社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

リソース分離型マイクロデータセンターにおけるリモートメモリ活用手法の性能評価と考察

生駒　昭繁† 　大下　裕一† 　村田　正幸†

† 大阪大学 大学院情報科学研究科　〒565–0871 大阪府吹田市山田丘 1–5
E-mail: †{a-ikoma,y-ohsita,murata}@ist.osaka-u.ac.jp

あらまし　近年、リアルタイム性の高い処理を実行するために、ユーザにより近いエッジに小規模のデータセンターを置き、そこで様々な処理を行えるようにするマイクロデータセンターが提案されている。マイクロデータセンターは、大規模なデータセンターと比べて保有しているリソースの量は限られているため、無駄のないリソースの使用が重要となる。そこで、サーバ単位で構成されていたマイクロデータセンターをリソースごとに分解し、リソース単位で構成するリソース分離型マイクロデータセンターが提案されている。これによって、リソース単位での最適化による効率的なリソース利用や柔軟なリソース管理が期待される。しかし、リソース間の通信遅延による性能低下が発生してしまう。そこで、本稿では性能低下の大きな原因となるリモートメモリと CPU との通信遅延を導入し、リソース分離型マイクロデータセンターでの利用が想定される処理に対して、性能低下率と帯域幅、レイテンシとの関係を求めた。その結果、帯域幅は 40Gbps 以上にしても大きな性能の向上は見込めない傾向があることが明らかになった。また、レイテンシによる影響の受けやすさは処理ごとに様々であり、それらに対応できるリモートメモリの配置や実行する処理ごとにリモートメモリの適切な資源配分を行うアルゴリズムが必要であるとわかった。
キーワード　マイクロデータセンター、リモートメモリ、レイテンシ、帯域幅、リソース分離、行列計算、画像分類

Performance evaluation of remote memory utilization method in disaggregated micro data center

Akishige IKOMA†, Yuichi OHSITA†, and Masayuki MURATA†

† Graduate School of Information Science and Technology, Osaka University
Yamadaoka 1–5, Suita, Osaka, 565–0871 Japan
E-mail: †{a-ikoma,y-ohsita,murata}@ist.osaka-u.ac.jp

Abstract　In recent years, a micro data center has been proposed to execute processing that requires real-time properties. Micro data centers are small data centers located close to users. Micro data centers have a limited amount of resources compared to traditional data centers. Therefore, it is important to use resources efficiently. A disaggregated micro data center has been proposed as a solution. The disaggregated micro data center is a data center constructed by connecting stand-alone resources via a network. These enable efficient resource utilization and flexible resource management by optimizing each resource. However, performance degradation occurs due to communication delay between resources. In this paper, we introduced the communication delay between the remote memory and the CPU, which is a major cause of performance degradation and evaluated the relationship between performance degradation, bandwidth and latency for jobs expected to be used in disaggregated micro data centers. The evaluation results revealed that even if the bandwidth is 40 Gbps or more, there is a tendency that large performance improvement cannot be expected. In addition, we found that the impact of increased latency on performance varies from job to job. Therefore, it is necessary to allocate a remote memory in consideration of latency and create the algorithm that allocates appropriate resources for remote memory for each job to be executed.
Key words　Micro data center, remote memory, latency, bandwidth, Disaggregation, Matrix calculation, Image classification

1. はじめに

計算機および計算機によって構成されるネットワークやインターネットの発展は著しく、ユーザはクラウドに接続することでユーザ端末では実行困難である大規模な計算や、大規模なストレージのような様々なサービスを手軽に利用することができるようになった。しかし、クラウドでの処理は端末からデータセンターに接続するまでにかかってしまう通信遅延や、処理したいデータをインターネットを介してクラウドに上げることへのセキュリティ上の問題、IoT の発達によって爆発的に増えているデータとクラウドの通信によるトラヒック量の増加という問題がある。これらの問題に対応するために、ユーザからより近い場所やデータの収集装置の近くにリソースを配置し、処理を行えるようにするエッジコンピューティングの研究開発が進められている。これによって、処理の低遅延化やトラヒック量の増加への対応が期待されている [1]。その中で、ユーザからより近い場所やデータの収集装置の近くにデータセンター自体を置き、そこでデータの処理を行うというマイクロデータセンターが考えられている [2]。しかし、マイクロデータセンターは、大規模なデータセンターと比べて保有しているリソースの量は限られているため、柔軟なリソース管理による無駄のないリソースの使用が重要となる。これらの事象を実現するための方法として、リソース分離型のデータセンターの技術が考えられる。リソース分離型のデータセンターは、図 1 に示すようにデータセンター内でサーバ単位で集約されていた CPU やメモリのようなリソースをリソースの機能ごとにリソースブレードとして分解し、データセンター内の構成をリソースブレードごとに分離させたものである [3]。リソース分離型のデータセンターは、実行処理ごとに分離されたリソースそれぞれを独立して割り当てることができるため、リソースの使用の最適化を行うことが可能である。これにより、データセンター内のリソース使用率を向上させることができる [4]。さらに、リソース単位で独立しているため、リソースの更新を他のリソースとのつながりを気にせずに自由に行うこともできる。

しかし、リソース単位でデータセンターを構成するため、リソースブレード間でのデータの転送時に発生する遅延による性能低下の影響が問題となっている。そこで、性能の低下を抑えるためのネットワークについての研究が進められており、RDMA の利用による高速化 [5] や光インターコネクトや光スイッチのような光通信技術の採用 [6] が提案されている。しかし、これらの研究は分散処理のような大規模な並列演算処理を対象としており、マイクロデータセンターで動作させるような画像認識等のアプリケーションを想定した検討は行われていない。さらに、これらの研究で対象とされていた大規模なデータセンターと異なり、マイクロデータセンターではリソース間の距離が小さく、発生しうる遅延も小さいと考えられる。

そこで、本稿では、マイクロデータセンターに焦点を当て、マイクロデータセンターにおいてリソース分離型の構成を構築する際の要件について評価を行う。CPU とリソース分離を行ったメモリであるリモートメモリ間の通信については、要求されるレイテンシと帯域幅の要件が厳しく、性能低下の直接的な原因となっている [5]。リモートメモリはメモリの使用率を上げ、より効率的なメモリの使用を可能にする技術である [7] が、この利点を最大限に生かすためには、リモートメモリの活用による性能低下を抑える必要がある。そこで、マイクロデータセンターでの実行が見込まれる処理において CPU とリモートメモリの通信による遅延が性能にどれほど影響するのかを調査し、実際にリソース分離型のマイクロデータセンターを利用していく際の帯域幅とレイテンシと遅延の影響との関係、実際に利用していく際のリソースの配置やリモートメモリの分配方法について考察する。

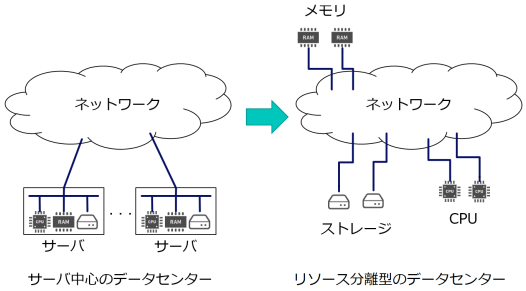


図 1: サーバ中心のデータセンターとリソース分離型のデータセンター

本稿の構成は以下の通りである。2. 章では本稿で想定しているリソース分離型のマイクロデータセンターについて説明する。3. 章でリモートメモリの活用手法における性能評価と考察を行う。最後に、4. 章でまとめと今後の課題について述べる。

2. リソース分離型のマイクロデータセンター

2.1 リソース分離型のマイクロデータセンターの構造

リソース分離型マイクロデータセンターは、リソース同士が密接につながれ構成されていたマイクロデータセンターをリソースごとに分離させて構成したものである。ここで、リソースとは、CPU、ストレージ、メモリのような計算機の構成要素を指している。各リソースはネットワークによって接続されていて、相互に通信することが可能である。また、リソース単位でデータセンターを構成するため、異なる場所にあるリソースを自身のマイクロデータセンターで行う処理に割り当てることも可能である。リソース同士はネットワークを通してつながれており、各リソースからほかのリソースへの通信はネットワークを通して行われる。このとき、リソース同士の距離が遠すぎるとリソース間の通信による遅延による影響が大きくなるので、リソース同士は一定の範囲内に配置される。それぞれのリソースは分離しているため、どのリソースであるか関係なく様々なリソースに対して自分の資源を割り当てていくことが可能となっている。また、リソース同士はネットワークでつながれているものの、リソース同士が統合しているわけではないため、リソースの取り換えや新しいリソースへの変更を全体の構成を設計し直すことなく行うことが可能となり、導入のコストや管理コストを削減できると考えられる。

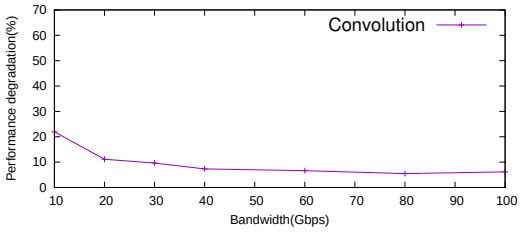


図 2: 行列計算の帯域幅による性能低下率

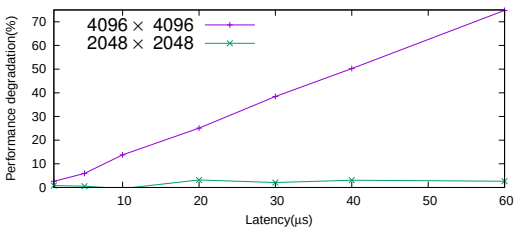


図 3: 行列計算のレイテンシによる性能低下率

2.2 リモートメモリ

リモートメモリとは、ネットワークを経由してアクセス可能なメモリである。リモートメモリは複数のリソースによって共有し、各リソースの処理ごとに必要なサイズのリモートメモリを割り当てるなど、柔軟なメモリ割当が可能となる。サーバ中心のデータセンターでは、サーバに取り付けられたメモリ量に反して実際に使用するメモリ量は少なく、メモリ使用率が低くなる場合があるという問題がある。リソース分離型のマイクロデータセンターで用いられるリモートメモリは、様々な処理に対して必要な分を割り当てていくため、効率的なメモリの利用が可能となる。さらに、実行される処理の増加に対しても、サーバ中心のデータセンターのようにサーバ単位で考えていくのではなく、リモートメモリの容量のみを考えて割り当てていくことができるため、より柔軟な対応が可能となる。

3. 性能の評価

3.1 評価環境

性能評価のために、リモートメモリと CPU との通信を再現した評価環境を実装する。CPU とリモートメモリとの通信のシミュレーションのために、文献 [5] で用いられた方法を採用した。この方法では、CPU とメモリとのやり取りを、CPU に取り付けられたローカルメモリと CPU からは離れたところに配置されたリモートメモリとのやり取りに置き換えることで CPU とメモリとのやり取りを再現する。このとき、ローカルメモリは計算機を動作させるのに最低限必要なキャッシュメモリとして働き、本稿では容量を 200MB に固定する。また、リモートメモリとローカルメモリのデータのやり取りはページ単位で行われる。

ローカルメモリはページフォールト発生時にリモートメモリとして割り当てたメモリ領域とデータのやり取りを行うので、このときにネットワークによる遅延を挿入することでリモートメモリとのやり取りを再現する。ネットワークによる遅延は、レイテンシと転送遅延の和である。これらは、パラメータとして設定されたレイテンシ (μs) と帯域幅 (Gbps) をもとに導出され、レイテンシはリモートメモリからの応答を待つまでの時間を、転送遅延は転送するデータサイズを帯域幅で割ったもの表している。本実験環境では、ページサイズは 4KB に設定し、データの転送は 1 ページずつ行われる。そのため、4KB 分のデータを転送する時間を転送遅延として挿入する。

また、本稿では、CPU として Intel(R) Xeon(R) CPU E5-2687W 0 @ 3.10GHz を、メモリとして DDR3 SDRAM 持つ計算機を用いて、CentOS7.7 上で評価を行った。なお、本計算機の CPU のバススピードは 8GT/s であり、使用したメモリの転送速度は 12.8GB/s である。

3.2 評価対象

本稿の性能評価の評価対象として、行列計算とその行列計算の応用であり、エッジでの利用が見込まれる機械学習による画像分類の処理を採用した。これらの評価対象に対して、実行時間に対する性能低下率を評価する。

3.2.1 行列計算

性能の評価のために、比較的大きい行列に対しての畳み込み計算を行う。処理内容としては、ランダムな値を格納した行列に対しての 3 × 3 の行列を用いた畳み込み計算であり、4096 × 4096 と 2048 × 2048 の二つの行列に対して評価を行う。この処理を 20 回繰り返したものを性能評価用のプログラムとする。

3.2.2 画像分類

本稿では、リソース分離型のマイクロデータセンターをエッジで使用することを想定している。そこで、3.2.1 章で説明した行列計算処理を利用し、エッジでの利用も想定される畳み込みニューラルネットワークのモデルを用いた画像分類の処理についても性能評価を行う。画像分類は、MobileNetV2 [10]、ResNet [11]、NASNet [12]、Inception-v3 [13] のそれぞれの機械学習モデルを用いて行われる。また、機械学習ライブラリとして Tensorflow [14] と Tensorflow のモバイル、エッジ端末向けライブラリである Tensorflow Lite [15] の二つの機械学習ライブラリを用いる。ここで、使用する機械学習モデルはすべて畳み込みニューラルネットワークのモデルとなっている。また、Tensorflow で動作させるモデルは Keras [16] によって提供されている学習済みのモデルであり、Tensorflow Lite で動作させるモデルは、Keras によって提供されている学習済みモデルを Tensorflow Lite で動作できるように変換したものである。

プログラムとしては、モデルの読み込み、画像の読み込みと変換、読み込んだ画像に対する推論処理を行う。このとき、バッチサイズは 1 として、異なる 100 個の画像に対して推論処理を行っていく。

3.3 行列計算の評価結果

3.3.1 帯域幅による性能低下率の評価

レイテンシを 5μs に固定した状態で、帯域幅を 10Gbps、20Gbps、30Gbps、40Gbps、60Gbps、80Gbps、100Gbps に変更していき、そのときの性能低下率について評価した結果を図 2 に示す。棒グラフは 10 回分の計測した実行時間の中央値に対する性能低下率を示している。

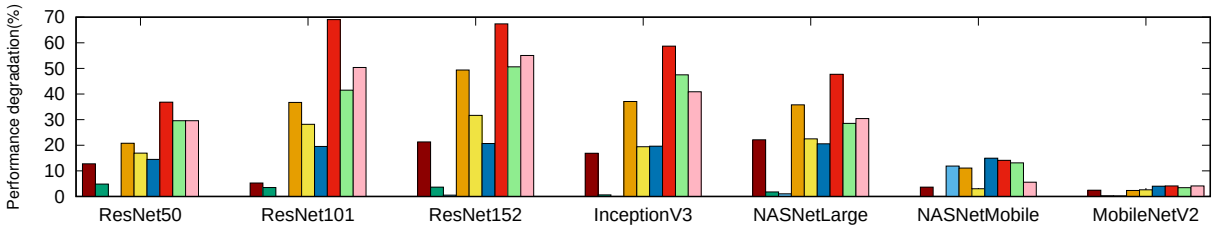


図 4: Tensorflow における評価対象ごとの性能低下率

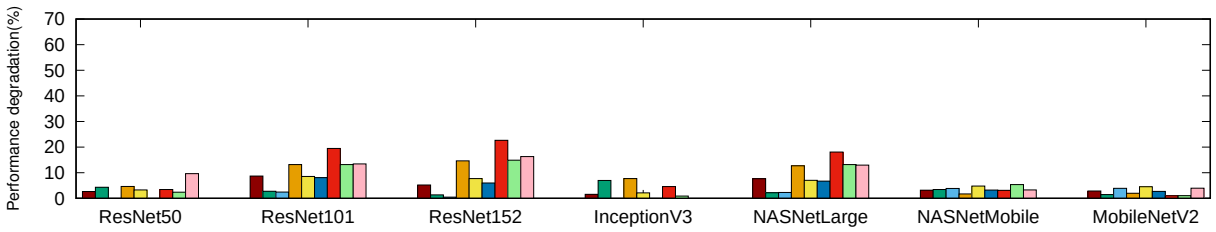


図 5: Tensorflow Lite における評価対象ごとの性能低下率

図 2 から、帯域幅を 40Gbps 以上にしても性能低下率が大きく減少することはないとわかる。そのため、40Gbps 以上の帯域幅があっても、大きな性能向上は見込めないということがいえる。以降の評価では、帯域幅を 40Gbps に固定して行う。

3.3.2 レイテンシによる性能低下率の評価

帯域幅を 40Gbps に固定してレイテンシの変更による性能低下率の調査した。このとき、レイテンシを、遅延の影響をほとんど受けないグループでそれ以外のグループで 1μs、5μs、10μs、20μs、30μs、40μs、60μs に変更していき、そのときの性能低下率について評価した。また、行列のサイズによる違いを調査するために行列のサイズを 2048 × 2048 に変更したものに対してもレイテンシを変更していき、性能低下率を評価した。それらの結果を図 3 に示す。

図 3 に示すように、4096 × 4096 の行列に対する行列は、レイテンシと性能低下率で正の相関がみられ、レイテンシの影響を強く受けていることがわかる。一方で、2048 × 2048 の行列に対する行列計算の性能低下率は 4096 × 4096 の行列に対する行列計算と比較して大幅に性能低下率が低く推移していることがわかる。つまり、行列のサイズによってリモートメモリによる影響は大きく変化するといえる。このことから、行列のサイズによってはリモートメモリを使用しても性能に影響はないといえる。

3.4 画像分類の評価結果

3.4.1 評価対象ごとの性能低下率の評価

レイテンシが 1μs、10μs、20μs で、帯域幅が 10Gbps、40Gbps、100Gbps であるときのすべての場合について、画像分類処理による処理を実行し、遅延を挿入しないで実行したときと比較してどれだけ性能が低下したのかを計測する。また、図 4 と図 5 にその結果を示す。棒グラフは 10 回分の計測した実行時間の中央値に対する性能低下率を示している。

図 4 と図 5 に示すグラフから、Tensorflow の MobileNetV2、NASNetMobile と Tensorflow Lite の ResNet50、InceptionV3、NASNetMobile、MobileNetV2 を用いた推論処理において、性能低下率とレイテンシや帯域幅との関係は見られず、遅延の挿入による影響をあまり受けていないといえる。一方で、それ以外の評価対象では、帯域幅の値が小さいとき、レイテンシの値が大きい時に性能低下率が大きくなっている傾向がみられ、メモリ分解の影響を大きく受けているといえる。

また、帯域幅が 40Gbps のときと 100Gbps のときで性能低下率にそこまで差がないモデルが多く、40Gbps 以上の帯域幅に設定しても大きな性能の向上は見込めない傾向があるといえる。そこで、以降の実験では、帯域幅を 40Gbps に設定してレイテンシによる性能低下率を評価していく。ただし、特に性能低下率の低かった Tensorflow Lite の NASNetMobile と MobileNetV2 に関しては、帯域幅を 10Gbps に設定して評価を行う。

3.4.2 レイテンシによる性能低下率の評価

遅延の影響をほとんど受けないグループの帯域幅を 10Gbps に固定し、それ以外のグループの帯域幅を 40Gbps に固定してレイテンシの変更による性能低下率の調査した。このとき、レイテンシを、帯域幅を 40Gbps に固定したグループで 1μs から 40μs に、帯域幅を 10Gbps に固定したグループで 1μs から 300μs に変更していき、そのときの性能低下率について調査し、それぞれの結果を図 6 から図 7 に調査結果を示す。

図 6 から、実行するモデルによってレイテンシによる影響の受け方は異なっていることが分かる。また、図 7 で示したモデルは、レイテンシが 300μs となっても性能低下率が 6% ほどにとどまっており、ほかのモデルと比べると大幅にレイテンシの影響を受けにくいといえる。

3.5 考　　察

今までの評価結果をもとに、性能低下率と評価対象との関係、リモートメモリの配置について、 リモートメモリの割当 についての考察を行う。

3.5.1 機械学習モデルと性能低下率の関係

図 4 の結果によって、メモリ分解の影響を大きく受けるのは Tensorflow で実行したときの ResNet50、ResNet101、

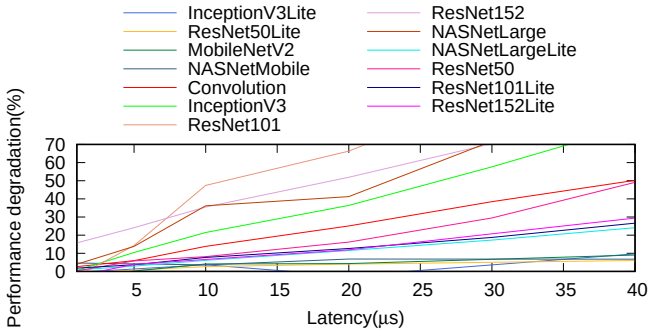


図 6: 帯域幅を 40Gbps に設定したグループのレイテンシによる性能低下率

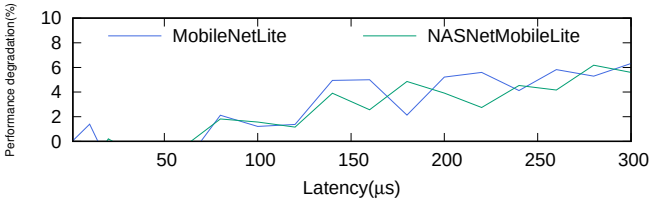
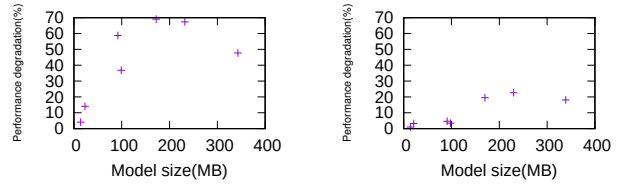


図 7: 帯域幅を 10Gbps に設定したグループのレイテンシによる性能低下率

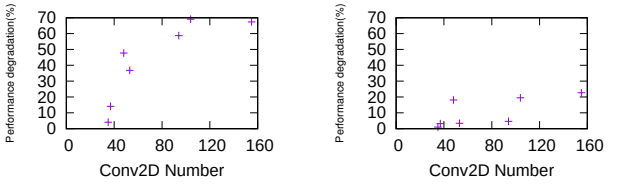


(a) Tensorflow における性能低下率とモデルサイズとの比較　(b) Tensorflow Lite における性能低下率とモデルサイズとの比較

図 8: 性能低下率とモデルサイズとの比較

ResNet152、NASNetLarge、InceptionV3 と Tensorflow Lite で実行したときの ResNet101、ResNet152、NASNetLarge であることが分かった。ここで、図 8 に、Tensorflow と Tensorflow Lite における性能低下率とモデルサイズの比較のためのグラフを示す。性能低下率は、図 4 に示した結果において一番メモリ分解の影響を強く受けるレイテンシが 20μs、帯域幅が 10Gbps のときのものを採用した。

図 8 から、メモリ分解の影響をあまり受けていなかったモデルはモデルサイズが比較的小さいということが分かった。ただし、Tensorflow と Tensorflow Lite で影響を受け始めるモデルサイズは異なり、Tensorflow Lite の場合は 91MB の Inception-v3 まで影響を受けていないのに対し、Tensorflow で影響を受けていないのは 23MB の NASNetMobile 以下のモデルだけである。これは Tensorflow Lite は内部変数を小さくして計算するので、より大きなモデルでもメモリをあまり使用せずに実行できるからだと考えられる。ただし、モデル全体で見たとき、Tensorflow Lite でのモデルのサイズとパフォーマンスの低下率の相関係数は 0.846798899 と正の相関があるといえるが、Tensorflow でのモデルのサイズとパフォーマンスの低下率の相関係数は 0.64934673 であり、正の相関があるものの、そ



(a) Tensorflow における性能低下率と Conv2D の数の比較　(b) Tensorflow Lite における性能低下率と Conv2D の数の比較

図 9: 性能低下率と Conv2D の数の比較

こまで大きい相関があるわけではないといえる。

ここで、3.3 章から畳み込み計算による処理は行列のサイズがある程度大きい場合、リモートメモリの使用による遅延の影響を強く受けていることが分かる。また、本稿で評価対象とした機械学習モデルはすべて畳み込みニューラルネットワークを使用したモデルである。そこで、Tensorflow と Tensorflow Lite における畳み込み計算を行う層の総数と性能低下率との比較を図 9 に示す。また、本稿で使用するモデルでは畳み込み層として Conv2D、DepthwiseConv2D、SeparableConv2D の 3 種類の畳み込み層がある。ただし、DepthwiseConv2D、SeparableConv2D はそれぞれ NASNetLarge、NASNetMobile と MobileNetV2 でしか使用されていないので、比較には Conv2D の数を用いる。

このとき、Tensorflow においては相関係数が 0.830466093 であり、モデルのサイズと比べて大きく正の相関がみられ、畳み込み計算の処理とパフォーマンスの低下には関係があるといえる。ここで、3.3 章から、畳み込み計算はある程度大きいサイズの行列の場合はメモリ分解の影響を大きく受けていることが分かる。また、今回実行したモデルの総パラメータ数は、リモートメモリの影響を受けずらい NASNetMobile と MobileNetV2 を除いて、InceptionV3 の 23851784 が最小である。これは、3.3 章で示した、リモートメモリの影響を受けにくいサイズである 2048 × 2048 の行列のパラメータ数 4194304 よりもはるかに大きい数字であり、畳み込み計算の影響を受けやすいといえる。そのため、パフォーマンス低下を抑えるためには、畳み込み計算の計算量を減らすことも重要となるといえる。

このことから、畳み込みニューラルネットワークにおいては、畳み込み処理の量はメモリの使用量に影響を及ぼしていることが分かる。また、畳み込み層の多いモデルはリモートメモリによる遅延の影響を受けやすいといえる。

3.5.2 リモートメモリの配置

本節では、リモートメモリの配置を考えるにあたり、簡単のため、CPU・メモリ間で発生する遅延は伝搬遅延のみとして考え、どれだけ離れた位置にあるメモリを利用できるのかについて考察を行う。伝搬遅延は 200m あたり 1μs である。例えば、性能低下率を 5% 以内に抑えられる最高のレイテンシはモデルによって、260μs、140μs、20μs、10μs、5μs、1μs となっている。このとき、レイテンシは一方のリソースへデータを送信しその返答が返ってくるまでの時間である。データの処理時間を 1μs ほどに仮定すると、データが往復するのに必要な時間はそ

れぞれ 259μs、139μs、19μs、9μs、4μs、0μs となる。実際にリモートメモリを配置するときはこれらの時間をもとに考えていく。

1μs 以下のレイテンシ要件をもつ処理に関しては、データの往復時間が 0 に近い値でなければならないので、自身が持つローカルメモリやリモートメモリによって処理されるべきであり、他のリソースを使用することはできないといえる。レイテンシ要件が 5μs 以上の処理に関しては、生じる遅延が伝搬遅延のみであるならば、400m 以内の範囲にあるリモートメモリを活用しても性能低下は 5% 以内に抑えられる。同様にして、各レイテンシ要件から活用できるリモートメモリの設置範囲を導出できる。

このように、必要なレイテンシををもとにリモートメモリをどこに配置するかを考える必要がある。

3.5.3 リモートメモリの割当

リモートメモリとのやり取りが多い処理ほど、リモートメモリを用いることによる性能低下が大きい。その反面、リモートメモリとのやり取りが多い処理は必要とするメモリサイズも大きい。そのため、そのような処理には、近くのリモートメモリや自身のメモリを優先的に割り当てる必要がある。一方で、リモートメモリとのやり取りが少ない処理は、遠くのリモートメモリを割り当てても性能低下が少ない。そのため、リモートメモリの割り当てのみならず、処理を実行する割り当ても含め、リモートメモリとのやり取りが多い処理については CPU・リモートメモリが近くになるように割当を行い、その残余資源を用いてそれ以外の処理を行うといった割当が適当であると考えられる。このような割り当てを行うための具体的な割り当てアルゴリズムは、今後の課題である。

4. おわりに

本稿では、リモートメモリを用いたリソース分離型マイクロデータセンターを想定し、リモートメモリはリソース分離型マイクロデータセンターの性能にどれほど影響があるのかについて評価を行った。帯域幅は、40Gbps 以上で大きな性能の向上は見られないということが示された。また、レイテンシによる影響は実行処理によって様々であり、様々な処理に対応できるようなリモートメモリの配置や割当アルゴリズムが必要であるとわかった。

本稿では、リモートメモリの使用を前提としたプログラムは用いていない。そのため、プログラムの書き方や命令のアクセスパターンの工夫次第では、今回の結果よりも性能低下率を低く抑えられる可能性がある。また、性能低下率との関係やデータの圧縮のようなデータの転送方法による工夫とその評価についてが今後の課題として挙げられる。

文　献

[1] 田中 裕之, 高橋 紀之, 川村 龍太郎, “IoT 時代を拓くエッジコンピューティングの研究開発,” *NTT* **技術ジャーナル**, pp. 59–63, Aug. 2015.

[2] K. Bilal, O. Khalid, A. Erbad, and S. U. Khan, “Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers,” *Computer Networks*, pp. 94–120, Jan. 2018.

[3] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, “Network support for resource disaggregation in next-generation datacenters,” in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, pp. 1–7, Nov. 2013.

[4] Y. Cheng, R. Lin, M. De Andrade, L. Wosinska, and J. Chen, “Disaggregated data centers: Challenges and tradeoffs,” *IEEE Communications Magazine*, Mar. 2019.

[5] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, “Network requirements for resource disaggregation,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 249–264, Nov. 2016.

[6] G. Zervas, H. Yuan, A. Saljoghei, Q. Chen, and V. Mishra, “Optically disaggregated data centers with minimal remote memory latency: Technologies, architectures, and resource allocation [invited],” *IEEE/OSA Journal of Optical Communications and Networking*, pp. A270–A285, Feb. 2018.

[7] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, “Efficient memory disaggregation with infiniswap,” in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pp. 649–667, Mar. 2017.

[8] M. Bielski, I. Syrigos, K. Katrinis, D. Syrivelis, A. Reale, D. Theodoropoulos, N. Alachiotis, D. Pnevmatikatos, E. H. Pap, G. Zervas, V. Mishra, A. Saljoghei, A. Rigo, J. F. Zazo, S. Lopez-Buedo, M. Torrents, F. Zyulkyarov, M. Enrico, and O. G. de Dios, “dredbox: Materializing a full-stack rack-scale system prototype of a next-generation disaggregated datacenter,” in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1093–1098, Mar. 2018.

[9] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, “Legoos: A disseminated, distributed OS for hardware resource disaggregation,” in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 69–87, Oct. 2018.

[10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, June 2018.

[11] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016.

[12] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8697–8710, June 2018.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, June 2016.

[14] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: A system for large-scale machine learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

[15] “Deploy machine learning models on mobile and IoT devices.” `https://www.tensorflow.org/lite`, last accessed on 2020-2-8.

[16] “Keras: Deep learning for humans.” `https://github.com/keras-team/keras`, last accessed on 2020-2-8.