CPU の高精度電力カウンタを活用した消費電力分析手法

平井聡[1]　福本尚人[1]

概要：エネルギー効率の重要性が高まり，装置規模に関わらずコンピュータシステムにおける電力効率向上のための取り組みが行われている．多くのシステムではメインボード上に電力監視機能を搭載し，ファームウェアレベルで FAN や供給電力の制御が行われており，オペレーティングシステムとの連携により，デバイスを含めたシステム全体の動的な電力制御が行われている．また，汎用プロセッサでは CPU の消費電力や冷却能力に余裕がある際にコア周波数を上昇させて性能を向上させる機能を有しているが．アプリケーションソフトウェアの消費電力を抑える最適化を行うことで，CPU コア周波数上昇の余力を作り出し，アプリケーション単体の消費電力を抑えるだけでなく，システム全体として消費電力対性能比を高めることが可能になる．このような背景から，アプリケーションのエネルギー消費を詳細に分析するための取り組みが行われているが，従来の CPU の消費エネルギー計測機能を利用した方法ではプロセスレベルの粒度の分析が主で，関数レベルまで分解能を高めることが困難であった．
今回我々は，CPU やメモリの消費エネルギー積算カウンタを CPU の PMU(Performance Monitoring Unit)イベントとして実装した A64FX プロセッサを使用し，CPU コアやメモリの一定エネルギー消費毎にサンプリングするプロファイリングを行うことで，低オーバヘッドで関数やベーシックブロック単位の細粒度の消費電力分析が可能であることを確認した．

1. はじめに

ICT システムのエネルギー消費量の抑制は，スマートフォンを始めとしたモバイルデバイスから HPC 向けハイエンドサーバまでコンピュータシステムの装置規模に関わらず重要な課題となっており，様々な低消費電力化の取り組みが行われている．

システムボードレベルで温度や電力を監視して FAN や供給電力を制御したり，オペレーティングシステムと連携した ACPI（Advanced Configuration and Power Interface）によるシステムワイドでの電力制御や DVFS（Dynamic Voltage and Frequency Scaling）によるプロセッサ電源電圧制御などが定常的に利用されている．

アプリケーションソフトウェアにおいてもエネルギー消費を含めたパフォーマンス最適化が行われている．汎用プロセッサには，実行時の消費電力に基づき動作周波数を変更する機能が搭載されることが多い．言い換えると，消費電力を抑えることで動作周波数向上による性能改善を達成できる．アプリケーションのエネルギー消費を分析し削減するためには，実行時全体の値だけでなく，関数単位や特定のコード・セクションごとに測定する必要があり，測定の高い時間分解能が求められる．

ARM CPU を利用したシステムの多くはメインボード上に power monitor chip を搭載しており [1]，Intel/AMD CPU では RAPL (Running Average Power Limit) [2, 3] のような CPU やメモリの消費電力監視制御機能を搭載している．これらの機構は，ボード単位あるいは CPU/メモリ単位の消費エネルギー積算カウンタを有しており，ソフトウェア実行時の消費電力を分析することが可能であるが，計測対象がボード全体あるいは CPU Core パッケージ全体の粒度と粗く，カウンタ更新間隔が最短でもミリ秒以上であるため，一般的にアプリケーション全体およびプロセス単位の粗粒度の消費電力分析として利用されている．

今回我々は，より細粒度の消費電力分析を行うため，消費エネルギー積算電力カウンタをプロファイリングのためのサンプリング・イベントとして利用可能な A64FX Processor [4] を利用し，異なる特性を持つベンチマークプログラムを用いて消費エネルギー・プロファイリングの評価を行った．その結果，通常の CPU 使用率やタイムベースサンプリングなどからでは推定が難しいアプリケーションの関数単位や特定コード・セクションでの消費エネルギーを，専用計測器や測定対象ソフトウェアの改変の必要なしに，100 μs分解能で 2%程度の実行オーバヘッドで計測できることが分かった．

2. 関連研究

2.1 Intel RAPL

アプリケーションソフトウェアのエネルギー消費量を計測するために，Intel CPU に導入された RAPL を利用した手法が広く用いられている．RAPL は Sandy Bridge マイクロアーキテクチャ以降の CPU で利用することができる機能で，図 1 に示すように Last Level キャッシュを含む CPU パッケージ全体（package），CPU パッケージの全コア（pp0/core），およびサーバ機ではメモリ（DRAM），クライアント機ではグラフィックコントローラ（pp1/graphics）の３つのドメインの消費エネルギー積算値の取得ができる．ユーザは，MSR（Model Specific Register）を介して読み出

1（株）富士通研究所
Fujitsu Laboratories Ltd.

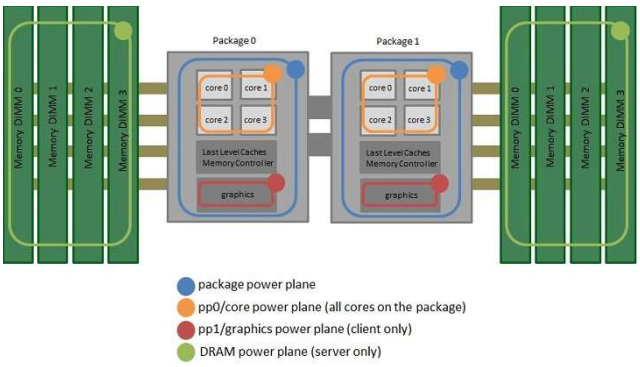

図 1　Intel RAPL 機構（出典[5]）

すことにより，任意時点でのこれらの値を得ることができる．

RAPL 利用の問題点は，CPU コア毎の値でなくパッケージ内の全コア単位でしか消費エネルギーを読み出せないこと，カウンタの更新間隔が 1ms と粗いこと，また RAPL はあくまで積算カウンタであるためプログラムとの関連付けを行うためにはプログラムに読出し機能を入れるか，定期的に RAPL の値を取得しその瞬間に動作していたプログラムと関連付けするような仕組みが必要であることが挙げられる．

2.2　プロファイリングツールと RAPL の連携

プロファイリングツール TAU [6] でのアプリケーション性能情報と RAPL での性能情報を合わせ，2 つの情報を電力軸と時間軸で補正をかける手法 [7] が提案されているが，補正方法が採取データに依存したヒューリスティックな手法であること，またプロファイル間隔が秒単位で関数単位の分析としては粗い問題がある．

より細粒度の分析を行うために，Intel RAPL の読出しタイミングを調整し精度を上げることで短いコードパスの消費電力を分析する手法 [8] や，PMU (Performance Monitoring Unit)を利用して CPU cycles，instructions 等のイベントベースサンプリングと RAPL カウンタ読出しを合わせて関数レベルの消費電力を分析する手法 [9] が提案されている．しかし，前者はアプリケーションに計測用コードを挿入する必要があり，後者は採取する PMU イベントと消費電力の相関などアプリケーションの動作特性を事前に分析しておく必要がある．汎用的に関数やベーシックブロック単位の消費電力分析を行う手法，およびそれをサポートするハードウェア機構は従来報告されていなかった．

3.　消費エネルギー・プロファイリング

本章では、消電エネルギー積算値を PMU イベントとして実装した A64FX Processor を使用し，CPU コア単位，Last Level Cache（=L2 Cache）単位，Memory 単位の３つのドメインそれぞれにおいて，一定エネルギー消費毎にサンプリングするプロファイリングを行うことで，アプリケーションの修正や事前評価なしに細粒度の分析を行う「消費エネルギー・ベース・サンプリング」手法を述べる．

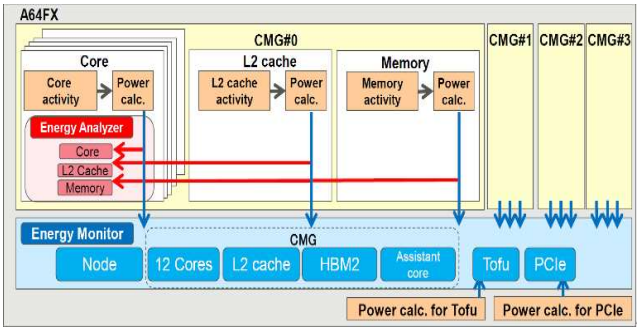

図 2　A64FX Energy Monitor/Analyzer 機構（出典[10]）

3.1　A64FX Processor の消費電力監視機構

A64FX は, Arm-v8.2A 仕様に準拠する富士通製 CPU で，図 2 のような構成および消費電力監視機構を実装している．青色の Energy Monitor は RAPL 同様に CPU Chip レベルで Core/Cache/Memory の消費エネルギーをミリ秒単位で積算するカウンタである．

赤色の Energy Analyzer が新たな機構で，CPU cycle 毎の消費エネルギーを積算するナノ Joule オーダのカウント機能を PMU イベントとして実装しており，Core 単位での計測に加え，CPU Cycles などと同様に指定カウント・アップ毎に割込み発生を可能としている．

なお，消費エネルギーを PMU イベントとして実装した CPU は，Linux Perf のソース内に定義された各種 CPU の PMU イベントを確認した範囲では存在しなかった．

3.2　Linux Perf による消費電力プロファイリング

前述の Energy Analyzer の PMU カウンタの設定/読出し，および一定イベント発生毎のサンプリングを行うためのツールとして，Linux Perf を利用した．

Linux Perf は、Linux Kernel ソースファイルに含まれるイベントデータ収集／トレース機能のフレームワーク，およびユーザコマンドツールの総称である．Kernel 内で補足可能な Context Switch や Page Faults など Software イベントの発生回数や呼出し元情報の収集をはじめ，CPU や周辺コントローラが有する PMU でカウント可能な各種ハードウェアイベントの発生回数やサンプリングデータ採取に至るまで, ソフトウェア／ハードウェア イベントをハンドリングするフレームワークである．

A64FX は, ARMv8.1 PMUv3 仕様に準拠しており, ARM64 用 Linux Perf のデフォルト機能を使用可能である．これは，ARM64 標準の PMU Driver で定義された標準的なイベントは，list 表示や perf コマンド（stat、record 等）での

Mnemonic(cpu-cycles，instructions 等のイベント名称)指定が行えることを意味する．一方，A64FX で追加定義された上記 Energy Analyzer のような固有イベント [11] は，16 進数値の raw 指定入力のみ可能で，Mnemonic による入力や結果表示が出来ない．このため，Linux Perf で消費電力の分析を行うには，A64FX 固有イベントを扱えるようにする対応を行う必要がある．

3.3 Linux Perf の A64FX 固有イベント対応

Linux Perf の perf コマンドソースは，Kernel ソースの tools/perf ディレクトリ下にあり，また ARM64 用の固有イベント定義ファイルは tools/perf/pmu-events/arch/arm64 ディレクトリに以下の形式で格納されている．

```
<arm64> dir
    mapfile.csv      Model ID と Event dir の対応指定
    <arm> dir        cortex-a53 等 Event 定義ファイル
    <ampere> dir     eMAG Processor 用ファイル
    <cavium> dir     thunderx2 Processor 用ファイル
        :
```

ここに <fujitsu>ディレクトリを作成し，その下に A64FX 固有イベント用定義ファイルを格納し，mapfile.csv ファイルに A64FX の Model ID と fujitsu ディレクトリの対応指定を追加した．なお，イベント定義ファイルは json 形式で，以下のようなフォーマットで記述する．

```
[
  {
    "EventName":"EA_CORE",
    "EventCode":"0x1e0",
    "BriefDescription":" Energy consumption per core cycle",
    "PublicDescription":"This event counts energy
                         consumption per cycle of core."
  }
]
```

A64FX の公開ドキュメント [11] をもとに，A64FX 固有イベント定義を追加，tools/perf ディレクトリ下の再コンパイルを行い，生成された perf コマンドを使用して以降の評価を行った．

4. 評価と考察

4.1 評価環境

評価には FUJITSU Supercomputer PRIMEHPC FX700 を使用し，構成は表 1 の通りである．今回の評価では，FX700 1 ノード（1CPU）を使用した．

消費電力測定および消費電力プロファイリングの対象として stress ベンチマークの以下 3 パターンを使用した．

表 1　実験に用いたホスト(FX700)の構成

Processor	Fujistu A64FX
Num of Cores	48 (12 core x 4 CMU) ※Core Memory Unit
Core Clock	2.0GHz
L1 Cache	64KiB I + 64KiB D (per core)
L2 Cache	32MiB (8 MiB x 4 CMU)
Memory	32GiB (8 GiB x 4 CMU)
OS	CentOS 8.1 (4.18.0-147.8.1.el8_1.aarch64)
Benchmark	stress-1.0.4-23.el8.src.rpm
Compiler	GCC 8.3.1

●stress:cpu ベンチマーク

CPU 負荷のみを指定時間かけ続けるベンチマークで，主要コードは以下の通りで，メモリアクセスは発生しない．

```
While (1)
    sqrt (rand ())
```

●stress:vm (Memory) ベンチマーク

指定サイズのメモリ領域を確保し，指定ストライドで write および read を行うベンチマークで，デフォルト設定の 256MB メモリ領域に 4096Byte ストライドとした．なお，メモリ領域確保は 1 度のみで解放せずに再利用するパラメータとした．

●stress:hdd (Disk) ベンチマーク

指定サイズのファイルに対してランダム書込みを指定時間繰り返すベンチマークで，ファイルサイズはデフォルトの 1GB とした．

4.2 評価ベンチマークの特性

まず，各ベンチマークパターンの特性を調べるため，perf stat コマンドを用いて，消費電力，および CPU イベント発生回数を計測した．各ベンチマークパターンの実行は個別に行い，Worker(Process)は 1 つのみで指定時間は 1 秒間，CPU Core Clock は 2GHz 固定で行っている．

図 3 は，各ベンチマークパターンの CPU コア(ea_core)，L2 Cache(ea_l2), Memory(ea_memory)それぞれの消費電力を示している．cpu ベンチマークと hdd ベンチマークを比較すると，全てにおいて hdd ベンチマークの消費電力が大きい．一方，vm ベンチマークは CPU コアの消費電力は cpu および hdd よりも小さいが，L2 キャッシュとメモリの消費電力は両者よりも大きくなっている．これは，各ベンチマークパターンの動作特性である CPU イベント発生回数と関連している．

図 4 は，各ベンチマークパターンの CPU イベント発生回数で，それぞれ CPU コアサイクル数(cycles)，実行命令数(instructions)， L2 Cache アクセス数(l2_cache)，Last Level

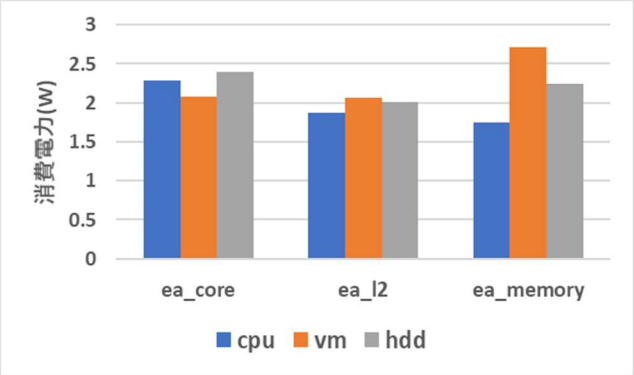

図 3　stress ベンチマーク実行時の消費電力

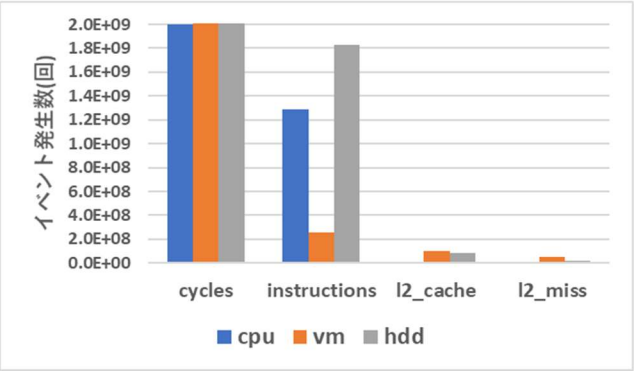

図 4　stress ベンチマーク実行時の CPU イベント発生回数

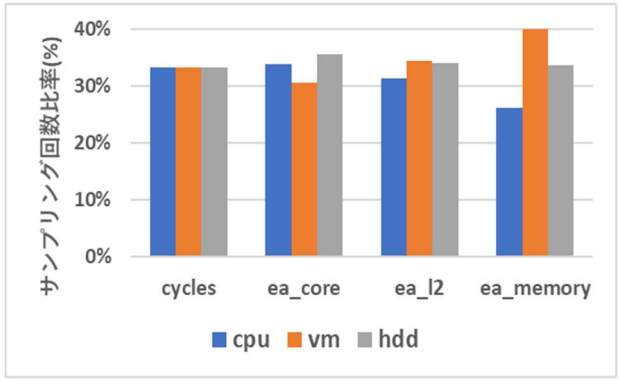

図 5　stress ベンチマーク　サンプリング回数比率

ea_core

Overhead	Enegry (mJoule)	Shared Object	Symbol	Benchmark Category
31.02%	751	libc-2.28.so	__random	cpu, hdd
29.87%	723	stress	hogvm	vm
19.25%	466	[kernel]	__arch_copy_from_user	hdd
3.12%	76	libc-2.28.so	__random_r	cpu, hdd
1.93%	47	[kernel]	__set_page_dirty	hdd
1.07%	26	[kernel]	free_unref_page_list	vm
1.05%	25	[kernel]	get_page_from_freelist	vm
11.63%	284	(others)	(others)	

ea_memory

Overhead	Enegry (mJoule)	Shared Object	Symbol	Benchmark Category
39.43%	946	stress	hogvm	vm
24.77%	595	libc-2.28.so	__random	cpu, hdd
20.95%	503	[kernel]	__arch_copy_from_user	hdd
1.99%	48	libc-2.28.so	__random_r	cpu, hdd
1.54%	37	[kernel]	__set_page_dirty	hdd
0.95%	23	[kernel]	get_page_from_freelist	vm
0.76%	18	[kernel]	free_unref_page_list	vm
8.93%	216	(others)	(others)	

図 6　stress ベンチマーク　プロファイル出力例

Cache ミス数(l2-miss-count) を示している．

CPU 負荷率は，全パターン共に cycles が 2 x 10^9 回であることから 100%であることを示している．実行命令数に関しては，CPU コア消費電力と同じ傾向を示しており，hdd ベンチマークが最も多く，cpu ベンチマーク，vm ベンチマークの順に少なくなる．L2 Cache アクセス数と L2 Cache 消費電力および Last Level Cache ミス数と Memory 消費電力は，vm/hdd ベンチマークにおいては同一傾向であるが，cpu ベンチマークはこれらのカウンタ値がほぼ 0 に近い（L 1 キャッシュ内でのアクセスに収まっている）． cpu ベンチマークの L2 キャッシュ消費電力は 1.88(W)，メモリ消費電力は 1.75(W)であり，これらの値は L2 キャッシュおよびメモリアクセスが殆ど無くても CPU コアが 100%稼働している際に最低限必要な消費電力と考えられる．

4.3　消費電力プロファイル

A64FX Energy Analyzer の PMU イベント（ea_core, ea_l2, ea_memory）に関し，Linux Perf の perf record コマンドを利用してサンプリングによるデータ採取を行った．対象となるプログラムは，stress ベンチマークのソースコードを一部修正し，cpu，vm，hdd の各ベンチマークパターンの関数を一定時間間隔で逐次呼び出すようにした．これは，素性を確認したベンチマークプログラムを連続して関数として呼び出してサンプリングデータ採取を行うことで，サンプリング回数の偏りがないことを確認するために実施した．

図 5 は，Energy Analyzer の各 PMU イベントに CPU コアサイクル数イベントを加え，それぞれ 100 μs相当のサンプリング間隔となるようサンプリングレートを調整し perf record コマンドで取得したデータを，関数（ベンチマークパターン）毎のサンプリング回数を集計してその割合を示したものである．CPU コアサイクル数によるサンプリングは，各関数共に CPU 負荷率 100%のプログラムで同一時間の採取を行ったため割合も同一の 33.3%ずつとなった．Energy Analyzer の PMU イベント（ea_core, ea_l2, ea_memory）に関しては，それぞれ「4.2 評価ベンチマークの特性」で示したベンチマークパターン毎の消費電力の比率と同一であることが確認できた．

次に，取得したデータを perf report コマンドで集計出力し，そのデータを基に付加情報を追加したものが図 6 である．このプロファイル出力例は，3 つのベンチマークパタ

表 2 stress ベンチマーク(cpu) 実行オーバヘッド

Sampling Rate (micro Joule)	Exec Time (second)	Overhead
(Sampling なし)	5.98	—
2272 (≒1 ㎳)	5.99	0.2%
227.2 (≒100 μs)	6.12	2.3%
113.6 (≒50 μs)	6.27	4.8%
56.8 (≒25 μs)	6.58	10.0%

ーンの関数呼び出しを 1 秒間行った際のものである．各ベンチマークパターンの関数が呼び出すライブラリやカーネルの関数も含め，関数単位に集計を行っている．左側から，その関数のサンプル数の割合（つまり該当関数の消費エネルギーがプログラム全体に占める割合），サンプル数から求めた消費エネルギー，関数が属するオブジェクト名，関数名，ベンチマークカテゴリを示している．この例では，複数のベンチマークから呼び出されるライブラリ関数は一つにまとめて集計している．

このように，消費電力プロファイルにより関数単位の消費エネルギーを計測することができ，集計単位を変更することによりライブラリ単位やカーネルの消費電力を求めることも可能である．また，perf report の annotation 機能により，関数のアセンブリコードとサンプリング箇所・回数の対応を表示させることにより，特定コード・セクションでの消費エネルギーを求めることも可能である．

4.4 オーバヘッド測定

サンプリングで発生する割込みの影響を調べるため，stress ベンチマークの cpu ベンチマークパターンのソースコードを一部修正し，主要コードの sqrt(rand()) を 1 億回実行した際の実行時間を，サンプリングレートを変更しながら計測した．表 2 は CPU コア(ea_core)イベントを使用し，サンプリング間隔を 1 ㎳/100 μs/50 μs/25 μs 相当になるようにサンプリングレートを調整し実行した結果である．この表から 1 ㎳相当のサンプリングレートでは 0.2%と殆どオーバヘッドが無く，100 μs相当でも 2%強で収まることが分かる．一方 50 μs相当にすると 5%程度，25 μs相当にすると 10%程度のオーバヘッドが発生するため，プロファイル対象プログラムの構成やオーバヘッドによる挙動変化の有無により適切なサンプリングレートを選択する必要があることが分かる．

5. おわりに

本論文は，消費エネルギーの積算カウンタを PMU イベントとして実装された A64FX を用いて，消費電力エネルギー・サンプリングによるプロファイリングの有効性を評価した．

異なる特性を持つ stress ベンチマークを利用し，通常のタイムベースサンプリングや実行命令数などの CPU PMU イベントからでは推定が難しいプロセスや関数ベースの消費エネルギーを，専用計測器や測定対象ソフトウェアの改変の必要なしに計測できることが確認できた．また，プロファイリングのサンプリングオーバヘッドに関し，100 μs 相当間隔でのサンプリングを行っても 2%程度で収められることが確認できた．

今後は，実用的なアプリケーションを用いた検証，およびプログラム開発フレームワークへの組込みによる実アプリケーション開発への適用を検討していく予定である．

参考文献

[1] D. Abdurachmanov et al: Techniques and tools for measuring energy efficiency of scientific software applications. Journal of Physics: Conference Series, vol.608(1), 2015.

[2] Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide. Intel Corporation, 2019.

[3] S. Pandruvada: RUNNING AVERAGE POWER LIMIT, https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl (参照 2020-05-18)

[4] A64FX Microarchitecture Manual, Fujitsu Limited, 2020. https://github.com/fujitsu/A64FX/tree/master/doc (参照 2020-05-18)

[5] https://software.intel.com/content/www/us/en/develop/articles/intel-power-governor.html (参照 2020-05-18)

[6] S. Shende et al: The Tau Parallel Performance System. International Journal of High Performance, May 2006

[7] 大坂隼平 他: HPC システムにおける性能プロファイリングツールの電力測定精度の評価, 2014-HPC-147

[8] M. Hähnel et al: Measuring energy consumption for short cod e paths using RAPL. Greenmetrics Workshop, June 2012

[9] 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

[10] https://www.ssken.gr.jp/MAINSITE/event/2019/20190820-hpcf/lecture-04/20190820_HPCF_shinjo.pdf (参照 2020-05-18)

[11] A64FX PMU Events, Fujitsu Limited, 2020. https://github.com/fujitsu/A64FX/tree/master/doc (参照 2020-05-18)