一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
VLD2018-84,CPSY2018-94,RECONF2018-58(2019-01)

Android OS向けMPI実行環境における並列処理性能の初期評価

新里　将大† 　杉山　裕紀† 　大津　金光† 　大川　　猛† 　横田　隆史†

† 宇都宮大学大学院工学研究科情報システム科学専攻 〒321-8585 栃木県宇都宮市陽東 7-1-2
E-mail: †{nissato,sugiyama}@virgo.is.utsunomiya-u.ac.jp, ††{kim,ohkawa,yokota}@is.utsunomiya-u.ac.jp

あらまし　昨今, スマートフォンやタブレット端末などのモバイルコンピュータ機器の高性能化が著しく，高性能計算環境のための新しい計算資源として注目されている．我々は，Android OS を搭載したモバイルコンピュータ機器からなるクラスタシステムの研究開発を行っている．先行研究では，MPI を基盤とした並列処理を実現するために，Android OS のスーパーユーザ権限が必要であった．我々は，一般ユーザ権限の範囲で利用可能なクラスタシステムを構築するために，クラスタシステムの並列処理基盤を Android アプリケーションとして構築した．しかし，我々の並列処理環境では，並列プロセスを Android OS のアプリケーション上から起動するため，並列処理実行時のオーバヘッドが生じることが予想される．我々は，ベンチマークプログラムにより我々の構築したシステムと先行研究のシステムとの比較評価を行った．結果として，複数ノードでの MPI 並列処理について，我々の構築したシステムにおいても，従来システムと比較して同程度の性能が得られることが分かった．

キーワード　Android OS，Open MPI，性能評価

Preliminary Evaluation of Parallel Processing Performance on MPI Runtime Environment for Android OS

Masahiro NISSATO†, Hiroki SUGIYAMA†, Kanemitsu OOTSU†,
Takeshi OHKAWA†, and Takashi YOKOTA†

† Graduate School of Engineering, Utsunomiya University Yoto 7-1-2, Utsunomiya, Tochigi, 321-8282, Japan
E-mail: †{nissato,sugiyama}@virgo.is.utsunomiya-u.ac.jp, ††{kim,ohkawa,yokota}@is.utsunomiya-u.ac.jp

Abstract　In recent years, performance of mobile computer devices such as smartphones and tablet computer devices has been greatly improved, and the devices are attracting attention as new computational resources for high performance computation. We are developing a cluster computer system that is composed of the devices running Android OS. In the preceding researches, super-user authority is required to realize a parallel processing environment. In order to realize the cluster system that can operate without super-user authority, we develop a parallel processing environment as Android application program. In the system with our new parallel processing environment, parallel processes are started from Android application. Therefore, Therefore, some overhead might occur on parallel processing. Hence, we clarify the performance of our system as compared to our conventional system by using benchmark programs. Preliminary evaluation results reveal that our system shows comparable parallel processing performance with the conventional system.

Key words　Android OS, Open MPI, performance evaluation

1. はじめに

スマートフォンやタブレットコンピュータなどのモバイル端末の計算性能は，マルチコア CPU や GPU の搭載に加え，メインメモリおよびストレージ容量の増加や，ネットワーク通信性能の高速化などの様々な要因と相まって性能向上した．また，モバイル端末は端末数が急激に増加したことにより，最も一般的に用いられるコンピュータ機器の一つとなった．

これらのモバイル端末の急速な性能向上と端末数の増加から，モバイル端末を高性能計算のための新たな計算資源として利用することが提案された．Busching らは，スマートフォンをノードコンピュータとして用いるクラスタシステムが構築可能であ

ることを示した．Busching らは，モバイル端末の SD カード上に Linux OS をインストールし，Linux OS 上にクラスタシステムを構築した [1]．Hinojos らは，モバイル機器による分散処理を実現するために，モバイル端末を Bluetooth で接続した分散処理環境を構築した [2]．このように，モバイル端末は高性能計算のための新たな計算資源として利用可能である．

我々は，一般に普及している Android OS を搭載した機器（Android 機器）を計算資源として利用するために，Android 機器を計算ノードとして使用するクラスタシステム（Android クラスタ）の研究開発を行っている [3]．本システムでは，MPI（Message Passing Interface）を使用して Android 機器上に並列処理基盤を構築することにより，Android 機器をクラスタシステムの計算ノードとして利用することを可能にした．

我々の研究も含め，MPI により Android 機器のクラスタシステムを構築した先行研究 [1] [3] では，並列処理環境を構築するために Android OS のシステム領域を編集することが必要である．これらの先行研究では，Android OS のシステム領域を編集するために，Android OS のスーパーユーザ権限（root 権限）を取得した．root 権限を取得するためには，Android OS のブートローダをアンロックし，root 権限を取得するためのプログラムを Android OS のシステム領域に書き込む必要がある．しかし，root 権限を取得するための作業は，失敗した場合に Android 機器が利用不可能になる危険性がある．

我々は，一般に普及している Android 機器を Android クラスタの計算ノードとして使用することを想定している．そのためには，スーパーユーザ権限を使用せずに Android 機器上で安全に構築することが可能な，MPI プログラムの実行環境（MPI 実行環境）を実現することが必要である．そこで，我々は，MPI プログラムの実行環境を，一般ユーザ権限で動作する Android アプリケーションとして実装する．これにより，Android アプリケーションを Android 機器にインストールするだけで，スーパーユーザ権限を使用せずに MPI プログラムの実行環境を Android 機器上に構築することを可能とする．

本論文の構成を以下に述べる．第 2. 章では，Android クラスタの計算ノード，ネットワーク構成および並列処理基盤ソフトウェアなどのシステム構成について述べる．第 3. 章では，本論文で提案した手法に基づいて構築した MPI 実行環境を用いて初期評価を行う．第 4. 章では，本論文のまとめを述べる．

2. Android クラスタ

本節では，我々の開発している Android クラスタの計算ノード，ネットワーク環境および並列処理プラットフォームについて述べる．

2.1 システム構成

Android クラスタの構成例を図 1 に示す．本システムでは，Android OS を搭載した機器を計算ノードとする．スマートフォンやタブレットコンピュータのみならず，Raspberry Pi などの Android OS を搭載することが可能な機器を用いてクラスタの処理性能を増強することを想定している．

本システムでは，各機器が備える Wi-Fi や Ethernet などのネットワーク通信機能を計算ノード間の通信に利用する．Android 機器は，ルータやスイッチングハブを介して互いに通信が可能である．また，Wi-Fi アクセスポイント（Wi-Fi AP）を介して無線通信を行うこともできる．我々は，各 Android 機器には DHCP によって固有の IP アドレスが割り振られることを想定しているが，現在は Android クラスタに DHCP サーバ機能を搭載していないため，静的に IP アドレスを設定している．

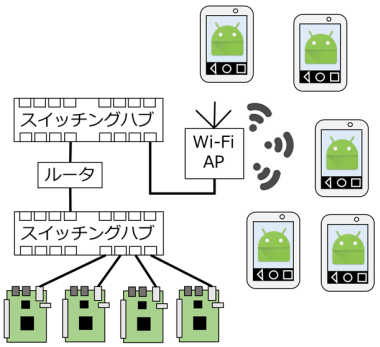

図 1　Android クラスタの構成例

並列実行環境としては，並列分散処理のための標準的な規格である Message Passing Interface (MPI) を利用する．Android クラスタでは，クラスタ上で MPI プログラムを実行することにより並列処理を行う．本クラスタでは，MPI のオープンソースな C 言語実装である Open MPI [9] ライブラリを使用して MPI プログラムの実行環境を構築する．Open MPI のような，既存の C 言語ソースコードを Android 上で利用するためのツールチェーンとして，Android NDK (Android Native Development Kit) [10] がある．Android NDK のクロスコンパイラツールチェーンによって C 言語ソースコードをビルドすることにより，Android OS 上で実行可能な機械語プログラムのファイルを生成できる．我々は，Android NDK を用いて Open MPI のソースコードをビルドする．ただし，Android OS の C ランタイムライブラリ（bionic libc）は Android のミドルウェアのための必要最小限な機能のみを実装しているため，libc のような一般的な C ランタイムライブラリと比較して様々な機能が欠落している．そのため，我々は Android OS 向けに Open MPI の移植を行った．

Open MPI 環境では，クラスタが使用可能な計算ノードを識別するために，計算ノードの IP アドレスなどの情報を記述したテキストファイルである hostfile を使用する．Android クラスタでは，並列処理に使用可能な計算ノードが動的に変化することを想定しているため，hostfile に記述するための情報を自動で収集するデーモンプログラムを開発した．このデーモンプログラムは，各計算ノードの情報を自動的に収集し，hostfile の更新を行う．計算ノードがクラスタに参加する際には，クラスタ内の全てのノードに対して自身の hostfile の情報をブロードキャストする．

2.2 MPI プログラム実行環境

先行研究において，Android 機器上に MPI 実行環境を構築す

るためには，Android 機器の root 権限が必要であった [1] [3]．これは，Android OS のシステム領域に MPI プログラムの実行環境を構築していたからである．

我々は，一般ユーザ権限の Android 機器を用いて Android クラスタを構築するために，MPI プログラムの実行環境を Android アプリケーションとして実装した．これにより，我々の開発した Android アプリケーションを Android 機器にインストールするだけで，スーパーユーザ権限を使用せずに Android クラスタを構築することが可能になった．

我々の MPI 実行環境のシステム構成を図 2 に示す．Android OS は Linux カーネルを基にカーネルを構築しており，その上でシステムライブラリを実装している．従来の Android クラスタでは，このシステムライブラリを保存するためのストレージ領域に並列処理環境のシステムファイルを格納していた．我々の MPI 実行環境では，Android アプリケーションがプライベートファイルを保存するために使用するストレージ領域である内部ストレージ領域に MPI プログラムや Open MPI ライブラリを格納する．内部ストレージ領域に格納する理由としては，ext 系のファイルシステムで管理されており，内部ストレージに格納した MPI 実行環境のファイルに適切な実行権限を付与することが可能だからである．

Open MPI 環境では，MPI 並列処理を実行するために，MPI プログラムのランチャーである mpirun コマンドを使用する．そのため，我々の MPI 実行環境についても，Android アプリケーション上から mpirun コマンド起動することにより MPI 並列処理を起動する．これを実現するためには，mpirun コマンドを実行する OS のプロセスを Android アプリケーション上から生成する必要がある．

我々の MPI 実行環境では，Android アプリケーション上からプロセスを生成するために，Java API Framework の ProcessBuilder クラスを利用する．ProcessBuilder クラスは，OS のプロセスの生成や管理を行うためのクラスであり，Android アプリケーションを記述するための Java API Framework において定義されている．我々の MPI 実行環境では，ProcessBuilder クラスを利用して Android アプリケーション上からのプロセス生成機能を実現した．

Android アプリケーション上からプロセスを生成する他の方法としては，Java Native Interface（JNI）によって fork システムコールを使用する方法がある．JNI は，Java アプリケーションとマシンネイティブなプログラムコードを連携するための仕組みである．JNI を利用することで，C/C++ 言語で記述した fork システムコールによるプロセス生成処理を，Android アプリケーション上から利用することができる．我々は，JNI によるプロセス生成についても実装を試みたが，fork システムコールの後に exec システムコールを起動してプロセスの実行を開始することが出来なかったため，JNI によるプロセス生成処理は実装できなかった．そのため，我々は，ProcessBuilder クラスを利用してプロセスの生成処理を実装した．

Open MPI 環境では，複数ノードで並列処理する場合，計算ノード間でのリモートログインを行うことで，リモートノード上

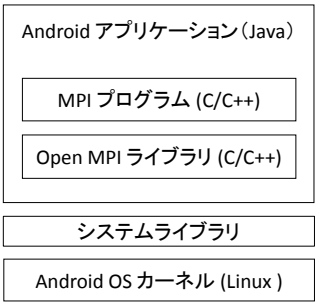

図 2　MPI 実行環境のシステム構成

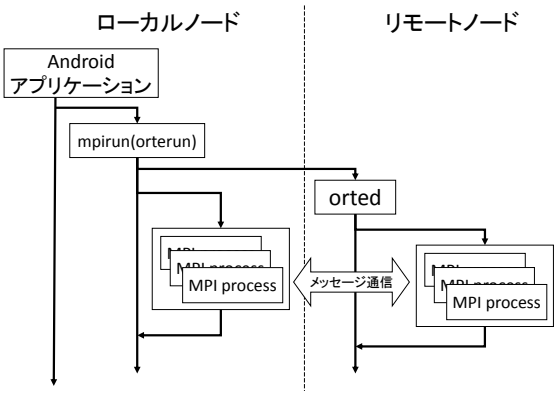

図 3　Android アプリケーション上での MPI 並列処理

で MPI プログラムを起動する．我々の MPI 実行環境では，最も一般的なリモートログインのためのツールである OpenSSH をリモートログインのために使用する．OpenSSH では，リモートログインを実行するために，リモートログインを受け付けるためのデーモンプログラムである sshd が各ノード上で起動していなければならない．我々の MPI 実行環境では，sshd を実行するプロセスについても，前述の ProcessBuilder クラスによる方法で Android アプリケーション上から起動している．OpenSSH は Secure Shell（SSH）プロトコルに基づいてリモートログインを行うが，SSH プロトコルのデフォルトポート番号は 22 番であり，ポートを開放するためにスーパーユーザ権限が必要である．そのため，我々の MPI 実行環境では，SSH プロトコルのためのポート番号として，一般ユーザ権限でも開放可能な他のポート番号を使用する．

我々の MPI 実行環境において，MPI 並列処理を起動する際の処理の流れを図 3 に示す．まず，Android アプリケーションから，MPI 並列処理プロセスを起動するためのプログラムである mpirun（orterun）を起動する．orterun はローカルノード上に MPI 並列処理プロセスを生成し，これらのプロセスの管理を行う．また，orterun はリモートノード上に，orted を起動する．orted はリモートノード上で MPI 並列処理プロセスの管理を行う．これらの MPI 並列処理プロセスはプロセス間で MPI のメッセージ通信を行う．

3. 性 能 評 価

我々の MPI 実行環境では, MPI 並列処理プロセスは Android

表 1　評価に使用する Android 機器の構成

端末名	Huawei MediaPad M3	Raspberry Pi 3 Model B
CPU	HiSilicon Kirin 950 8 コア 1.81GHz × 4, 2.3GHz × 4	ARM Cortex-A53 4 コア 1.2GHz
メモリ	LPDDR4 SDRAM 4GB	LPDDR2 SDRAM 1GB
OS	Android 7.0	Android 4.4
通信環境	IEEE 802.11 a/b/g/n/ac	100BASE-T

アプリケーションを介して起動する．そのため，計算ノード上で起動している Android アプリケーションのプロセスや，並列プロセスの起動方法などの影響により，MPI 並列処理にオーバヘッドが発生する可能性がある．よって，我々の MPI 実行環境が Android クラスタの並列処理性能に与える影響を明らかにする必要がある．本節では，我々の MPI 実行環境のオーバヘッドについて計測するため，先行研究のスーパーユーザ権限が必要な MPI 実行環境 [3] と我々の MPI 実行環境との比較評価を行う．比較評価には，MPI 並列化されたベンチマークプログラムを使用する．

3.1　評 価 環 境

評価には，4 台の Huawei MediaPad M3 を計算ノードとする Android クラスタ（MediaPad クラスタ）と，Android OS を導入した 16 台の Raspberry Pi 3 Model B を計算ノードとする Android クラスタ（Raspberry Pi クラスタ）の二種類のシステムを用いる．使用する Android 機器の構成を表 1 に示す．Huawei MediaPad M3（以下，MediaPad　M3）は，8 コア CPU と Android 7.0 を搭載したタブレットコンピュータ機器である．MediaPad は 5GHz 帯の Wi-Fi（IEEE 802.11 ac）による無線 LAN を使用して他の計算ノードと通信を行う．Wi-Fi アクセスポイントとしては，NEC 社製の Aterm WG2600HP2 を使用する．Raspberry Pi 3 Model B（以下，Raspberry Pi 3）は，4 コア CPU と Android 4.4 を搭載した小型コンピュータである．Raspberry Pi 3 は，他の計算ノードとの通信を 100BASE-T の有線 LAN を使用して行う．

3.2　ベンチマークプログラム

評価は次の三種類の MPI 並列化されたベンチマークプログラムを用いて行う．

（1） qn24b
（2） 姫野ベンチマーク
（3） 手書き文字の学習アプリケーション

以下，これらのベンチマークの特徴と Android クラスタ上で実行する際の設定について述べる．

3.2.1　qn24b

qn24b [4] は，組み合わせ最適化問題の 1 つである N クイーン問題の解法プログラムである．我々は，MPI 並列化された qn24b ベンチマークプログラムを評価に使用する．また，N クイーン問題のサイズ N は 17 に設定する．qn24b は，マスタ・ワーカ方式で実装されており，マスタがワーカに対してタスクを割り当てながら並列処理を行う．

3.2.2　姫野ベンチマーク

姫野ベンチマーク [5] は，ポアッソン方程式をヤコビの反復法で解く際の主要なループの処理速度を計測するベンチマークプログラムである．姫野ベンチマークの処理性能はメモリアクセス性能に大きな影響を受けると言われている．

本研究では，姫野ベンチマークの問題サイズは（i，j，k）＝（129 x 129 x 257）に設定する．並列処理を行う際には，この 3 次元格子をプロセス数で分割し，計算する領域を各プロセスに割り当てる．本研究では，分割を k 方向のみで行う．これは，プロセス間で行われる MPI 通信のデータ量を最も抑えることができるからである．分割方法の例として，8 プロセスで姫野ベンチマークを実行する場合には，分割数を（i，j，k）＝（1，1，8）とし，三次元格子を k 方向に 8 個の小領域に分割する．姫野ベンチマークは，主要なループのイテレーション数をベンチマークの実行時間が 60 秒になるように自動的に調整されるが，我々は，各計測の計算量を固定するために，ループのイテレーション数を 300 回に固定した．

3.2.3　ニューラルネットワークの学習アプリケーション

ニューラルネットワークの学習アプリケーションは，Android クラスタ上でニューラルネットワークの学習を行うアプリケーションである [7]．このアプリケーションは，ニューラルネットの C ライブラリである Fast Artificial Neural Network Library（FANN）を基に，ニューラルネットワークを用いて手書き文字を学習するアプリケーションである．

このアプリケーションは，データ並列学習に従って学習を MPI 並列化している．データ並列学習では，全てのプロセスが同じ学習モデルを生成し，分割した入力データを各プロセスが学習する．各プロセスの学習が終了した後，各モデルのパラメータの変化を集約し，モデルの更新を行う．モデルの更新結果は，全てのプロセスに更新結果をブロードキャストすることにより共有する．これを繰り返し行う．

このアプリケーションは，MNIST データセットを使用して，手書き文字の学習を順伝搬型ニューラルネットワークの教師あり学習で行う．ニューラルネットワークの学習に用いたモデルは，入力層のニューロンが入力データのピクセル数と同じ 784 個，出力層のニューロン数が 10 個である．中間層のニューロン数は，200 個で計測を行う．学習データは 60000 個用意し，学習は繰り返し 100 回行う．

3.3　評 価 結 果

3.3.1　qn24b

qn24b ベンチマークの速度向上率を図 4 に示す．図の縦軸は速度向上率である．図の横軸はプログラムを実行した際のノード数とプロセス数を表す．例えば，"1(4)"は，1 ノード 4 プロセスの実行を表す．また，New は，我々の MPI 実行環境におけるベンチマークプログラムの実行結果を表す．Conv. は，スーパーユーザ権限を必要とする従来の MPI 実行環境 [3] におけるベンチマークプログラムの実行結果を表す．なお，qn24b ベンチマークは，N クイーン問題を解く時間を計測するベンチマークである．そのため，速度向上率は Raspberry Pi クラスタと MediaPad クラスタのそれぞれについて，1 ノードでの Conv.

の実行時間を基準として求めた．

Raspberry Pi クラスタでは，New と Conv. を比較すると，1，2，4 および 16 ノードでの実行結果では，速度向上率に大きな違いは見られない．しかし，8 ノードでの実行結果では，New の速度向上率が 10.2 倍であり，Conv. の速度向上率が 8.4 倍であるため，New は Conv. と比較して大幅に速度向上率が高い．この原因としては，Conv. では，8 ノードでの qn24b ベンチマークの実行時に一部の計算ノードにおける CPU の動作クロック周波数が低下したことが考えられる．我々は，Raspberry Pi クラスタ中に動作周波数が低下しやすい計算ノードが存在することを確認している．Raspberry Pi 3 Model B が搭載する SoC である Broadcom BCM2837 は，CPU の温度が一定の温度に達すると冷却のために動作周波数を下げる機能を持っている．また，qn24b ベンチマークでは，マスタ・ワーカ方式で並列処理を行い，並列プロセス間での通信の待ち合わせをほとんど行わない．よって，他の計算ノードと比較して動作クロック周波数が低い計算ノードが存在したとしても，その計算ノードによってクラスタ全体の並列処理性能は律速されない．そのため，Conv. では並列処理中に動作クロック周波数が低下したノードが出現したことにより 8 ノードでの速度向上率が 8.4 倍になったと考えられる．動作クロック周波数が低下した計算ノードが出現しなければ，8 ノードについても New と Conv. は同程度の速度向上率になったと考えられる．

MediaPad クラスタでは，New と Conv. を比較すると，全体の傾向として New の方が低い速度向上率である．1 ノードと 2 ノードのときは，New が Conv. よりも 2 割ほど速度向上率が低い．3 ノード以上の実行結果については，New と Conv. は同程度の速度向上率である．このような結果となった原因としては，MediaPad M3 は big.LITTLE 構成であることが考えられる．MediaPad M3 の CPU は，動作周波数 2.3GHz の big コア 4 つと，動作周波数 1.81GHz の LITTLE コアからなる．1 ノード 8 プロセスで qn24b を実行するときは，LITTLE コアも qn24b の並列処理プロセスを実行する．我々の MPI 実行環境では，並列処理中にも MPI 実行環境のための Android アプリケーションのプロセスが常駐する．このプロセスは，MPI 並列プロセスの標準出力への出力を読み込むことや，Android アプリケーションのユーザインターフェースの処理などを行う．Android 機器が実行するプロセスをシステムレベルでトレースできる Android SDK のコマンドである systrace コマンドを使用して確認したところ，このようなプロセスは qn24b の実行中には LITTLE コアに割り当てられることが分かった．LITTLE コアの計算リソースはこのようなプロセスのために頻繁に使用されることから，qn24b のために使用できる計算リソースが減少し，New の速度向上率が Conv. と比較して低くなったと考えられる．

3.3.2 姫野ベンチマーク

姫野ベンチマークの性能向上率を図 5 に示す．姫野ベンチマークはシステムの MFLOPS 値を測るプログラムである．そのため，性能向上率は，Raspberry Pi クラスタと MediaPad クラスタのそれぞれについて，1 ノードでの Conv. の MFLOPS 値を基準として求めた．

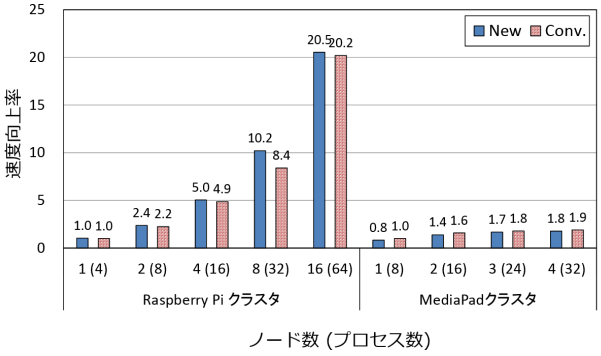

図 4　qn24b の速度向上率

Raspberry Pi クラスタでは，1 ノードおよび 2 ノードの New と Conv. については同程度の性能向上率を得た．しかし，4 ノード以降の性能向上率は大幅に異なっている．8 ノードのときは，New は Conv. よりも約 6 割高い速度向上率である．New が Conv. よりも高い速度向上率である原因は，前述の CPU の動作クロック周波数の低下によるものと考えられる．姫野ベンチマークでは，並列処理中にプロセス同士の通信による待ち合わせ処理が発生する．そのため，クラスタ中のどれか 1 つのノードの動作クロック周波数が低下した場合，そのノードがクラスタ全体の処理性能を律速する．Conv. では，姫野ベンチマークの実行中に動作クロック周波数が低下したノードが出現したため，New と比較して性能が低くなったと考えられる．一方，New では，動作クロック周波数が低下する計算ノードが出現しなかったため，動作クロック周波数の低下による並列処理性能の低下も起こらなかったと考えられる．Raspberry Pi の動作周波数の変化による処理性能への影響については，我々が今後明らかにすべき課題である．

MediaPad クラスタでは，qn24b の実行結果とは異なり，New と Conv. で同程度の速度向上率が得られた．MediaPad クラスタでは，姫野ベンチマークの実行結果からみると，我々の MPI 実行環境を用いたことによる処理性能の低下は見られなかった．これは，姫野ベンチマークのプログラムの性質によるものと考えられる．姫野ベンチマークは，並列プロセス間で通信の待ち合わせを行うプログラムであるため，無線通信のオーバーヘッドによる並列処理性能への影響が大きく，我々の MPI 実行環境が並列処理性能に与えるオーバーヘッドの影響が相対的に小さくなったことが考えられる．姫野ベンチマークでは，プログラムの実行中に我々の MPI 実行環境のプロセスが CPU のリソースを消費していたとしても，それが並列処理性能に与える影響は小さいと考えられる．

3.3.3 ニューラルネットワークの学習アプリケーション

ニューラルネットの学習処理の速度向上率を図 6 に示す．速度向上率は Raspberry Pi クラスタと MediaPad クラスタのそれぞれについて，1 ノードでの Conv. の学習時間を基準として求めた．

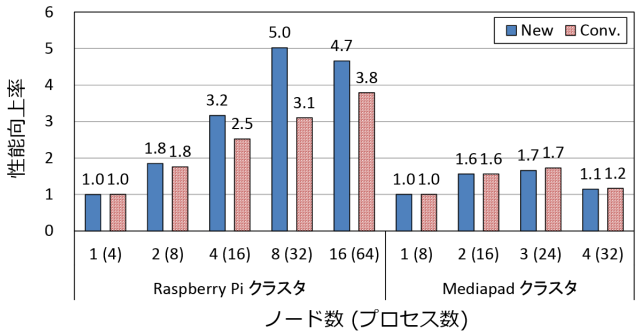

図 5　姫野ベンチマークの性能向上率

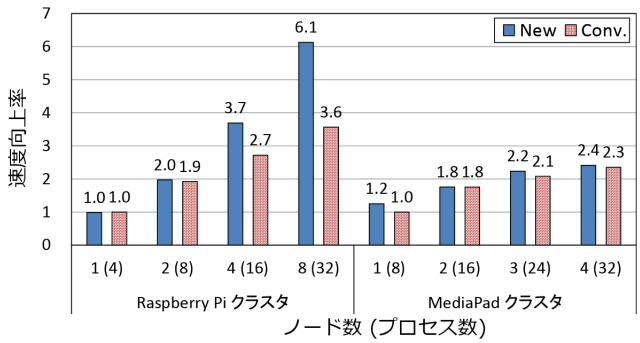

図 6　ニューラルネットの学習処理の速度向上率

図 6 では，Raspberry Pi クラスタにおいて，ニューラルネットワークの学習アプリケーションを 16 ノードで実行することが出来なかったため，8 ノードまでの実行性能を示す．これは，アプリケーションのバグによりセグメンテーションフォールトでアプリケーションが終了するためである．

Raspberry Pi クラスタでは，1 ノードおよび 2 ノードの New と Conv. の速度向上率では，同程度の速度となった．しかし，4 ノードおよび 8 ノードでの速度向上率については，New の性能が高い結果となった．この原因についても，前述した Raspberry Pi の CPU の動作クロック周波数の低下によるものだと考えられる．ニューラルネットワークの学習アプリケーションでは，各並列プロセスが学習を行った後，学習したモデルのパラメータの変化を集約し，モデルの更新を行う．このとき，プロセス同士の通信による待ち合わせ処理が発生する．他のプロセスと比較して学習に時間が掛かるプロセスがあった場合，このプロセスによって全体の処理性能が律速される．Conv. では，4 ノードおよび 8 ノードでの並列処理時に動作クロック周波数が低下したノードが出現したため，New よりも低い速度向上率になったと考えられる．

MediaPad クラスタでは，1 ノードにおいて New と Conv. を比較したとき，New が 2 割ほど高い性能となった．この原因は現在解析中であり，詳細な解析は今後の我々の課題である．2 ノード以上については，New と Conv で同程度の性能となった．

4. おわりに

本研究では，Android アプリケーションとして MPI 実行環境を実現することで，スーパーユーザ権限を必要としない Android クラスタシステムを構築し，ベンチマークプログラムによって従来の MPI 実行環境との比較評価を行った．結果として，我々の MPI 実行環境を用いた場合においても，従来の MPI 実行環境と同程度の速度向上率が得られることが分かった．ただし，我々の MPI 実行環境では，qn24b のような並列プロセス間でほとんど通信の待ち合わせを行わないプログラムでは，Android アプリケーションのプロセスを起動していることに起因するオーバヘッドが見られた．

今後の課題としては，オーバーヘッドについて詳細に解析するために，より多くのベンチマークプログラムを用いて我々の MPI 実行環境を評価することや，Android 機器の CPU の動作周波数の変化による並列処理性能への影響について明らかにすることが必要である．

謝辞　本研究は一部 JSPS 科研費 16K00068, 17K00072 の助成による．

文　献

[1] F. Busching, S. Schildt, and L. Wolf, “DroidCluster: Towards Smartphone Cluster Computing–The Streets are Paved with Potential Computer Clusters,” In Distributed Computing Systems Workshops (ICDCSW), pp.114-117, 2012.

[2] G. Hinojos, C. Tade, S. Park, D. Shires, and D. Bruno, “BlueHoc: Bluetooth Ad-Hoc Network Android Distributed Computing,” Int. Conf. on Parallel and Distrib. Process. Tech. and Appl. (PDPTA), pp.468-473, 2013.

[3] Yuki Sawada, Yusuke Arai, Kanemitsu Ootsu, Takashi Yokota, Takeshi Ohkawa, “Performance of Android Cluster System Allowing Dynamic Node Reconfiguration,” Wireless Personal Communications, Vol.93, Issue 4, pp.1067-1087, Apr. 2017.

[4] 吉瀬　謙二，片桐　孝洋，本多　弘樹，弓場　敏嗣，“qn24b:N-queens の解を計算するベンチマークプログラム，”　FIT2004 第 3 回情報科学技術フォーラム，第 4 分冊，No.O-011，pp.389-392，2004.

[5] “姫野ベンチマーク，”　http://accc.riken.jp/supercom/documents/himenobmt/，（2018 年 12 月 17 日参照．）

[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 andS. Weeratunga, “THE NAS PARALLEL BENCHMARKS”, RNR Technical Report, RNR-94-007, March, 1994.

[7] 杉山　裕紀，大津　金光，大川　猛，横田　隆史，“無線接続型 Android クラスタにおける MPI 並列プログラムの性能評価，”　信学技報，Vol.118, No.165, CPSY2018-13, pp.1-6, July, 2018.

[8] A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary, “HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers” http://www.netlib.org/benchmark/hpl/, （2018 年 12 月 17 日参照．）

[9] Edgar Gabriel et al., “Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation”, Proceedings, 1th European PVM/MPI Users’ Group Meeting, pp.97-104, 2014.

[10] “Android Developers,” https://developer.android.com/, （2018 年 12 月 17 日参照．）