の内訳を提示する．このような情報を出力することによって，電力を消費したプロセスや関数，さらに1つの関数が複数のプロセスから実行された場合，どのプロセスから実行された場合の消費電力が多いかを調べることが可能となる．

### 4.2 短時間実行アプリの消費電力計測

アプリケーションによる消費電力量は常に一定とは限らない．一時的に大量の電力を消費し，電力制限を超えてしまうことも考えられる．このような瞬間的に電力を消費するものを計測できることも必要である．

そこで，実際に複数イベントベースサンプリングを行い，前節で示したような電力プロファイルを作成することにより，短時間しか実行されないアプリケーションの消費電力を正しく計測できるかを検証する．具体的には，単位時間の消費電力量が小さめのアプリケーションを長めに，単位時間の消費電力量が大きめのアプリケーションを短めに実行する．アプリケーション実行の前後でRAPLの値を採取し，その差分をアプリケーションの消費電力の実測値とする．この値とサンプルデータから集計した電力値を比較する．単位時間の消費電力量が小さめのアプリケーションとしてCPU負荷をかけるstressコマンドを1秒，単位時間の消費電力量が大きめのアプリケーションとしてAVXプログラムを0.1秒実行する．これを図5に示すように，stressコマンド，AVXプログラム，stressコマンド，AVXプログラム，stressコマンドの順に続けて実行する．なお，プロファイラの測定がstressコマンドの初回実行の途中からとなっていたため，stressコマンドは2回目と3回目のみを集計の対象とする．

stressコマンドはtasksetコマンドでCPU（コア）を固定して8並列で実行し，AVXプログラムはOpenMPにより8コアを使用して8並列で実行する．2つのアプリケーションは全コアでほぼ同じ動きをするため，各サンプルデータのRAPL値の差分を求め，この値の8分の1の値をJouleに換算したものをCore消費電力量とする．なお，今回の実験環境では，1ソケット8コアで実行した場合の消費電力はstressコマンド約70J／秒，AVXプログラム約100J／秒である．


図5　評価パターン

4つのイベントでのイベントベースサンプリングとタイムベースサンプリングを行い，結果を比較する．タイムベースのサンプリング間隔は，100ms，50ms，10msの3パターンで採取し，各サンプルデータのRAPL値を使用する．イベントベースはタイムベースのサンプリング間隔に相当する回数を設定した．具体的には3.2で述べた10ms間隔タイムベースサンプリング相当の回数を基準とし，50msでは5倍，100msでは10倍の回数とした．採取したサンプルデータに対して100ms相当は0.5J，50ms相当は0.3 J，10ms相当は0.1Jの3パターンを集計単位電力量として集計した値を使用する．

まず，アプリケーションの各実行における消費電力量の集計結果を比較する．Stressコマンドは1回の実行で各コア1プロセスが生成されるため，各回8プロセスの消費電力量を合計する．AVXプログラムは1回の実行で生成されるのは1プロセスのみであり，このプロセスの消費電力量を使用する．

表3　AVXプログラムの各実行の消費電力

| | | 測定値 | 集計値 | 誤差 |
|---|---|---|---|---|
| time 100ms | 1 | 9.6 | 5.4 | 44% |
| | 2 | 11.6 | 10.3 | 11% |
| e4 0.5J | 1 | 10.2 | 11.0 | 8% |
| (100ms相当) | 2 | 9.7 | 12.0 | 24% |
| time 50ms | 1 | 9.6 | 8.9 | 7% |
| | 2 | 10.2 | 9.5 | 8% |
| e4 0.3J | 1 | 9.6 | 9.9 | 3% |
| (50ms相当) | 2 | 9.7 | 9.6 | 1% |
| time 10ms | 1 | 11.5 | 11.7 | 2% |
| | 2 | 9.5 | 9.7 | 2% |
| e4 0.1J | 1 | 9.6 | 9.7 | 1% |
| (10ms相当) | 2 | 9.6 | 10.1 | 5% |

表4　stressコマンドの各実行の消費電力

| | | 測定値 | 集計値 | 誤差 |
|---|---|---|---|---|
| time 100ms | 2 | 69.3 | 72.7 | 5% |
| | 3 | 69.7 | 72.2 | 4% |
| e4 0.5J | 2 | 69.6 | 68.0 | 2% |
| (100ms相当) | 3 | 69.8 | 66.5 | 5% |
| time 50ms | 2 | 69.3 | 70.6 | 2% |
| | 3 | 69.5 | 70.5 | 2% |
| e4 0.3J | 2 | 69.5 | 69.6 | 0% |
| (50ms相当) | 3 | 69.8 | 68.1 | 2% |
| time 10ms | 2 | 69.4 | 69.3 | 0% |
| | 3 | 69.6 | 69.5 | 0% |
| e4 0.1J | 2 | 69.5 | 69.3 | 0% |
| (10ms相当) | 3 | 69.7 | 69.5 | 0% |

表 3 に AVX プログラム，表 4 に stress コマンドの結果をまとめる．各値の項目名は，サンプリング種（タイムベース”time”／イベントベース”e4”），サンプリング間隔（100ms／10ms）あるいは集計単位（0.5J／0.3J／0.1J）をつなげたものである．例えば，”e4 0.5J (100ms 相当)”は 100ms 相当間隔のイベントベースサンプリングの結果を 0.5J 単位に集計した値を表す．項目毎に実行された回数分の測定値，集計値，集計値の測定値に対する誤差の絶対値（以降，誤差と表記）をまとめる． AVX プログラムは，サンプリング間隔 10ms（相当）の場合，各回の集計値と測定値の誤差は小さく，5%以下である．サンプリング間隔 100ms（相当）の場合は誤差が大きく，イベントベースで最大 24%，タイムベースでは最大 44%である．各実行における誤差にも幅があり，イベントベースで 16 ポイント，タイムベースでは 33 ポイントの差がある．一方，stress コマンドでは，いずれも 5%以下であり，サンプリング間隔 10ms（相当）の場合はどちらのサンプリングも 0%である．サンプリング間隔 100ms（相当）の場合は少し誤差が大きくなる．どちらのアプリケーションでも，サンプリング間隔が大きいほど，集計値の測定値に対する誤差は大きくなる．

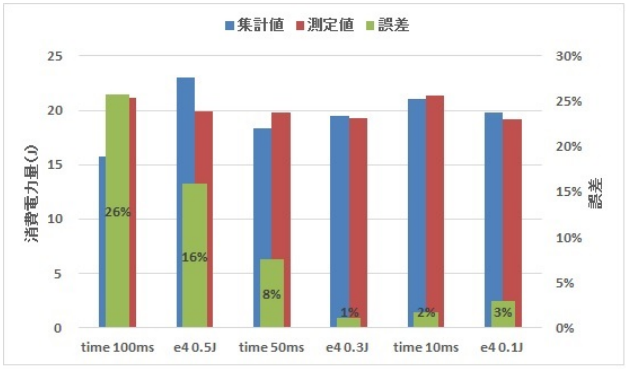


図 6　AVX プログラムの総消費電力

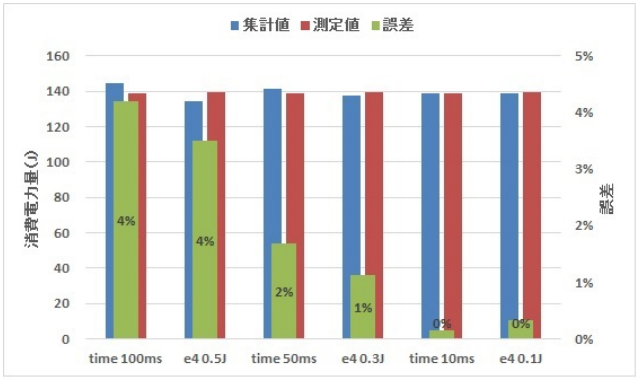


図 7　stress コマンドの総消費電力

次に，2 つのアプリケーションの消費電力量を集計する．stress コマンドは 2 回の消費電力量の和，AVX プログラムは 2 回の消費電力量の和となる．AVX プログラムの消費電力量をまとめたグラフが図 6，stress コマンドの消費電力量をまとめたグラフが図 7 である．2 つの図は，左の縦軸は消費電力量 (J)，右の縦軸は誤差であり，集計値を青の棒，測定値を赤の棒，誤差を緑の棒で表す．項目名は，表 3 と表 4 と同様に，サンプリング種（タイムベース”time”／イベントベース”e4”），サンプリング間隔（100ms／50ms／10ms）あるいは集計単位（0.5J／0.3J／0.1J）をつなげたものである．AVX プログラムの場合，100ms 間隔（相当）サンプリングの場合，イベントで 16%，タイムベースでは 26%の誤差が発生している．50ms 間隔（相当）サンプリングの場合，イベントベースで 1%，タイムベースで 8%の誤差となっている．stress コマンドの場合は，どちらも 5%未満の誤差に収まっている．2 つのアプリケーションともに，イベントベースサンプリングのほうがタイムベースサンプリングよりも集計値の測定値に対する誤差が少ない傾向にある．10ms 間隔（相当）サンプリングの場合は，わずかにタイムベースサンプリングの誤差のほうが少ない．

今回評価した CPU 負荷をかけるアプリケーションに関しては，短時間しか実行されないアプリケーションの消費電力の計測では，サンプリング間隔が大きい場合にはイベントベースサンプリングのほうがタイムベースサンプリングよりも正確に計測できると考えられる．サンプリング間隔が小さい場合には，両者にそれほど違いはない．

## 5. 関連研究

これまで，専用ハードウェアによる電力ベースサンプリングシステム[11][12] が提案されてきた．[11] は PC サーバの消費電力を累積カウンタにより計測する装置及び制御ソフトウェアで構成され，OS やアプリの関数単位の電力消費を時系列で高精度に分析可能である．また，専用計測器から一定時間毎に電力を計測し，同時にプログラム情報をサンプリングする電力プロファイルツールも提案された[13]．このツールではサンプリング結果からプロセスや手続き単位の電力プロファイルを作成する．ツールを使用して，動画再生アプリケーションの消費電力を最大 46%削減できたと報告しており，電力プロファイルの効果を示した研究である．しかし，専用ハードを用いずに電力ベースサンプリングを行う研究は報告されていない．

電力制約適応型システムの実現のため，性能解析ツール TAU [7] でアプリケーションの性能情報を，RAPL で消費電力情報を採取し，2 つの情報を統合する手法 [8] が提案されている．TAU から PAPI(Performance Application Programming Interface)と連携させることで RAPL の電力情報も同時に取得可能だが，実行時間のオーバーヘッドとそれによって生じる消費電力の誤差が大きいため，別々に採

取して統合する方法を採用している．TAU からの電力測定はアプリケーションに計測の開始と終了のための関数を追加することが必要となる．また，TAU の計測間隔は秒単位であり，関数プロファイルとしては荒い．この手法では，RAPL の値を一定時間毎にサンプリングしており，イベントベースサンプリングは行われていない．

提案手法では，専用ハードウェアを使用せずに，CPU の RAPL 機能を用いることにより，電力値を獲得する．電力値の計測にはタイムベースではなく，電力消費に関係する CPU の性能イベントをベースとしたサンプリングを行う．こうすることにより，時間がかかる箇所を採取するのに適したタイムベースサンプリングと比べて，より正確な消費電力量のサンプリングを可能にする．

## 6. おわりに

我々はこれまでに電力消費に関係する CPU の性能イベントをベースとしたサンプリングを行うことにより，従来のタイムベースサンプリングよりも精度の高い電力サンプリングを可能とする手法を提案した．本稿では，この手法を検証するため，電力消費に関係する 4 つの性能イベントをベースとしたサンプリングを同時に行う複数イベントベースサンプリングを実装し，採取したサンプルデータの電力値を一定消費電力量毎に集計することにより，疑似的電力サンプリングを試作した．短時間しか実行されないアプリケーションの消費電力の計測を行った場合において，サンプリング間隔が大きい場合にはイベントベースサンプリングのほうがタイムベースサンプリングよりも正確に計測できることを示した．

今回，イベントベースサンプリングで採取したデータを集計し，プロセスや関数単位の総消費電力量をまとめる電力プロファイルを試した．総消費電力量では長く実行されるものは見つけやすいが，瞬間的に電力を消費するものは発見しにくい．時間による消費電力量の変化を把握できると，ある単位時間における電力消費のホットスポットの発見が可能になる．この情報を利用して，電力を考慮したスケジューリングなどが可能になるかもしれない．そのために，イベントベースサンプリングしたデータに時間情報を取り込むことが考えられる．

## 参考文献

[1] Top500 : http://www.top500.org/
[2] Green500 : http://www.green500.org/
[3] Intel: CHAPTER 14.9 PLATFORM SPECIFIC POWER MANAGEMENT SUPPORT, Intel Software Developer's Manual, Volume 3B, (April 2015)
[4] https://software.intel.com/en-us/articles/intel-power-governor
[5] Srinivas Pandruvada : Running Average Power Limit, https://01.org/blogs/tlcounts/2014/running-average-power-limit-%E2%80%93-rapl
[6] 近藤正章 他：ミニアプリを用いた HPC システムの電力解析，SDHPC10「ミニアプリセッション」(2013)
[7] Sameer S. Shende, Allen D. Malony : THE TAU PARALLEL PERFORMANCE SYSTEM, International Journal of High Performance Computing Applications (Summer 2006)
[8] 大坂隼平 他 : HPC アプリケーションの消費電力最適化に向けた性能・消費電力情報の統合手法,HPCS2015
[9] 小野美由紀 他：ソフトウェアの消費電力分析手法，情報処理学会研究報告，2015-HPC-150 N0.29
[10] 小野美由紀 他：サンプリングによる消費電力の計測手法，HPCS2016-031
[11] 平井聡 他 : 電力ベースサンプリングシステム PARITS の提案，情処第 72 回全国大会
[12] 三輪真弘 他 : 電力ベースサンプリングシステム PARITS の評価，情処第 72 回全国大会
[13] Jason Flinn et al: PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications, WMCSA'99