Vol.2017-HPC-158 No.3
2017/3/8

Deep Q-Networkを用いての計算機の制御による電力最適化

寺西 賢人[1,a)]　野村 哲弘[1]　松岡 聡[1]

概要：近年のスーパーコンピュータは大量に電力を消費するようになり，実用的なスーパーコンピュータの性能の向上には電力効率が課題となっている．省電力手法としてはCPUの周波数や電圧などの制御による電力の最適化があり，その制御に適した値をパフォーマンスカウンタなどのデータを用いて算出する研究が多く進められている．しかし，既存の研究では各データを詳細に解析する手法を取っており，扱うデータ数の制限や環境の変化による再解析を必要としている．そこで我々は，近年研究が盛んに行われている深層学習を用いて解析をする汎用性が高い制御方法を提案する．特にゲーミングや囲碁のAIなどで使用されているDeep Q-Networkという深層強化学習手法によって計算機を直接制御する装置を実装し，評価する．

1. はじめに

近年のスーパーコンピュータではGPUを大量に搭載することもあり大量の電力を消費するようになっている．演算性能の向上につれてさらに消費電力が増大してしまうが，消費電力が膨大すぎるスーパーコンピュータはあまり実用的ではない．そのため省電力がHPCの分野で大きな課題となっている．省電力手法としてはCPUの動作周波数や電圧の制御により電力の最適化を行うDVFS(Dynamic Voltage and Frequency Scaling)という手法があり，機械学習などを用いてその制御に適した値を算出する研究が多く進められている．既存の周波数制御による省電力手法は各データを詳細に解析して適した制御を行うという手法を取っており，扱うデータの数や種類などの制限や環境の変化による再解析を必要としている．そのため様々なデータや環境に対応させることが難しく汎用性が低くなってしまっている．

本論文では，近年研究が盛んに行なわれている深層学習を用いて解析をする計算機制御手法を提案する．深層学習は多層のニューラルネットワークによって様々な特徴量を抽出して適した値を出力する学習方法であり，この手法によりAI技術が著しく発展している．特にDQN(Deep Q-Network)という深層強化学習手法を使用したゲーミングや囲碁などのAIは既存のものと比べて飛び抜けて良い結果を出し続けている．DQNは周波数制御による電力最適化にも適していると考え，DQNを用いて様々な入力データから計算機の周波数を直接制御する汎用性の高い装置を深層学習用ライブラリChainerによって実装し，評価する．

[1] 東京工業大学
Tokyo Institute of Technology
[a)] teranishi.k.aa@m.titech.ac.jp

2. 背景

2.1 省電力と周波数制御

近年のスーパーコンピュータでは演算能力の向上のためにGPU(Graphics Processing Unit)を搭載することが増えている．高い演算能力を誇るGPUだが消費電力がCPUに比べて大変大きいという問題があり，消費電力を抑えることが重要視されるようになってきている．省電力を重視したスーパーコンピュータを評価する場として，演算速度を競うTop500とは別に省電力を競うGreen500というランキングが存在する．東京工業大学のTSUBAME-KFC/DL(TSUBAME Kepler Fluid Cooling/Deep Learning)では油性冷却溶媒液を用いて冷却に使用する電力を抑えることにより省電力化を実現しており，Green500でも高い評価を得ている．

計算機の動作周波数と電圧の動的な制御により省電力化を行うDVFS(Dynamic Voltage and Frequency Scaling)という手法がある．電力は電圧の2乗や周波数に比例するので，それらを制御することで消費電力を減らすことができる．しかし，周波数を下げると電力だけではなく実行速度も下がってしまうため適した周波数設定が容易にはわからない．そのため機械学習などによって様々な方法で解析して適した周波数を調べる研究が多く進められている．

[1][2] しかし，どれも詳細な解析を行うため学習に用いるデータや環境が限定的になり汎用性が低くなってしまっている．

2.2 Deep Q-Network

DQN(Deep Q-Network) は Google Deep Mind 社が開発した深層強化学習手法である [3]．Q 学習に深層学習を組み合わせたアルゴリズムであり，そこからこのように名付けられた．

2.2.1 深層学習

深層学習（Deep Learning）とは機械学習の手法の一つであり，近年盛んに研究が進められている．深層学習は多層のニューラルネットワークによって様々な特徴量を抽出して適した値を出力する学習方法であり，この手法により AI 技術が著しく発展している．特に様々な特徴から判断する必要がある画像処理 [4] や音声認識 [5] の分野の AI で深層学習は活用されている．応用例も多岐にわたり，言語の翻訳 [6] や車の自動運転 [7][8]，ゲームプレイの AI[3][9] などにも使われている．本研究では特にゲームプレイの研究で用いられている手法を扱う．

2.2.2 強化学習

強化学習（Reinforcement Learning）とは機械学習手法の一つであり，主に行動決定を行う問題を扱う手法である．強化学習では環境とエージェントという 2 つの概念から構成される．環境とはあるものの状態を示す．エージェントは環境から現在の状態を得て，特定の方策から行動を選び環境の状態を遷移させる．その後環境はエージェントに対してその遷移に応じた報酬を与える．最終的にエージェントの得られる報酬を最大化するような行動を選んでいく方策を学習するのが強化学習である．

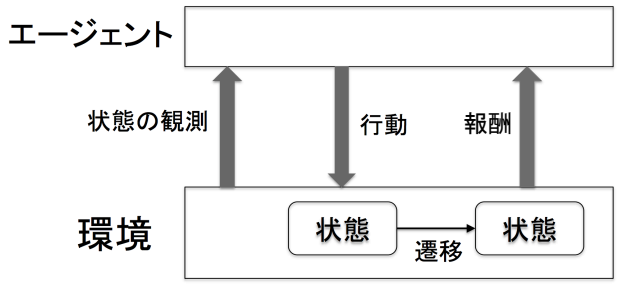

図 1　強化学習

2.2.3 Q 学習

Q 学習とはエージェントがある状態 s で行動 a を選択する場合の将来的に期待できる報酬の合計 Q から行動の方策を学習する手法である．この報酬の期待値を行動価値関数と呼び，$Q(s,a)$ で示す．t step 目の状態が s_t で行動 a_t を選ぶと報酬 r_t が得られるとすると，

$$Q(s_t,a_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^n r_{t+n} \qquad (1)$$

という式で示される．γ は割引率といい 0 から 1 の間の値をとる数で，より遠い未来に得られる報酬は小さくなるよう補正をかける．そして学習した $Q(s,a)$ が最大となるような行動 a を常に取り続ける方策を取ることで，エージェントが得られる報酬 r の最大化を目指す．この $Q(s,a)$ の値は最初はわからないが，実際に行動を選び報酬を得ることを繰り返し試行錯誤することで最適化していく．$Q(s,a)$ の一般的な更新式は式 2 のように示される．

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)] \quad (2)$$

α は学習率であり新しい Q 値にどれぐらい誤差の大きさを反映させるかを示す．この更新を行うためには $\max_a Q(s_{t+1},a_{t+1})$ を把握する必要がある，つまり状態遷移後の状態 s_{t+1} からさらに全ての行動を取った場合それぞれのことを把握する必要がある．そのため膨大な学習ステップが必要になるので，現実的には状態の数が有限で少なめな場合しか扱えないという問題がある．

2.2.4 DQN について

DQN はニューラルネットワークによって行動価値関数 Q を算出する Q 学習手法であり，環境の状態 s_t を入力として Q の値を出力とする．Q 学習の更新式をもとにした誤差を損失関数としてニューラルネットワークの重みを更新する．大量の状態があるような問題に対しても深層学習なら Q 関数を導出できるので，DQN によって様々な問題で Q 学習を行うことが実現出来る．また，DQN には学習を上手く進めやすくするためにいくつかのテクニックがあり代表的なものは，Experience replay，報酬のクリッピング，Target Q-Network がある．

Experience replay

DQN では実際に行動した結果をサンプルとして扱うが，学習を行う際に Experience Replay という手法を用いる．状態 s の時に行動 a を実行し報酬 r を得て状態は s' に遷移したとすると，(s, a, r, s') の組みを経験として蓄積する．この作業を行いつつ，蓄積された経験から一定数をランダムに抽出し，それをサンプルに Q 学習を行うという手法が Experience Replay である．DQN で扱うデータは時系列のデータであるため，近いデータ同士で強い相関関係がある．そのため新しいデータを順番に選び学習を行うとデータ間に相関性が出て過学習が起きてしまうので，相関性を減らすためにこの手法が採用されている．

報酬のクリッピング

強化学習では報酬を元に方策の更新を行う．行動の結果良い報酬を得られたとしてもある程度の差が出ることがある．例えばゲームではスコアの増加量，囲碁ではどれぐらいの差で勝ったかの違いがある．しかし，DQN ではその報酬の値をクリッピングする．良い報酬なら+1，悪い報酬なら-1，どちらでもないなら 0 と

いうように報酬の種類を 3 段階に狭めるのである．こうすることで報酬の重みづけは出来なくなってしまうが様々な問題に対応できるようになり，学習が進みやすくなる．

Target Q-Network

Q 学習の更新式である式 2 より，深層学習における損失関数 loss は

$$loss = r_t + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \quad (3)$$

となり，

$$r_t + \gamma \max_a Q(s_{t+1}, a_{t+1}) \quad (4)$$

は教師データの役割を果たす．しかしこの更新により計算モデルが更新され $Q(s_{t+1}, a_{t+1})$ 自身の値も変わってしまうので教師データが変わってしまう．更新する度に教師が変わると振動して学習が上手くいかない可能性がある．そこで $Q(s_{t+1}, a_{t+1})$ を計算するモデルは学習中のモデルとは別に固定したターゲットモデルを扱う．ターゲットモデルは一定周期が経つと学習中のモデルからコピーを行う．こうすることで教師データをある程度固定し学習を進めやすくするという手法が Target Q-Network である．

2.3 使用するソフトウェア

2.3.1 RAPL

Intel の Sandy Bridge マイクロアーキテクチャ以後のプロセッサには，RAPL(Running Average Power Limit) と呼ばれる電力監視インターフェースが備えられている．RAPL インターフェースではパフォーマンスカウンタや温度などを用いてプロセッサの消費電力を範囲で分けて別々に計測・制御を行うことができる．今回の実験では CPU を 2 つ搭載したマシン上で行うので，CPU1,2 それぞれでチップ全体，コア部分，DRAM の消費電力の計測が可能である．RAPL を用いることで高精度の電力計測が可能であり，RAPL で取得した電力はノード全体の電力と高い相関関係がありモデリングができることなどが判明している [10]. よって，RAPL により計測できる消費電力を扱った実験をすることでノード全体の消費電力についての研究にも繋がることがわかる．

2.3.2 PAPI

本研究ではパフォーマンスカウンタの計測を行う．パフォーマンスカウンタとはプロセッサの演算やメモリアクセスなどの実行回数を表すカウンタであり，アプリケーション実行時にその特徴ごとに対応したカウンタの値が増加する．今回の実験ではプロセッサのハードウェアカウンタを読み出すためのライブラリ PAPI(Performance Application Programming Interface) を用いて計測を行う．なお，PAPI は近年のメジャーなマイクロプロセッサのほとんどに対応しており，十分に可搬性がある．PAPI では各イベント毎に対応するカウンタ名が設定されており，例えば PAPI_L2_TCM というカウンタは L2 キャッシュミスの回数を表している．計測できるカウンタの種類は環境によって変化し，同時に計測可能なカウンタの組み合わせにも制限があり事前に調べておく必要がある．あるプロセスでのパフォーマンスカウンタの値は，プロセス ID とカウンタのイベント名と範囲を指定して PAPI の関数を実行することで計測することができる．

2.3.3 Chainer

深層学習を扱うプログラムの実装を補助するために，様々なライブラリが提供されている．例えば Chainer，TensorFlow，Caffe，Theano，Keras などがあるが，本研究では Chainer[11] を用いて学習プログラムを実装した．Chainer は Preferred Networks が開発した Python のライブラリであり，ニューラルネットワークの構築が直感的かつ柔軟にできることや GPU での学習にも容易に対応出来るという特徴を持っている．DQN での周波数制御では時系列データを用いて随時学習を行う必要があったため Chainer を採用した．Chainer は日本の企業が開発したライブラリということで日本人が採用することも多いが，その自由度から時系列データを扱う際に採用される傾向がある．

3. 関連研究

3.1 メモリ消費電力に基づく CPU 周波数の動的制御

三輪らは，アプリケーションが CPU とメモリどちらに依存しているかをメモリ消費電力から判断し，適した CPU 周波数を設定することで省電力を実現するという手法を提案している [12]．周波数と実行時間から算出したアプリケーションの CPU 周波数依存度とメモリ消費電力が高い相関関係があることを実験により確かめ，メモリ消費電力が小さい場合は CPU 依存型，メモリ消費電力が大きい場合はメモリ依存型のアプリケーションとした．そしてメモリ消費電力が小さい時は CPU 周波数を高く，メモリ消費電力が大きい時は CPU 周波数を低く設定することでアプリケーションの性能低下を抑えつつ消費電力量を削減している．実際にメモリ依存型のアプリケーションに対して周波数制御を行ったところ，3%の性能低下率に対し 9%の消費電力量の削減を達成した．

3.2 Deep Q-Network の開発

Google DeepMind 社の Mnih らは DQN(Deep Q-Network) いう深層強化学習手法を提案し，TV ゲームプレイ，つまりゲーミングの AI で高い成果を出している．[3] 評価には Atari2600 で動作する複数のゲームのセットを用いている．これまでのゲーミングではゲームごとに細かいデータの入力を必要としたりどう行動するかを人間の手で設定する必要があった．しかし DQN ではゲーム画面

だけを入力として，行動の方策を全く決めてない状態からランダムに動かし学習するという違いがある．それにより様々なゲームに対して同じアルゴリズムを使うことを可能にしている．実験では複雑なゲームではあまり良い成果が出なかったものの，単純なゲームでは人間よりも高いスコアが出るという結果となった．

また，同じく Google DeepMind 社の David Silver らは DQN を用いた囲碁の AI である AlphaGo を開発している．[13]AlphaGo は初期の学習は棋譜を用いて教師あり学習を進め，その後 DQN を用いて自分自身と対局を行い学習を行う．囲碁の AI は AlphaGo 以前はハンデとして石を 4 子以上は置かないとプロ棋士には勝てないという状況だったが，AlphaGo は世界トップクラスのプロ棋士にハンデ無しで勝利するという驚くべき結果となり DQN の有効性を示した．

4. 提案手法と実装

4.1 計算機制御への DQN の利用

DQN でのゲーミングは計算機制御による電力の最適化と類似点があると考えられる．例えば，時系列データの入力，一定時間ごとの行動決定，最終的な報酬の最大化，0 からの学習であることなどである．したがって DQN が適していると考え DQN を計算機制御に対応させる．DQN によるゲーミングでは入力される状態はゲーム画面（数フレーム間）と直前に押したボタン，行動はボタン入力，報酬がスコアであるのに対し，計算機制御では状態はパフォーマンスカウンタなどのデータ（数ステップ間），行動は周波数の増減，報酬は電力に対してのパフォーマンスカウンタの値の増減とする．同じ電力あたりのパフォーマンスカウンタの増加量が大きくなると最終的な消費エネルギーが少なくなると考え報酬計算に採用する．このようにデータを設定することで，DQN での計算機制御による電力最適化手法を提案する．

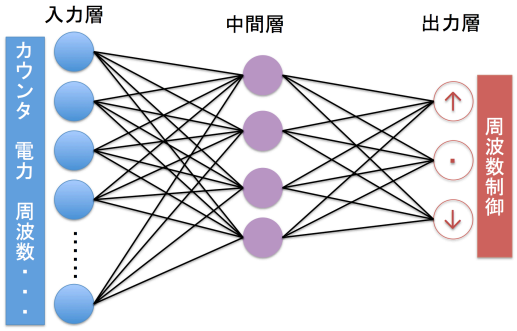

図 2 計算機制御でのニューラルネットワーク

4.2 装置の仕組み

本研究で実装した装置は二つのプログラムで構成されている．一つはアプリケーションを実行し電力やパフォーマンスカウンタなど環境の状態を観測する環境プログラムであり，もう一つは DQN のアルゴリズムを用いて学習を行い実際に周波数制御をするエージェントプログラムである．環境プログラムからエージェントプログラムが状態を受け取り，その値から学習をしながら行動決定を行うという流れを繰り返す．

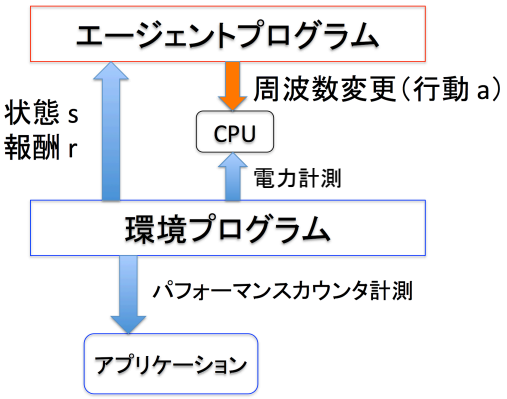

図 3 装置の全体像

4.3 環境プログラム

環境プログラムはまず計測したいアプリケーションを実行し，その間の時間と電力，パフォーマンスカウンタを取得する．環境プログラムは図 4 のような流れで実行される．

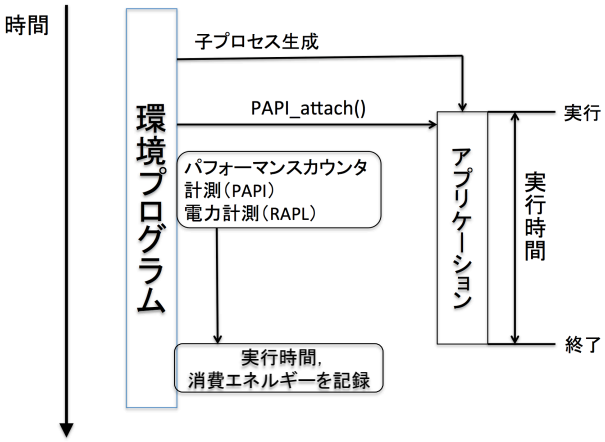

図 4 環境プログラム

4.3.1 アプリケーションの実行

環境プログラムの子プロセスを作成しそのプロセスで exec 関数を用いて指定した実行ファイルを動かす．アプリケーション実行中のプロセス ID と開始時間を記憶しておくことで，データ計測が可能になる．

4.3.2 消費電力の測定方法

消費電力の計測には RAPL インターフェースを用いる。RAPL では各 CPU のチップ全体、チップ上のコア部分、メモリの消費電力を調べることができる。今回は CPU を 2 つ搭載のマシンで計測するので表 4.3.2 の 6 種類の消費エネルギーが計測できる。この電力の値とアプリ実行からの経過時間，後に示すパフォーマンスカウンタの値を等間隔で同時に取得してそれを状態とする．今回は 0.1 秒間隔とした．

表 1 RAPL を用いて計測できる消費電力

cpu1pkgenergy	cpu1 のチップ全体の電力
cpu1pp0energy	cpu1 のコア部分の電力
cpu1dramenergy	cpu1 での DRAM の電力
cpu2pkgenergy	cpu2 のチップ全体の電力
cpu2pp0energy	cpu2 のコア部分の電力
cpu2dramenergy	cpu2 での DRAM の電力

4.3.3 パフォーマンスカウンタの測定方法

PAPI によって与えられるパフォーマンスカウンタを計測する。得られるパフォーマンスカウンタは papi_avail コマンドで確認でき，一度に計測できるカウンタの組み合わせは制限があるので、papi_event_chooser コマンドで計測可能な組み合わせを調べておく．また，今回の実験では 1 回のアプリケーション実行中の状態として常に取得し続ける必要があり，一度に計測できるだけのカウンタしか実際には扱うことができない．そこで，今回は筆者の過去の研究 [14] から電力と関係性が深いと思われる表 2 のカウンタを用いる．

表 2 環境プログラムで取得するパフォーマンスカウンタ

PAPI_L2_DCA	Level 2 data cache accesses
PAPI_L2_TCM	Level 2 cache misses
PAPI_TOT_INS	Instructions completed
PAPI_TOT_CYC	Total cycles
PAPI_STL_ICY	Cycles with no instruction issue

カウンタの取得には PAPI により与えられる各関数を用いる．取得したいカウンタはアプリケーションを実行しているプロセスであるがそのままではこの計測プログラムのプロセス自身のカウンタを取得してしまうので，PAPI_attach 関数と最初に作成した子プロセスのプロセス ID を用いてセットする．これでアプリケーション実行時の状態を観測することができる．このカウンタの取得方法ではアプリケーションのファイルを PAPI 関数を書き加えたりすることはなく，実行ファイルがあればそのアプリケーションに対してデータ取得ができるので様々なアプリケーションを用いて実験することが可能である．

4.4 エージェントプログラム

エージェントプログラムは DQN による学習と行動の実行，つまり周波数の制御を行う．エージェントプログラムは図 5 のような流れで進んで行く．

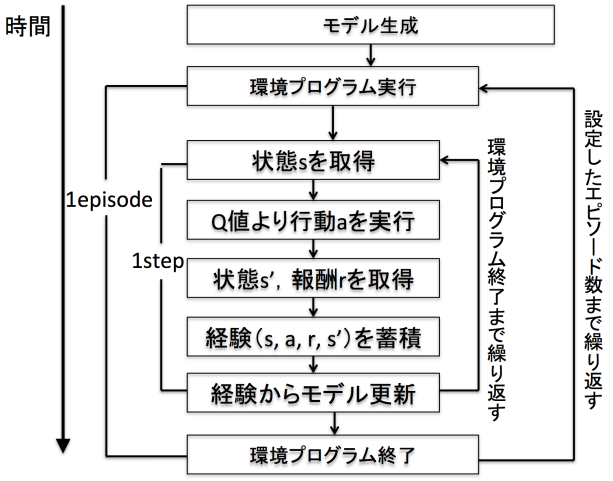

図 5 エージェントプログラム

4.4.1 DQN での学習

python のライブラリである Chainer を用いて学習部分を実装する．今回は入力として扱うデータの種類が少ないことや，学習時に消費する電力が大きいことなどから 3 層の簡単なニューラルネットワークによるモデルを使用する．まずは計算モデルとしてニューラルネットワークを構築するクラスを作成する．このクラスでは各層を繋ぐ Linear 関数 $f(x) = Wx + b$ を定義する．W は重みで b はバイアスでありその値を保持し，学習を進めてより適した値となるようにモデルを更新していく．中間層で用いる活性化関数には ReLU(Rectified Linear Unit) とする．ReLU は

$$f(x) = max(0, x) \tag{5}$$

という式で表される関数である．モデルの最適化を担う optmizer には Chainer で与えられる RMSpropGraves というアルゴリズムを用いる．RMSpropGraves は Alex Graves により提案された特殊な RMSProp である．[15] また，DQN のアルゴリズムでは Target Q-Network を採用する必要があるためモデルを複製して一時固定しておくターゲットモデルも作成する．このターゲットモデルは一定 step 数ごとにモデルをコピーするとして，今回の実験ではコピーの頻度は 1000step とした．

実際に状態 s_t を受け取り行動を決定するには ϵ-greedy 法を用いる．ただ Q 値が高い行動を返すわけではなく，ある程度のランダム性を持って行動を選ぶ．今回の実験では行動は 3 種類であり，現状維持=0，周波数を上げる=1，周波数を下げる=2 とそれぞれの行動に番号付けをする．ランダムに行動するか Q 値が高い行動を選ぶかは ϵ の確率に

準拠する．ϵ の初期値は 1 として step ごとに 10^{-6} ずつ減らし，最終的には 0.1 で固定する．0 から 1 の間で乱数を受け取り，ϵ 未満なら 0,1,2 からランダム，ϵ 以上ならば Q 値が最も高い行動の番号を返す．

次にメインの学習部分を担当する forward 関数について記す．forward 関数では状態 s_t，s_{t+1}，行動 a_t，報酬 r_t，終了フラグを複数回分まとめて受け取り，損失関数 loss を返す．まず状態 s_t からそれぞれの行動に対しての Q 値をまとめて $Q(s_t, a)$ を受け取る．実際に実行した行動 a_t として，

$$Q(s_t, a_t) \leftarrow r_t + \gamma \max_a Q(s_{t+1}, a) \tag{6}$$

と代入する．γ は割引率と呼ばれるパラメータであり今回は 0.99 とする．また，終了フラグによりアプリケーションが終了したタイミングか否かを判断し，もし終了している場合は

$$Q(s_t, a_t) \leftarrow r_t \tag{7}$$

とする．$Q(s_{t+1}, a)$ は通常のモデルではなくターゲットモデルを使用する．その後元々の $Q(s, a)$ と代入後の値との平均二乗誤差を求めて，損失関数 loss の値として返す．

DQN ではいくつかのデータを経験として保存し，その後ランダムに抽出して学習を行う Experience Replay という手法を使用している．保持するデータ数は 1000 組としてそれを越えると古いものから上書きすることにする．学習を始めてすぐは初期探索期間としてデータを取得するだけにし，100step を越えると毎 step モデルの更新を行う．更新に扱うデータは，保存した 1000 個のデータセットからランダムに 32 個抽出して決める．その後

- optimizer の初期化
- forward 関数を用いて損失関数 loss の算出
- loss の逆伝搬
- optimizer によるモデルの更新

という流れによって学習を進める．

4.4.2 データの取得

まず現在の状態 s_t を受け取る．この値は，
(経過時間，電力データ，単位時間あたりのパフォーマンスカウンタ，周波数)
によって構成される現在の状態に加え，一つ前の状態，二つ前の状態を全て合わせて並べたリストとなる．そして方策に従い行動を実行し，その後の状態 s_{t+1} と選んだ行動 a_t を取得する．報酬 r_t は状態の変化から算出する．取得したデータは Experience replay のために蓄積していく．

4.4.3 報酬の算出

今回はアプリケーション実行後の合計の消費エネルギーが少なくなるようにすることを目指すため，カウンタの単位時間当たりの変化量を電力で割った値の変化量を報酬に用いる．命令数や計算などに関するカウンタは実行速度に限らず同じアプリケーションなら終了時には同じ値になるはずなので，そのようなカウンタは実行速度に関する評価に扱うことができる．同じ電力でカウンタの増加量が大きくなると最終的に少ないエネルギーになると考えそのように設定する．報酬の値は DQN のテクニックである報酬のクリッピングを参考にして，各カウンタに対して一定量以上の増加で+1, 減少で-1，その他で 0 とする．また，今回の実験の前に，周波数を常時最大と常時最小にしてアプリケーションを実行し最終的なカウンタの値を比較したところ，PAPI_STL_ICY のカウンタだけ値に大幅な変化があったがその他の 4 つはほぼ同じ値になった．そこで表 3 の 4 つのカウンタは実行速度の指標になると判断し報酬計算に用いる．

表 3 報酬計算に使用するパフォーマンスカウンタ

PAPI_L2_DCA	Level 2 data cache accesses
PAPI_L2_TCM	Level 2 cache misses
PAPI_TOT_INS	Instructions completed
PAPI_TOT_CYC	Total cycles

4.4.4 周波数の変更

行動として，現状維持，周波数の増加，周波数の減少の 3 つを実行する．今回周波数の制御には Linux のパッケージである cpupower コマンドを扱う．
sudo cpupower -g userspace
で動作周波数の変更が可能になり，
sudo cpupower -f (frequency)
で周波数の変更が行うことができる．今回の環境では 1.2GHz から 2.1GHz の間で 0.1GHz ずつの設定が行うことができる．アプリケーション実行してすぐの設定は 1.7GHz として，増加の行動で+0.1Hz，減少の行動で-0.1Hz とする．この設定した周波数の値は変数として保持しておき，状態 s_t の取得時にも扱う．

4.4.5 全体の制御

以上の機能を含めた全体のプログラムの制御をエージェントプログラム上で行う．まず計算モデルを作成し，環境プログラムを実行する．環境プログラム内でアプリケーションの実行が行われる．その後アプリケーションが終わるまで学習と行動実行を毎 step(0.1 秒毎) 行い，終了後また環境プログラムの実行をするという繰り返しを行う．この 1 回の流れを episode と呼ぶことにする．一定の episode 回数ごとに学習結果の確認や，モデルの保存を行う．保存したモデルのテスト時には $\epsilon = 0.05$ とし，ほぼ Q 値に従って行動を選ぶ．そして事前に決めた合計 episode 回数を終えたらガバナーを元に戻し終了する．

5. 評価

5.1 実験環境

本研究の評価には，東京工業大学に設置されているスー

パーコンピュータ TSUBAME-KFC/DL に付随する，比較検証用の空冷ノードを用いた．ノードの性能は表 4 に示す通りである．

表 4　実験環境

CPU	Intel Xeon CPU E5-2620 v2(2.1GHz) × 2
物理コア数	6 × 2
カーネル	3.10.0-514.2.2.el7.x86_64
OS	CentOS7.3.1611
PAPI	5.5.1
Python	3.4
Chainer	1.19.0

5.2　実験

実装した装置により，アプリケーション実行中の周波数制御方策を DQN を用いて学習を行った．1 回のデータ蓄積や行動実行をするサイクルを step として，1step=0.1 秒と設定した．アプリケーションの 1 回の実行を 1 episode として，100 episode ごとにモデルの評価を行った．また比較のために周波数を 2.1 GHz で固定してアプリケーションを実行した際の実行時間と消費エネルギーの計測も行った．

5.2.1　mat_mult

行列積演算プログラム mat_mult を用いて評価を行った．このプログラムは 1024 × 512 のサイズの行列と 512 × 512 のサイズの行列の積を全てのループパターンで演算を行う．

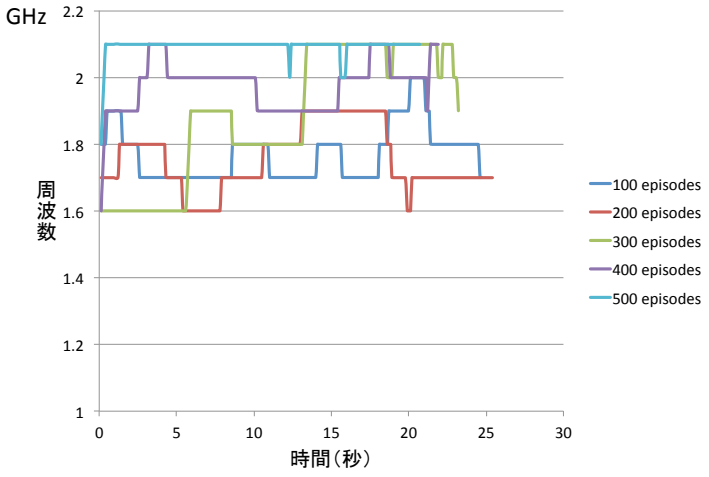

図 6　mat_mult に対しての各モデルの周波数制御

その結果，図 6 で示されるように学習を始めて徐々に高い周波数で動作するようになり，その後周波数が最大となるようなモデルとなった．消費エネルギーは図 7 のように変化し最終的に 2.1GHz で固定した時と同じ値となった．他にもいくつかのアプリケーションで実験したが，どれも周波数最大で動作させると消費エネルギーが最小になると思われ同様の結果となった．

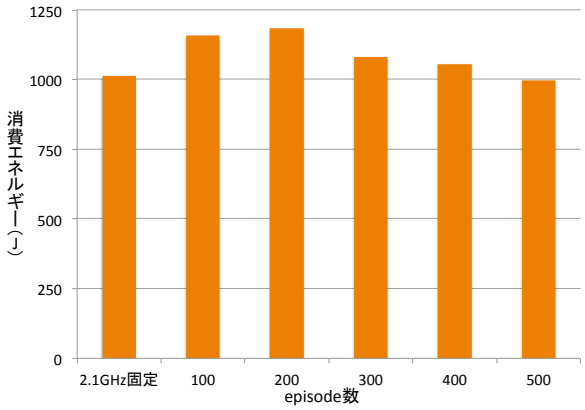

図 7　mat_mult に対しての各モデルの実行結果

5.2.2　mat_mult_sleep

行列積演算プログラム mat_mult の後に 20 秒間 sleep 関数で停止するプログラム mat_mult_sleep を作成して実験を行った．

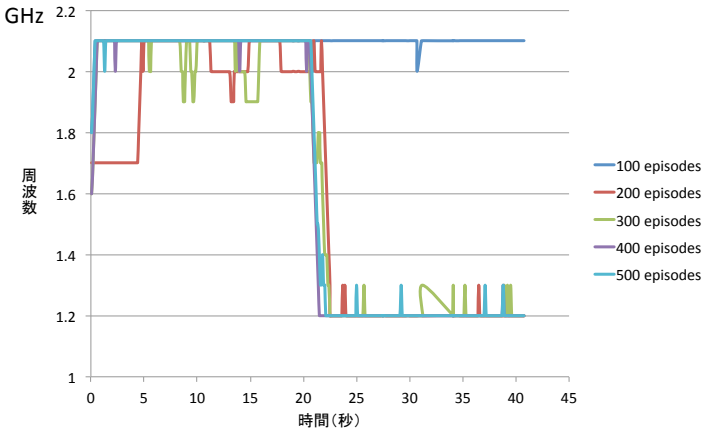

図 8　mat_mult_sleep に対しての各モデルの周波数制御

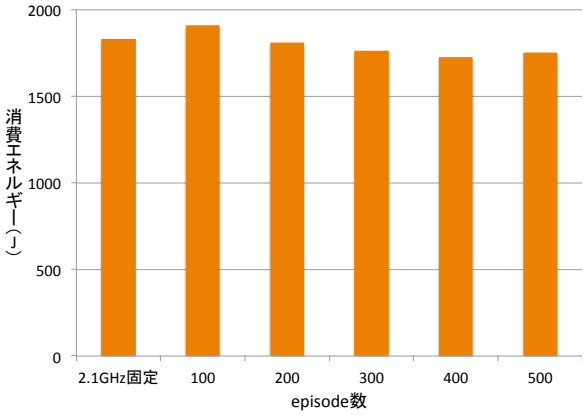

図 9　mat_mult_sleep に対しての各モデルの実行結果

周波数は図 8 のように最初は mat_mult と同様に最大で

動作したが学習が進むにつれて後半で周波数が下がるようになり，400,500 episode 後のモデルでは最初に周波数最大で安定し，後半の 20 秒間，つまり sleep 関数に入ってすぐ周波数を最小まで下げるようになった．また，図 9 で示しているように消費エネルギーも学習が進むにつれて下がり，最も低い 400 episode では 2.1 GHz で固定した時より 5.6%減少した．

6. まとめと今後の課題

本研究では，Deep Q-Network という深層強化学習手法を用いて周波数を制御して電力を最適化する装置を Chainer を用いて実装した．実験では学習回数ごとに周波数の動きや実行時間，消費電力などを評価し，アプリケーションの動作に適した周波数を設定して省電力ができることを示した．また，最終的な消費エネルギーを最小にすることを目指して報酬設定をすると，常に周波数最大が適しているアプリケーションが多く同じような動作をした．途中で待機時間を入れるようなアプリケーションでは待機時間中周波数最小になるように学習できたことも示した．

今後の課題としては，今回の実験では扱うアプリケーションの幅が少なく実装した装置の効果がわかりにくくなってしまったので，周波数最大が最適ではないアプリケーションを探し実験を行う．例えばメモリバウンドなアプリケーションや，MPI を用いるアプリケーションなどである．報酬については単純に最終的な消費エネルギーを減らすことだけを考えて設定したが，ある程度の性能低下を許容するような設定など色々な報酬の設定方法が考えられる．また，今回は学習にはアプリケーション 1 つずつ電力の最適化を行ったが，複数のアプリケーションについて学習を行い，学習したモデルを他の未知なアプリケーションに使用するとどうなるかといった実験を行い実用性を確かめる．

謝辞 本研究の一部は，JST，CREST の支援を受けたものである．

参考文献

[1] 浅井雅司, 池田佳路, 佐々木広, 近藤正章, 中村宏. 統計処理に基づくコンパイラ協調型 dvfs 手法. 情報処理学会研究報告, 2006-ARC-166, Vol. 2006, No. 8, (2006).
[2] 都筑一希, 遠藤敏夫. Cpu・gpu 混載ノードにおける電力・性能モデルを用いたオンラインパワーキャッピング手法. 情報処理学会研究報告, Vol. 2015-HPC-152, No. 2, pp. 1 – 7, 2015.
[3] Mnih Volodymyr, Kavukcuoglu Koray, Silver David, Rusu Andrei A., Veness Joel, Bellemare Marc G., Graves Alex, Riedmiller Martin, Fidjeland Andreas K., Ostrovski Georg, Petersen Stig, Beattie Charles, Sadik Amir, Antonoglou Ioannis, King Helen, Kumaran Dharshan, Wierstra Daan, Legg Shane, and Hassabis Demis. Human-level control through deep reinforcement learning. *Nature*, Vol. 518, No. 7540, pp. 529–533, 02 2015.
[4] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. *CoRR*, Vol. abs/1603.08511, , 2016.
[5] W. Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *CoRR*, Vol. abs/1610.05256, , 2016.
[6] Jason Lee, Kyunghyun Cho, and Thomas Hofmann. Fully character-level neural machine translation without explicit segmentation. *CoRR*, Vol. abs/1610.03017, , 2016.
[7] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, Fernando Mujica, Adam Coates, and Andrew Y. Ng. An empirical evaluation of deep learning on highway driving. *CoRR*, Vol. abs/1504.01716, , 2015.
[8] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, Vol. abs/1604.07316, , 2016.
[9] Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. *CoRR*, Vol. abs/1609.05521, , 2016.
[10] カオタン, 和田康孝, 近藤正章, 本多弘樹. RAPL インタフェースを用いた HPC システムの消費電力モデリングと電力評価. 情報処理学会研究報告, Vol. 2013-HPC-141, No. 20, pp. 1 – 8, 2013.
[11] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.
[12] 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（ACS）, Vol. 5, No. 5, pp. 1–9, 2012.
[13] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, and Sander Dieleman. Mastering the game of go with deep neural networks and tree search. *Nature*, Vol. 529, pp. 484–503, 2016.
[14] 寺西賢人, 野村哲弘, 松岡聡. ノード内同時実行ジョブにおけるパフォーマンスカウンタによるプロセス毎消費電力のモデル化. 情報処理学会研究報告, Vol. 2015-HPC-150, No. 28, pp. 1 – 6, 2015.
[15] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, Vol. abs/1308.0850, , 2013.