Vol.2019-HPC-170 No.2
2019/7/24

アプリケーションの実行速度を重視した省電力ツール

佐賀一繁[†1]　丹生智也[†1]　孫静涛[†1]　高橋公俊[†2]　合田憲人[†1]

概要：近年，HPC システムやクラウドデータセンタでは，システムの消費電力が TCO や環境に与える影響が大きな問題となっている．これに対し，省電力手法のひとつとして CPU クロック周波数と動作電圧を下げて消費電力を削減する Dynamic Voltage and Frequency Scaling (DVFS) が提案されている．しかし，利用者は CPU クロック周波数を下げるとアプリケーションの実行速度が低下すると考え，その仕組みを利用しない場合が多い．そこで，実行時間が低下しない範囲で CPU クロック周波数を自動的に調整する省電力ツールを検討した．まず，CPU クロック周波数と実行時間の関係を調査するため，マイクロアーキテクチャやコア数などが異なる 4 種類のシステムにおいて，設定可能な CPU クロック周波数とスレッド数を変化させて NAS Parallel Benchmarks (NASpara) を実行させた．その結果，実行時間の 5%程度の延伸を認めれば, sp ベンチマークにおいて CPU 消費電力を 60%近く削減できる場合があることを確認した．これは，Last Level Cache (LLC) のキャッシュミスによる CPU ストールが原因と考えられる．しかし，一般にアプリケーションには CPU 周波数依存性の高い部分と低い部分があり，低い部分にのみ CPU 周波数制御を適用しないと省電力効果が減少する．HPC アプリケーションでは，スレッド並列化された大規模データの計算ループが，他の部分に比べ CPU 周波数依存性が比べて低い傾向にある．そこで，OpenMP によってスレッド並列化されたアプリケーションを対象に，実行時に計算ループを自動的に検出し，CPU 周波数依存性特徴の抽出とその特徴に従った CPU クロック周波数制御を行い，実行時間をあまり延伸させることなく自動的に CPU 消費電力を低下させる省電力ツールを提案する．

キーワード：省電力，DVFS，CPU 周波数依存性，OpenMP，機械学習

1. はじめに

HPC システムでは高性能化のためにノード構成やノード数の大規模化が進行している．このため，システムの消費電力の増加が大きな課題となっており，エクサスケールスーパーコンピュータ開発でも解決すべき主要課題の 1 つとして挙げられている[1]．消費電力の問題は，単に運用コストに対する影響だけでなく, 発電のための CO2 の排出や発電からシステムに至る全ての過程で発生する熱の放出などが環境に与える影響が懸念されている．また，近年のクラウドデータセンターではスーパーコンピュータを凌ぐ規模の計算資源を持っており，やはり消費電力が大きな課題となっている．これはサービス価格にも影響を与える．

消費電力を低下させる手法として，CPU のクロック周波数（以下，CPU 周波数と略）と動作電圧を調整して省電力を図る Dynamic Voltage and Frequency Scaling (DVFS)が多く利用されている．CPU の動作状態に応じたミクロな周波数制御だけでなく，アプリケーションの実行状態に応じたソフトウェアよるマクロな CPU 周波数制御に関する研究も多い[2][3][4]．しかし，一般にアプリケーション実行時の CPU 周波数を低下させると実行時間が延伸すると考えられ, 高速性を求める HPC システム利用者には利用されない場合が多いと思われる. またソフトウェアによる省電力は，アプリケーション利用者によるパラメータの検討やアプリケーション開発者によるプログラミングを必要とし，これらはアプリケーションの利用目的や開発目的とは無関係であるため，環境やコストに対する意識が高くない限り，メリットが感じられないことも利用されにくい原因となっている．そこで，対象とするアプリケーションのタイプが限定されるものの，利用者や開発者に負担をかけずに自動的に実行時間の延伸の少ない消費電力の削減が可能な省電力ツールの検討を行った．このとき，特殊なハードウェア機能を必要とすると，システムの選択肢を狭め，システムコストの上昇を招くため，CPU が具備する機能だけで実現することも目標のひとつとした．

一般にどのようなタイプのアプリケーションでも，CPU 周波数に実行時間が影響を受けやすい CPU 周波数依存性が高い部位と，CPU 周波数に実行時間があまり影響を受けない CPU 周波数依存性が少ない部位があり，この CPU 周波数依存性は Last Level Cache (LLC) のキャッシュミスと相関があることが報告されている[2]．まず，現象の確認と情報収集のため，HPC ベンチマークのひとつである NAS Parallel Benchmarks (NASpara) [5][6]を，異なるハードウェア仕様の 4 種類のシステムで測定した．測定では，CPU 機種毎に定義されている設定可能な CPU 周波数とスレッド数毎に，実行時間，平均消費電力，Performance Monitoring Unit (PMU) のカウンタ（以下，PMU カウンタと略）メモリアクセス関連の情報を収集した．結果として，従来研究でも報告されているように，LLC キャッシュミスが多発する sp ベンチマークでは CPU 周波数依存性が低いことが確認され，システムによっては，CPU あたり 19W 以上もの消費電力が削減できること判明した．sp と同様の CPU 周波数依存性を持つアプリケーションを，「京」クラス（約 88,000CPU）の規模のシステムの全 CPU で実行すると仮定すると，全体で 1.7 MW 弱もの省電力となる．

しかし, このような省電力も CPU 周波数依存性の低い部分を検出のみに対し制御しないと実行時間に与える影響が

†1 国立情報学研究所
National Institute of Informatics
†2 総合研究大学院大学
The Graduate University for Advanced Studies

ⓒ 2019 Information Processing Society of Japan

大きい．LLC キャッシュミスが CPU 周波数依存性と相関があることから，HPC アプリケーションでは大規模データの計算ループで CPU 周波数依存性が低いと考えられる．さらにこの計算ループがマルチスレッド化されていると，依存性はさらに低くなると考えられる．ここで，どのように実行中のアプリケーションから計算ループ部分のみを検出するかが問題となる．HPC アプリケーションのマルチスレッド化には OpenMP[7]がよく用いられ，並列化するループの開始部分にディレクティブを挿入する．そこで，省電力対象を OpenMP によってマルチスレッド化されたアプリケーションのみとし，このディレクティブをトリガーとして，CPU 周波数依存性の低い計算ループ部分のみに CPU 周波数制御を行う．

設定する CPU 周波数の決定には機械学習による推論を用いる．あらかじめ様々なメモリアクセスパターンを持つテストプログラムを CPU 周波数とスレッド数を変化させて実行し，このときの PMU カウンタの情報と実行時間を収集，学習させておく．アプケーション実行時の最初の数回の計算ループの実行時に PMU カウンタの値を収集し，先に学習した結果で各 CPU 周波数における実行時間を推測し，この中で実行時間が延伸しない最も低い CPU 周波数を選択，以降のループでこの選択した CPU 周波数で実行する．計算ループの終了時には CPU 周波数を元に戻す．これらの動作により，CPU 周波数依存性の低い部分のみの CPU 周波数制御する省電力を実現できる．プロトタイプによるテストでは，77.4%の精度で適切な CPU 周波数を推測できた．

本報告では，2 章で平均消費電力と実行時間の測定について，3 章で省電力ツールの構成とプロトタイプによる学習実験について，4 章で関連研究について，5 章でまとめと将来構成について報告する．

2. 平均消費電力と実行時間の測定

現象の確認と情報収集のため，HPC ベンチマークのひとつである NAS Parallel Benchmarks (NASpara)の測定を行った． NASpara は，流体計算で多用される複数のアルゴリズムの計算カーネル部を集めた HPC ベンチマークプログラムであり，並列化方法として OpenMP によるスレッド並列，MPI による分散並列，さらに一部ベンチマークではこれらのハイブリッド並列をサポートしている．また，測定に使用する配列も様々なサイズをサポートしている．今回は，OpenMP 版のクラス C の 9 種類のベンチマーク（bt.C, cg.C, ep.C, is.C, ft.C, lu.C, mg.C, sp.C, ua.C）を使用した．クラス C を採用した理由は，測定に使用する配列のメモリサイズが，システムの CPU の Last Level Cache (LLC)のサイズより大きく，かつシステムの物理メモリサイズ以下であるためである．HPC システムのノードでは高速化のために swap を使用しない場合が多く，クラス C の状態が HPC アプリケーションの一般的なメモリ状態を表す最も小規模なものと考えられるためである．なお，ソースコードや Makefile には変更を加えずに，各システム上の gfortran ならびに gcc でコンパイルした実行プログラムを測定に使用した．

測定環境のハードウェア構成を表 1 に示す．ハードウェアによる違いによる影響を確認するために，CPU マイクロアーキテクチャ，コア数，メモリバンド幅など仕様が異なる 4 種類のシステムを使用した．ソフトウェア構成を表 2 に示す．デバイスドライバなどハードウェア依存性があるソフトウェアは基本的にその最新のものを使用したが,CPU の電力管理ドライバとして Intel p-state ドライバ[8]が使用されている場合のみ，ACPI [9]ドライバに換装して測定した．これは，Intel p-state ドライバが，CPU 周波数を自由に設定するための “userspace” ガバナーをサポートしていないためである．

表 1　測定システムハードウェア構成
Table 1　Measurement System Hardware Specifications.

システム	1	2	3	4
	Oracle Cloud BM. Standard2. 52	富士通 ESPRIMO WD2/B2	富士通 PRIMERGY TX1310 M1	DELL R420
CPU	Xeon Plutinum 8167M	Core-i7 7700	Xeon E3-1226v3	Xeon E5-2450
開発コード	Skylake	Kaby Lake	Haswell	SandyBridge
ベース動作周波数	2.0 GHz	3.6 GHz	3.3 GHz	2.1 GHz
CPU 数	2	1	1	1
コア数/CPU	26	4	4	8
スレッド数/CPU	52	8	4	16
LLC 容量	36.6 MB	8 MB	8 MB	20 MB
メモリチャネル数	6	2	2	3
メモリ容量	768 GB	16 GB	28 GB	32 GB
メモリタイプ	DDR4	DDR4	DDR3	DDR3
メモリ周波数	2400 MHz	2400 MHz	1600 MHz	1600 MHz

表 2　測定システムソフトウェア構成
Table 2　Measurement System Software Specifications.

項目	ソフトウェア
OS	・Ubuntu 18.04.02, swap 設定なし, ACPI 使用
コンパイラ	・gfortran 7.4.0 ・gcc 7.4.0
プロファイラ	・perf 4.15.18
CPU 周波数制御	・cpufreq
テストプログラム	・NASpara 3.3 OpenMP バージョン

測定方法は，各測定システム上でベンチマーク毎に CPU 機種毎に定義されている設定可能な CPU 周波数とスレッド数で実行し，実行時間の計測と PMU カウンタ情報の収

集を行った．CPU 周波数の変更には cpufreq-set コマンドを使用し，実行時間の計測と PMU カウンタ情報の収集には perf コマンドを使用した．なお，消費電力は，PMU カウンタの CPU パッケージ消費エネルギー（power/energy-pkg/）を実行時間で割ることで求めた．

図 1 に ep.C を，図 2 に sp.C を各システムの最大スレッド数で実行したときの測定結果を示す．横軸は設定した CPU 周波数である．測定範囲は，CPU 機種毎に定義されている段階的な設定可能周波数のうち，指定通りに設定できた下限周波数から，Intel 社が定義するプロセッサー・ベース動作周波数（以下，ベース周波数と略）[10]までとした．参考までに，システム 2 で使用している Kaby Lake アーキテクチャの CPU では，低い CPU 周波数域において指定した通りに設定されないことがある．縦軸は，平均消費電力 [W]（赤色）と，ベース周波数での実行時間を 1 としたときの各 CPU 周波数での実行時間比（青色）である．

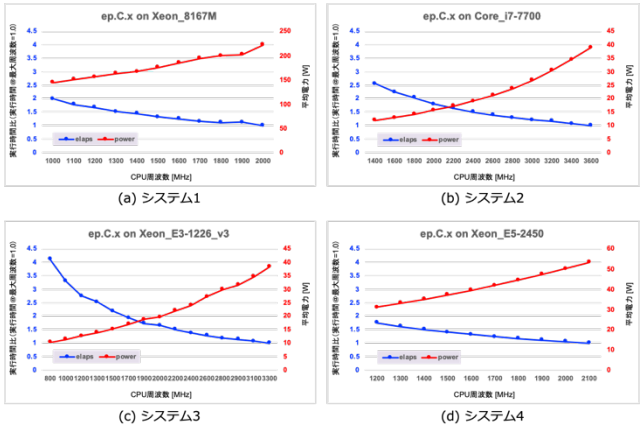

図 1　NASpara ep ベンチマークの測定結果
Figure 1　Results of NASpara ep benchmark

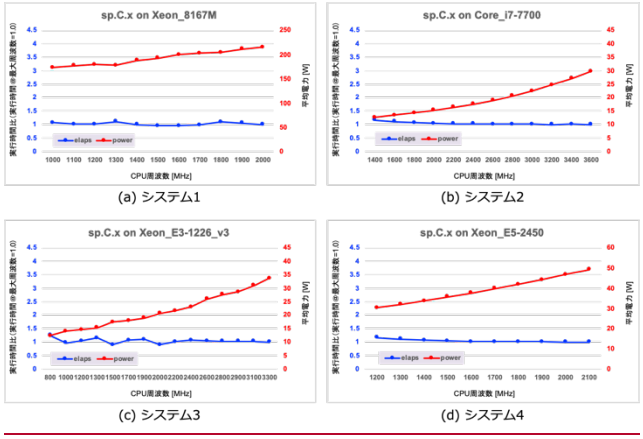

図 2　NASpara sp ベンチマークの測定結果
Figure 2　Results of NASpara sp benchmark

結果として，ep.C は CPU 周波数依存性が最も高く，CPU 周波数を下げると，消費電力が低下し，実行時間が延伸するといった一般的に容易に想像される結果を最も強く示した．また，sp.C は CPU 周波数依存性が最も低く，CPU 周波数を下げると消費電力は低下するものの，実行時間はあまり延伸しないという結果を最も強く示した．なお，ep,C, sp.C とも全ての測定システムで，程度の差はあるが同様の CPU 周波数依存傾向を示した．その他のベンチマークは，消費電力，実行時間比とも ep,C と sp.C 中間の傾向を示し，これらもまたシステムの違いにかかわらず，同様の CPU 周波数依存傾向を示した．ただし，その程度はシステム構成や実行スレッド数によりかなり異なる．

表 3 に，ベース周波数と実行時間延伸が 5%未満である最低の CPU 周波数における CPU の平均消費電力を示す．この結果から，適切な CPU 周波数を設定すれば，CPU 周波数依存性が低い sp.C では 17.6% (システム 1: 37.8 W, 2CPU の合計) 〜 58.3% (システム 3: 19.62 W)もの CPU 消費電力の削減が可能であり，CPU 周波数依存性が高い ep.C でも 5.8% (システム 4: 3.11 W) の削減が可能であること確認できた．CPU あたり 19W の削減はさほど大きくないように感じられるが，もし sp と同様の CPU 周波数依存性を持つアプリケーションを，「京」クラスのスーパーコンピュータ（約 88,000CPU）のすべての CPU で実行することを仮定すると，1.7 MW 弱という非常に大きな消費電力の削減となる．さらに，この 1.7 MW 分の熱の冷却に必要な消費電力も削減できることになる．

表 3　NASpara の消費電力と省電力率
Table 3　Power consumption and power-saving rates

ベンチマーク	システム1 (2CPU合計) 消費電力 (W) ベース周波数	低電力周波数	削減電力 (W)	省電力率 (%)	システム2 (1CPU) 消費電力 (W) ベース周波数	低電力周波数	削減電力 (W)	省電力率 (%)
bt.C	223.01	212.41	-10.59	-4.8%	40.95	36.71	-4.24	-10.3%
cg.C	235.51	220.37	-15.15	-6.4%	37.57	33.85	-3.72	-9.9%
ep.C	222.01	222.01	0.00	0.0%	39.06	39.06	0.00	0.0%
ft.C	248.42	248.42	0.00	0.0%	42.11	37.75	-4.36	-10.3%
is.C	183.90	176.83	-7.07	-3.8%	32.73	29.97	-2.75	-8.4%
lu.C	225.32	220.48	-4.84	-2.1%	33.38	20.63	-12.75	-38.2%
mg.C	210.59	178.20	-32.39	-15.4%	32.36	18.85	-13.51	-41.8%
sp.C	214.88	177.08	-37.80	-17.6%	29.77	15.26	-14.51	-48.7%
ua.C	217.87	212.37	-5.50	-2.5%	32.69	29.46	-3.23	-9.9%

ベンチマーク	システム3 (1CPU) 消費電力 (W) ベース周波数	低電力周波数	削減電力 (W)	省電力率 (%)	システム4 (1CPU) 消費電力 (W) ベース周波数	低電力周波数	削減電力 (W)	省電力率 (%)
bt.C	40.49	36.58	-3.90	-9.6%	57.46	54.22	-3.23	-5.6%
cg.C	39.52	39.52	0.00	0.0%	52.56	40.53	-12.03	-22.9%
ep.C	38.34	38.34	0.00	0.0%	53.33	50.22	-3.11	-5.8%
ft.C	43.59	39.78	-3.81	-8.7%	59.16	55.74	-3.42	-5.8%
is.C	32.76	30.38	-2.37	-7.2%	50.91	48.12	-2.79	-5.5%
lu.C	34.06	16.72	-17.34	-50.9%	52.83	42.56	-10.27	-19.4%
mg.C	34.68	20.81	-13.87	-40.0%	49.68	36.10	-13.59	-27.3%
sp.C	33.68	14.06	-19.62	-58.3%	49.28	35.79	-13.49	-27.4%
ua.C	35.20	32.13	-3.07	-8.7%	50.98	41.04	-9.94	-19.5%

3. 省電力ツール

HPC システムでは，利用者が CPU 周波数や実行時間の許容延伸度を設定して消費電力を削減できるジョブマネージャ[11]などが製品化されており，またソフトウェア開発者がプログラム中から利用できる省電力ライブラリ[12]も提案されている．しかし，利用者や開発者にとって省電力は本来の業務目的ではないため，操作や作業を求められると利用されない場合や利用が後回しにされる場合が多いと

ⓒ 2019 Information Processing Society of Japan

考えられる．このため，利用者や開発者に操作や作業を求めない省電力ツールが必要と考える．つまり，CPU が負荷状態に適した CPU 周波数に自動的に調整してミクロレベルの省電力を実現しているように，アプリケーションの動作状態に適した CPU 周波数に自動的に調整するマクロレベルの省電力を実現するシステムツールである．このとき，特別なハードウェアを必要とする省電力ツールでは，システム選択肢を狭め，コスト高になることもあると考えられるため，たとえ利用方法に制限があっても，CPU が具備している機能のみで実現できることが重要である．そこで，アプリケーションの CPU 周波数制御による省電力が有効な部分の実行が開始されたことを自動的検出し，PMU カウンタの情報から CPU 周波数を制御するような省電力ツールの実現を検討し，プロトタイプによる評価を行なった．

3.1 基本アイデア

上で議論したように，CPU 周波数制御による省電力が有効である条件は，CPU 周波数依存性が低いことである．また，アプリケーションには必ず CPU 周波数依存性の高い部分と低い部分の両方が存在し，低い部分の実行時間の割合が高くないとあまり省電力効果はない．さらに，CPU 周波数の切り替えに時間を要することを考慮すると，CPU 周波数依存性の低い部分の実行時間がある程度続く必要がある．HPC アプリケーションにおいては，大規模データの計算ループがこのような条件に該当する．特に計算ループがスレッド並列化されている場合，スレッド数分だけメモリアクセス回数が増加するため，LLC キャッシュミスが起こりやすくなること多く，CPU 周波数依存性が低くなる傾向にあると考えられる．そこで，今回は OpenMP によってマルチスレッド化されたアプリケーションに限定することで，実行中アプリケーションの計算ループ部分のみを自動的に検出することを可能とし，その計算ループの CPU 周波数依存傾向にしたがった CPU 周波数制御を行うことで，実行時間の延伸が少ない省電力ツールを実現することを検討した．

このようなツールを実現する基本的な課題は，誰がどのように実行中のアプリケーションから CPU 周波数制御が有効な計算ループ開始と終了を検出し，誰がどのように適した CPU 周波数を決定するかである．まず，誰がどのように CPU 周波数制御が有効な計算ループの開始と終了を検出するかを検討する．今回対象とした OpenMP による並列化は，アプリケーション開発者がマルチスレッド化有効と判断したループの先頭に OpenMP ディレクティブを挿入することで行われる．一般にこのようなループが，CPU 周波数制御が有効なループである場合が多いと考えられるため，本ツールではこれを利用する．つまり，開発者が高速化のために書いたディレクティブを利用して省電力機能を埋め込むことで，課題である「誰が」をアプリケーション自身とすることができる．また，「どのように」には 2 つの方法が考えられる．ひとつは，プリプロセッサによる方法である．上記 OpenMP ディレクティブのループ開始（” omp for”や “omp do”，並列化開始）と対応するループ終了（並列化終了）を検出し，本ツールの CPU 周波数制御関数を埋め込む方法である．挿入例を図 3 に示す．ただし，開発者がアプリケーションコンパイル時にプリプロセッサを指定しする必要があることと，本プリプロセッサを使ってコンパイルしたアプリケーションしか省電力機能が動作しないという問題がある．もう一方は，OpenMP でマルチスレッド化された実行プログラムでは，並列化開始や終了位置などで OpenMP ライブラリの関数が呼び出されることを利用する．これら OpenMP 関数をフックして，本ツールの CPU 周波数制御関数を呼び出す．この方法あれば，利用者や開発者の労力なしに，かつ実行プログラムしかないアプリケーションでも省電力が可能となる．

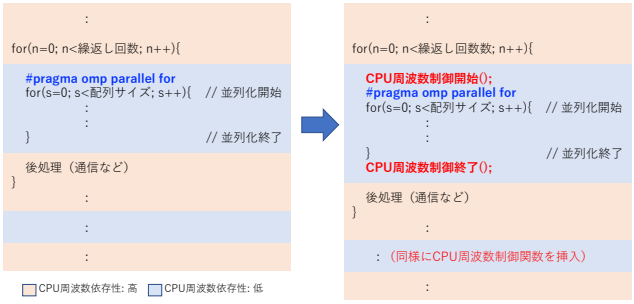

図 3　CPU 周波数制御関数の挿入例

Figure 3　Example of CPU freq. control functions insertion

次に誰がどのように適した CPU 周波数の決定と設定を行うかである．「誰が」については並列化開始で呼び出される本ツールの CPU 周波数制御開始関数内で行う．「どのように」については，CPU 周波数の決定方法としてメモリ消費電力と実行時間の許容低下率から計算する方法が提案されているが[2]，マザーボードにメモリ消費電力測定の対応が必要であり，本ツールの外部機能を使用しないという方針に合致しない．本ツールでは CPU の PMU カウンタ情報のみを利用した機械学習による方法を提案する．

機械学習による方法では，あらかじめ CPU 周波数制御のための学習を行っておく必要がある．具体的には，システムの導入時の運用準備段階で様々なメモリアクセス方法を実行するテストプログラムを全ての設定可能な CPU 周波数とスレッド数で実行し，各実行での PMU カウンタの情報と実行時間を収集，その関係を機械学習させておく．システム運用が開始され，利用者によりアプリケーションが実行された時，OpenMP の並列化開始で呼び出される本ツールの CPU 周波数制御開始関数にて，初期の数回のループ実行で学習時と同じ種類の PMU カウンタの情報を収集，これと実行時のスレッド数などを先の学習済みニューラルネットワークに入力し，全ての設定可能な CPU 周波数で実行させた時の実行時間比を推論させる．そして，その中か

ⓒ 2019 Information Processing Society of Japan

ら実行時間が延伸しない最も低い CPU 周波数を選択, 以降のループの実行時にこの CPU 周波数を適用することで省電力を図る．OpenMP の並列化終了時にも本ツールの CPU 周波数制御終了関数が呼び出されるため, この中で CPU 周波数を並列開始時の値に元に戻すことで，CPU 周波数依存性の高い部分での実行時間の延伸を防ぐ. このように, CPU 周波数依存性が低い計算ループ部分のみに適した CPU 周波数を適用することで，実行時間の延伸が少ない省電力を実現することができる．

3.2 簡易プロトタイプによる学習実験

上記アイデアによる省電力ツールを開発するにあたり，まず機械学習による CPU 周波数制御の有効性を探るため，簡易プロトタイプで学習実験を行なった．結論として，非常に少数ではあるものの矛盾ある学習データの存在や，実行時間比の推論の正答率向上のため，学習データ項目のさらなる検討が必要であると考えられる．

(1) 学習データ項目

入力側学習データの条件は，自動的に取得できる情報のみで情報で構成できることである. CPU 周波数依存性は, LLC キャッシュミスによる CPU ストールが短時間に多数発生することが原因と考えられる. これに関連する情報として，PMU カウンタの LLC キャッシュミス回数とスレッド数が考えられる．しかし，LLC キャッシュミス回数は，CPU 機種が異なるとキャッシュサイズやメモリなどの条件が同じでも，同じアプリケーションで全く異なる値を示すことが知られている．システムの違いに依存しない汎用的なツールを目標とするため, 今回は LLC キャッシュミス回数だけでなく，ストール回数関連の PMU カウンタの情報も入力側学習データも追加した．具体的には，PMU カウンタの LLC キャッシュミス回数（cache-misses）を実行時間で割った値，リソース関連のストールサイクル数（resource_stalls.any）を実行時間で割った値，インストラクション数（instructions）をサイクル数（cpu-cycles）で割った値，スレッド数，実行時 CPU 周波数の 5 種類を入力側学習データとした．

出力側学習データは, 設定可能な各 CPU 周波数の実行時間比とした（図 1, 2 における青色線の値）．この形式であれば，CPU 周波数決定時に実行時間の許容延伸度を自由に設定することができる．

(2) ニューラルネットワーク構成

ニューラルネットワークによる推論時間は，アプリケーション内の CPU 周波数制御対象の各ループの実行時間に含まれる．1 回のアプリケーション実行における，対象ループで少なくとも 1 回は推論が必要であり，かつ対象ループはひとつとは限らないため，ニューラルネットワークの構成は極力小さいことが求められる．そこで今回は，3 層ニューラルネットワークで検証した．隠れ層ユニット数は小さすぎると収束に時間がかかり，さらに過学習が発生する可能性もあるので，学習の状況を見てユニット数を決定した．

- ●入力層ユニット数：5
- ●隠れ層ユニット数：学習状態により決定
- ●出力層ユニット数：設定可能 CPU 周波数の数による

(3) 学習実験

学習データを作成するために，メモリアクセスだけのループのテストプログラムを開発した．アクセス方法として，連続アクセス，ランダムアクセス，ストライドアクセス，ステンシルアクセス（2D:1 種，3D: 2 種）をサポートしている．このテストプログラムと NASpara を，CPU 周波数とスレッド数を変化させながら，上記 PMU カウンタの値の収集と実行時間の計測を行ない, 各 CPU 周波数におけるカウンタ値と実行時間比とを対にしたデータを作成した．このデータ対のうちランダムに選択した 2/3 で学習を行った. 実際の学習データ数は，システムのスレッド数や設定可能な CPU 周波数数により異なる．また，システム隠れ層ユニットは 64 ユニットとした. 少ない隠れ層ユニットでは収束に時間を要したためである

この学習結果を用いて，学習に使用していない上記 1/3 のデータで推論テストを実施した. NASpara の sp ベンチマークにおいて一番省電力効果が高かったシステム 2（学習データ数：1621，テストデータ数：811）において，5%以内の実行遅延を許容する CPU 周波数の推測を行ったところ，77.4%の正答率であった．全く CPU 周波数制御しないと消費電力削減の可能性が 0%であることを考えると良い結果であると言える．なお，学習時になかなか収束しないので調査したところ, 入力側学習データは値が似ているものの，出力側学習データが全く違うデータを数件検出した．つまり矛盾があるデータが存在していた．このようなデータが存在する原因は，データ収集時のノイズと入力側学習データ項目が不足している可能性が考えられ，今後詳細に検討していく．

4. 関連研究

三輪らは[2]，PMU カウンタが異常監視などにも利用され，消費電力のための利用が制限される場合があることを考慮し，メモリ消費電力とアプリケーション特性に相関があることを利用してシステムの消費電力を削減する提案を行っている．外部のコントローラによるアプリケーション特性の判定も可能で，システムに負荷をかけない省電力機能の実現が可能である．

ライプチヒ・スーパーコンピューティング・センターでは，DVFS を使用してスーパーコンピュータの省エネルギーを図っている[13][14]. 具体的には，同センターで利用する多くのアプリケーションについて，あらかじめ様々な CPU 周波数で実行し，消費エネルギー（消費電力 x 実行時間 Wh）を計測，同じアプリケーションの次回から実行は，消

ⓒ 2019 Information Processing Society of Japan

費エネルギーが小さくなる CPU 周波数で使用するものである．これは，環境のためにはアプリケーションの実行速度をある程度犠牲にしても構わないポリシーでの省電力方法であり，HPC センターとしては珍しい使い方である．利用者は実行時間の許容延伸率（%）を指定できるようになっている. おそらくこの消費エネルギー最小の CPU 周波数は，本報告の実測（図 1）における実行時間比が上昇し始める付近の CPU 周波数，もしくはそれより若干低い CPU 周波数であると思われ，アプリケーションによっては，ほとんど実行時間の低下はない可能性がある．

Pakin らも[4]，CPU 周波数毎の消費エネルギーを推定する計算方法を提案しており．かなり高い推定精度が報告されている．エネルギー消費量を CPU 周波数，負荷，ベース周波数での消費電力の関数とする回帰モデルになっている．

吉田らは[15]，消費電力の削減ではなく,アプリケーションの実行特性による CPU とメモリなどのデバイスへの電力の配分変更を可能とする事で，仮想的なシステム構成の変更を可能とし，電力効率の向上を図っている．利用可能な電力の利用効率を上げることができ，エネルギーの有効利用につながるものと思われる．

5. まとめ

アプリケーションの実行時間の延伸が少なく，利用者や開発者の操作が不要な省電力ツールについて検討を行った．DVFS による省電力手法は，一般にアプリケーションの実行時間の延伸を招くと思われ利用されない場合が多い．しかし，アプリケーションによっては実行時間が延伸しないことを複数のシステムの測定によって示し，システムによっては 19%以上もの消費電力を削減できることを示した．また，従来の DVFS による省電力ツールは利用者による設定や開発者によるプログラムへの省電力関数の埋め込みなどの作業が必要であり，省電力は利用者や開発者の本来の業務目的ではないため利用されにくい．このため，これら作業が不要な自動化された省電力ツールの必要性を説いた．その実現方法として，OpenMP のディレクティブをトリガーとして，アプリケーションのコンパイル時もしくはアクリケーションの実行時に省電力機能を自動的に埋め込む方法を提案した. また実行時間の延伸が少ない CPU 周波数の決定に機械学習を用いる方法を提案し，プロトタイプで 77.4%の正答率を確認した．

今後は，学習データ項目の検討による推論精度の向上，本格的なプロトタイプの開発, 一般 HPC アプリケーションでの省電力実証実験などを実施する予定である．

参考文献

[1] DOE ASCAS Subcommittee. Top ten Exascale Research Challenges. DOE ASCAS Subcommittee Report. Feb. 2014.
[2] 三輪真弘，中島耕太，平井聡，風間哲，原靖，成瀬彰．メモリ消費電力に基づく CPU 周波数動的制御手法の評価．情報処理学会論文誌．コンピューティングシステム, Vol5, 2012.
[3] K. Choi, R. Soma, and M. Pedram. Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade-off based on the Ratio of Off-chip Access to On-chip Computation Times, IEEE. Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'04), 2004.
[4] Scott. Pakin, M. Lang, Energy Modeling of Supercomputers and Large-Scale Scientific Applications. IEEE. 2013 International Green Computing Conference Proceedings, 2013.
[5] "NAS Parallel Benchmarks". https://www.nas.nasa.gov/publications/npb.html, (参照 2019-02-05).
[6] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan and S. Weeratunga. THE NAS PARALLEL BENCHMARKS. RNR Technical Report. March 1994, RNR-94-007.
[7] "OpenMP". https://www.openmp.org, (参照 2019-02-05).
[8] "Intel P-State driver", https://www.kernel.org/doc/Documentation/cpu-freq/intel-pstate.txt, (参照 2019-02-05).
[9] "ACPI: Advanced Configuration & Power Interface". http://www.acpi.info, (参照 2019-06-16).
[10] "インテル Core i7-7700 プロセッサー". https://ark.intel.com/content/www/jp/ja/ark/products/97128/intel-core-i7-7700-processor-8m-cache-up-to-4-20-ghz.html, (参照 2019-06-16).
[11] "IBM Spectrum LSF energy aware scheduling". https://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/lsf_welcome/lsf_kc_eas.html, (参照 2019-06-17)
[12] "Sandia National Laboratories High Performance Computing Power Application Programming Interface (API) Specification". https://powerapi.sandia.gov, (参照 2019-06-16).
[13] Luigi Brochard, Chong Chen, Herbert Huber, Aaron Andersen, Terry Hewitt. Maximize Data Center Power Efficiency through Energy Aware Computing. bof121s1, SC13, November 2013.
[14] 近藤正章, エクストリームスケールコンピューティング時代の省電力技術, ACSI2015 チュートリアル.
[15] 吉田匡兵，佐々木広，深澤圭一郎，稲富雄一，上田正嗣，井上弘士，青柳睦. CPU と主記憶への電力バジェット配分を考慮した HPC アプリケーションの性能評価．情報処理学会研究報告, vol2013-HPC-141 No.21.

ⓒ 2019 Information Processing Society of Japan