Vol.2015-EMB-38 No.3
2015/8/28

LTE無線基地局におけるレイヤ2信号処理のOSCARコンパイラによる自動並列化

田中 優利[1,a)]　小松 裕樹[1]　影浦 直人[1]　見神 広紀[1]　松元 映二[4]　横山 正浩[2]　江崎 孝斗[2]
箕輪 守彦[2]　高村 守幸[3]　木村 啓二[1]　笠原 博徳[1]

概要：スマートデバイスの普及に伴う移動体通信の急速な普及や端末の通信スループット増大に伴い，端末と基地局の両者に要求される信号処理は増大の一途を辿っている．また，基地局のプログラムは膨大なトラフィックを高速に処理するために複雑化している一方で，将来の機能拡張性の考慮や標準化規格の定期的な更新への対応が必須となっている．このような背景から，無線基地局における信号処理はこれまで一般的だったハードウェアのみの実装ではなく，DSP を用いたソフトウェアによる実装が一般的になっている．加えて近年では，より高い処理性能を実現するためにマルチコア DSP を搭載した計算機を用いることが主流となってきている．マルチコア DSP の資源を最大限活用するためには，ソフトウェアの並列化が必要となるが，ソフトウェア更新毎に人手で並列化を行うには膨大な工数が必要となるだけでなく，そもそも潜在的な並列性を人手で見出すことが困難だという課題がある．本研究では，LTE 無線基地局におけるレイヤ 2 信号処理アプリケーションに対して，OSCAR 自動並列化コンパイラを用いて並列化を行い，無線基地局向けのマルチコア DSP を搭載した freescale MSC8156 上で評価した．freescale マルチコア DSP 上で，レイヤ 2 信号処理のサブレイヤである MAC レイヤ，及び RLC レイヤにおける各機能ブロックの並列化を行ったところ，MAC レイヤではヘッダ解析処理が，1 コア逐次実行時に比べて 6 コア並列実行時に 3.02 倍，SCH 終端処理が，1 コア逐次実行時に比べて 6 コア並列実行時に 3.53 倍の速度向上が確認できた．また，RLC レイヤでは PDU 受信処理が 1 コア逐次実行時に比べて 6 コア並列実行時に 3.14 倍の速度向上が確認され，freescale マルチコア DSP 上での無線基地局信号処理アプリケーションにおける自動並列化コンパイラを用いた並列化の有用性が確認できた．

1. はじめに

スマートフォンやタブレット端末などのスマートデバイスの普及に伴う移動体通信の急速な普及や，端末の通信スループット増大に伴い，端末と基地局の両者における信号処理は複雑化し，要求される処理性能は増大の一途を辿っている．特に，第 3 世代以降の移動体通信においては，これまで一般的だったハードウェアのみの実装では複雑で高い処理性能が求められる基地局の信号処理を実現することは困難になっている．また，近年では移動体通信への機能，性能両面での要求の高まりによって，通信規格の更新も頻繁に行われるようになったことから，定期的な処理方式の更新を行うことが求められている．このような背景から，第 3 世代以降の移動体通信では，信号処理に特化した計算機である DSP (Digital Signal Processor) を用いて，ソフトウェアにより信号処理を実現する実装が一般的になってきている．

加えて近年では，単一の DSP による処理では処理性能に限界があるため，複数の DSP を用いた並列処理が行われているが [1]，人手によるプログラムの並列化は膨大な工数が必要になるだけでなく，そもそも潜在的な並列性を人手のみで見出すことが困難だという問題がある．そのためこれまでは，端末から基地局への無線通信を処理するアップリンクと基地局から端末への無線通信を処理するダウンリンクなど，独立した機能単位で各 DSP に処理を振り分けるような機能分散による並列処理を行っていた．しかしながら，機能分散による並列処理では各機能の処理時間が不均衡となることなどの理由によって処理性能の向上には限界がある．すなわち，今後の無線通信の高速化には，各機能単位の自動並列化による処理性能の向上が必須となる．

[1] 早稲田大学
Waseda University.
[2] 富士通株式会社
FUJITSU LIMITED.
[3] 株式会社富士通研究所
FUJITSU LABORATORIES LTD.
[4] 富士通九州ネットワークテクノロジーズ株式会社
Fujitsu Kyushu Network Technologies Limited.
[a)] yuuri@kasahara.cs.waseda.ac.jp

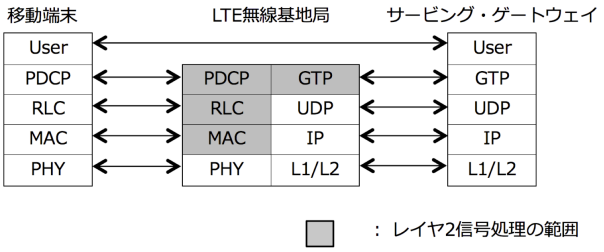

図 1 U-Plane のプロトコルスタックとレイヤ 2 信号処理の範囲

自動並列化を実現するコンパイラとして，OSCAR 自動並列化コンパイラ [2] の研究，開発が行われている．OSCAR 自動並列化コンパイラでは，Parallelizable C で記述されたプログラムの潜在的な並列性を解析し，マルチグレイン並列化されたプログラムを出力する．

LTE (Long Term Evolution) 無線基地局の処理はレイヤ 1 からレイヤ 3 までの各レイヤに分割できる．特に本研究で対象とするアップリンクのレイヤ 2 処理では，レイヤ 1 から送信されてきたデータのヘッダを解析し，MAC (Media Access Control)，RLC (Radio Link Control) のサブレイヤの順にデータを抽出していく必要があるため，各機能間に依存関係を多く含んでいる．このように，レイヤ 2 は機能部単位で見た場合に他のレイヤと比較して処理の独立性が低いことが特徴となっているため，機能分散による並列化が困難である．そこで本研究では，レイヤ 2 信号処理に対して OSCAR 自動並列化コンパイラを用いた負荷分散による並列化を適用した．これらの処理に対して，並列化が自動で実現できるようになれば，より高い処理性能の実現が期待できるだけでなく，飛躍的な生産性の向上にも貢献できる．

本稿では，第 2 節でレイヤ 2 信号処理の概要について，第 3 節で並列性抽出のためのリストラクチャリングについて，第 4 節で OSCAR 自動並列化コンパイラについて，第 5 節でレイヤ 2 信号処理の並列化コードの評価環境及び評価結果について述べる．

2. レイヤ 2 信号処理の概要

LTE は，ユーザ情報を送受信する場合と，ネットワーク内の制御を行う場合でプロトコルスタックが異なる [3]．前者を U-plane (User plane)，後者を C-plane (Control plane) と呼ぶ．本稿におけるレイヤ 2 信号処理は，ユーザ情報を送受信する U-plane を対象としている．U-plane のプロトコルスタックとレイヤ 2 信号処理の範囲を図 1 に示す．

図 1 に示したように，U-plane のプロトコルスタックのうち，レイヤ 2 信号処理に該当する主なサブレイヤは，MAC レイヤ，RLC レイヤであり，他に PDCP レイヤ，GTP レイヤがある．

次に，各サブレイヤの概要について述べる．MAC レイヤは，無線リソース割当て，データ伝送などの処理を行う際

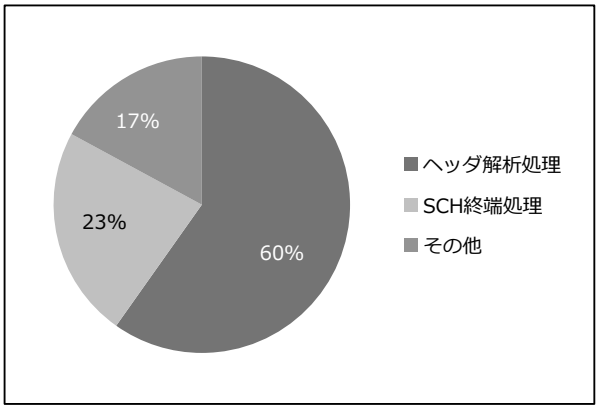

図 2 MAC レイヤのプロファイル結果

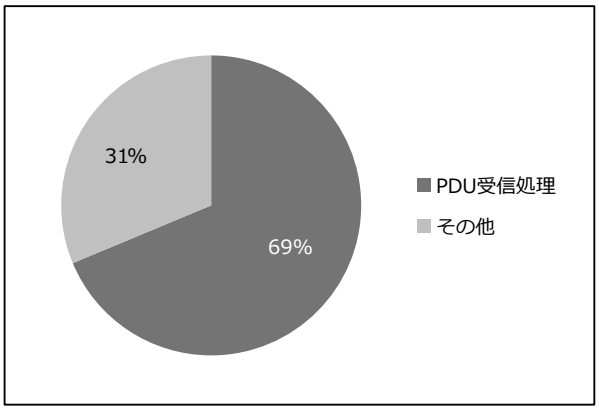

図 3 RLC レイヤのプロファイル結果

の基本単位である TB (Transport Block) へのデータマッピング，及び再送制御を行う．RLC レイヤは，プロトコルが処理するデータ単位となる PDU (Protocol Data Unit) の再送制御，重複検出，順序整列を行う．PDCP (Packet Data Convergence Protocol) レイヤは，秘匿，ヘッダ圧縮などを行う．GTP (GPRS Tunneling Protocol) レイヤは，基地局とサービング・ゲートウェイ間で IP 伝送を行う．本稿では，レイヤ 2 信号処理の主なサブレイヤである MAC レイヤ及び RLC レイヤの処理のうちアップリンクの処理を対象とする．

次に，MAC レイヤ及び RLC レイヤを対象として，レイヤ内の各機能のプロファイリングを行い，ボトルネックとなっている機能について調査した．プロファイリングで用いた機材の詳細は 5.1 節で述べる．MAC レイヤのプロファイル結果を図 2 に，RLC レイヤのプロファイル結果を図 3 にそれぞれ示す．

図 2 より，MAC レイヤでボトルネックとなっている機能はヘッダ解析処理及び SCH (Shared CHannel) 終端処理であることが判明した．ここで Shared Channel とは，レイヤ 1 でデータ転送に使用される物理チャネルである．本研究の対象はアップリンク処理となるのでここでの SCH はレイヤ 1 における PUSCH (Physical Uplink Shared CHannel)

を意味し，これはレイヤ 2 における MAC PDU を意味する．また図 3 より，RLC レイヤでボトルネックとなっている機能は PDU 受信処理であることが判明した．

次に，プロファイリングによってボトルネックとなっていることが明らかになったヘッダ解析処理，SCH 終端処理，及び PDU 受信処理の概要について述べる．アップリンクにおけるヘッダ解析処理は，レイヤ 1 から通知された MAC PDU 数分の MAC ヘッダ情報を解析し，MAC CE (Control Element)，及び MAC SDU (Service Data Unit) の分離等を行う．アップリンクにおける SCH 終端処理は，MAC ヘッダ解析で得られた MAC SDU の情報から，RLC PDU を抽出し，RLC レイヤに対して RLC PDU の転送を行う．アップリンクにおける PDU 受信処理は，端末と基地局間で通信するデータの一致確認を行う．

3. 並列性抽出のためのリストラクチャリング

本節では，並列性抽出のために行った以下の 3 つのリストラクチャリングについて述べる．

- ヘッダ解析処理において破棄情報を管理する仕組みを導入することで，MAC PDU の破棄を考慮した状態で並列性を抽出
- SCH 終端処理においてメモリ確保を動的確保から静的確保に変更
- PDU 受信処理において RLC PDU 取り出し処理に対するループディストリビューションを適用

3.1 MAC PDU の破棄を考慮したヘッダ解析処理の並列性抽出

ヘッダ解析処理では，レイヤ 1 から通知された MAC PDU に含まれる MAC ヘッダを解析する．解析結果は配列 result に格納される．ここで，全ての MAC PDU が正常に受信されている場合には MAC PDU 間に依存がないためアルゴリズム上は並列性が存在する．しかしながら，受信した MAC PDU が破損していた場合，当該 MAC PDU は破棄され，次の SCH 終端処理には通知されない．そのため，オリジナルソースでは図 4 に示すように，MAC PDU の破棄を考慮し，それ以降の解析結果を配列 result に詰めて格納するため，並列化の阻害要因となっていた．

そこで，MAC PDU の破棄を考慮した仕組みを維持した状態で並列性を抽出するため，図 5 に示すように MAC PDU の破棄状況を管理するためのフラグを用いたリストラクチャリングを行った．このリストラクチャリングにより，SCH 終端処理への通知用配列 result のインデックスが元の MAC PDU の順番と一致するため，ループイタレーション間の依存を解消することができる．同様の構造が SCH 終端処理にも存在していたため，併せてリストラクチャリングを行うことで並列性を抽出した．

```
struct _str result[全 MAC PDU 数];
-----------------------------------------
ヘッダ解析 () {
    index = 0;
    for (i = 0; i < 全 MAC PDU 数; i++) {
        work = 解析本体 ();
        if (破棄判定 (work)  == TRUE) {
            continue;
        }
        memcpy(result[index], work, size);
        index++;
    }
}

SCH 終端 (int index) {
    work = result[index];
    ...
}

MAC 処理 () {
    ヘッダ解析 ();
    for (i = 0; i < 正常な MAC PDU 数; i++) {
        SCH 終端 (i);
    }
}
```

図 4 ヘッダ解析処理における並列化の阻害要因

3.2 SCH 終端処理におけるメモリ動的確保の静的確保への変更

SCH 終端処理では，ヘッダ解析処理後に整形され保存されたデータをヘッダ解析と同様に MAC PDU 単位に処理するため，アルゴリズム上は並列性が存在する．しかしながら，SCH 終端処理後に RLC レイヤ向けにデータを整形して保存する過程で，保存先のデータサイズが変動するため動的メモリ確保によって保存先の領域を確保している．そのため，処理コストの大きい排他制御が必要となってしまい，並列処理による高速化の効果が著しく低下してしまう．そこで，本研究では当該メモリ確保部分を動的メモリ確保から静的メモリ確保に変更することで並列化を行った．静的メモリ確保によってメモリ確保を行うためには，動的に変化する領域の最大値を確保しなければならないため，メモリ領域を無駄にしてしまう．これは並列化を実施することとのトレードオフであるが，今回のプログラムではメモリ使用量の増大が大きな影響を及ぼさないことを確認した上で，上記の修正を適用した．

3.3 RLC PDU 受信処理におけるループディストリビューション

PDU 受信処理では，MAC レイヤから受け取った RLC PDU 単位で処理をするため，RLC PDU 間に依存がなく

```
struct _str result[全 MAC PDU 数];
int destruction_flag[全 MAC PDU 数] = {0};
-----------------------------------------
ヘッダ解析 () {
    for (i = 0; i < 全 MAC PDU 数; i++) {
        work = 解析本体 ();
        if (破棄判定 (work)  == TRUE) {
            destruction_flag[i] = 1;
            continue;
        }
        memcpy(result[i], work, size);
    }
}

SCH 終端 (int index) {
    work = result[index];
    ...
}

MAC 処理 () {
    ヘッダ解析 ();
    for (i = 0; i < 全 MAC PDU 数; i++) {
        if (destruction_flag[i] != 1) {
            SCH 終端 (i);
        }
    }
}
```

図 5 リストラクチャリングにより並列性が抽出されたヘッダ解析処理

```
RLC 処理 () {
    struct _str pdu;

    /***PDU 受信処理 ここから***/
    for (i = 0; i < 全 RLC PDU 数; i++) {
        pdu = RLC PDU 取り出し処理 ();
        処理本体 (pdu);
    }
    /***PDU 受信処理 ここまで***/
    ...
}
```

図 6 PDU 受信処理における並列化の阻害要因

アルゴリズム上は並列性が存在する．しかしながら，MAC レイヤにおける処理が完了後，MAC レイヤは RLC PDU をリスト状に構成し RLC レイヤに通知する．PDU 受信処理では図 6 のように，初めにリスト構造になった RLC PDU を順に取り出しながら処理を進めていくため，ループイタレーション間に依存が存在していた．

そこで並列性を抽出するため，図 7 に示すようにループ

```
RLC 処理 () {
    struct _str pdu[全 RLC PDU 数];

    /***PDU 受信処理 ここから***/
    for (i = 0; i < 全 RLC PDU 数; i++) {
        pdu[i] = RLC PDU 取り出し処理 ();
    }
    for (i = 0; i < 全 RLC PDU 数; i++) {
        処理本体 (pdu[i]);
    }
    /***PDU 受信処理 ここまで***/
    ...
}
```

図 7 リストラクチャリングにより並列性が抽出された PDU 受信処理

ディストリビューションにより MAC レイヤから受信した RLC PDU を取り出す処理を切り離した．このリストラクチャリングにより，RLC PDU を取り出す処理のみ先に逐次で実行して全ての RLC PDU を取り出した後，ループイタレーション間の依存が解消された他の処理を並列に実行することが出来るようになった．

4. OSCAR 自動並列化コンパイラ

本節では，レイヤ 2 信号処理アプリケーションの並列化に使用した OSCAR 自動並列化コンパイラについて述べる．

OSCAR 自動並列化コンパイラにおけるマルチグレイン並列化では，粗粒度並列性，中粒度並列性及び近細粒度並列性の複数粒度の並列性を組み合わせて並列化を行う [4], [5], [6]．粗粒度並列性としては，複数の代入文等によって構成される基本ブロック，関数呼び出し，及び繰り返し文の各ブロック間に存在する並列性を抽出する．中粒度並列性としては，ループイタレーション間に存在する並列性を抽出する．近細粒度並列性としては，基本ブロック内の各ステートメント間に存在する並列性を抽出する．OSCAR コンパイラは，Fortran または Parallelizable C[7] で記述された逐次プログラムの並列性を解析し，並列化されたプログラムを出力する．ここで，Parallelizable C とは自動並列化コンパイラによる並列性抽出を補助するための制約付き C 言語である．C 言語はポインタの利用等，記述の自由度が高いことからアルゴリズム上は並列性が存在してもコンパイラによって十分な並列性を抽出できない場合があるため，本稿におけるレイヤ 2 信号処理アプリケーションにおいても Parallelizable C の規約を遵守する形に変更を加えている．

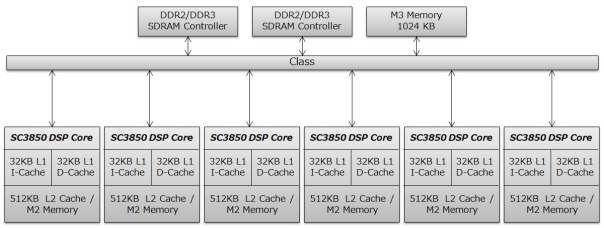

図 8 freescale MSC8156 のメモリ構成図

表 1 レイヤ 2 信号処理アプリケーションの実行シナリオ

入力データ	pattern1	pattern2
MAC PDU	36	36
MAC SDU	1SDU/PDU	1SDU/PDU
RLC SDU	-	1SDU/PDU

5. 評価結果

本節では，まずレイヤ 2 信号処理アプリケーションの性能評価を行った評価環境である freescale MSC8156 マルチコア DSP[8] の構成について述べる．次にレイヤ 2 信号処理アプリケーションに対して 3 節で述べたリストラクチャリングを適用し，OSCAR コンパイラで並列化した際の MSC8156 上での性能評価結果を述べる．

5.1 評価環境

本稿の評価対象であるレイヤ 2 信号処理アプリケーションは，富士通株式会社のプログラムである．レイヤ 2 信号処理アプリケーションの性能評価を行った評価環境は，実際の基地局開発で用いられる freescale MSC8156 マルチコア DSP を使用する．ここで，MSC8156 のメモリ構成図を図 8 に示す [9]．MSC8156 は StarCore 3850 DSP(動作周波数:1.0[GHz]) を 6 コア集積したマルチコア DSP である．この StarCore 3850 プロセッサは，各コアあたり L1 命令キャッシュ 32[KB] と L1 データキャッシュ 32[KB]，L2 キャッシュとローカルメモリ (M2 メモリ) の切り替えが可能な領域 512[KB]，オンチップ共有メモリ (M3 メモリ)1056[KB]，オフチップ共有メモリとして最大 1024[MB] のメモリと接続可能な DDR メモリコントローラを 2 つ搭載している．本評価では L2 キャッシュとローカルメモリの切り替えが可能な領域は全てローカルメモリ (M2 メモリ) として使用した．また，各メモリにはキャッシュを使用する CACHEABLE 領域とキャッシュを使用しない NONCACHEABLE 領域の 2 種類の領域を設定可能である．

次に，レイヤ 2 信号処理アプリケーションを実行するために使用した実行シナリオを表 1 に示す．表 1 に示すように，本稿での評価では MAC レイヤの処理のみを行う pattern1 及び MAC レイヤ，RLC レイヤの順に処理を行う pattern2 の 2 つのシナリオを使用した．MAC レイヤにおけるヘッダ解析処理，及び SCH 終端処理については pattern1 を，RLC レイヤにおける PDU 受信処理については pattern2 をそれぞれ使用して評価を行った．

また，OS はマルチコア DSP 向けに設計されたリアルタイム OS である freescale SmartDSP OS[10] を使用し，サイクル数取得には MSC8156 に搭載されている DPU カウンタを用いて計測を行った．

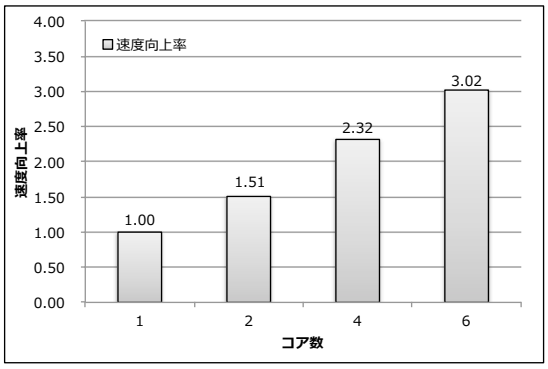

図 9 MAC レイヤ ヘッダ解析処理の並列化評価結果

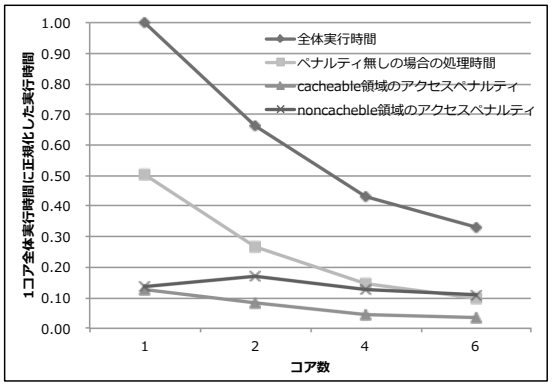

図 10 コア数の増加に対するヘッダ解析処理の実行時間

5.2 レイヤ 2 信号処理の並列化性能評価

本節では，レイヤ 2 信号処理アプリケーションに対して 3 節で述べたリストラクチャリングを適用し，OSCAR コンパイラで並列化した際の MSC8156 上での性能評価結果について述べる．

まず，レイヤ 2 信号処理のサブレイヤのうち，MAC レイヤ内の処理であるヘッダ解析処理の評価結果について述べる．図 9 に 1 コア，2 コア，4 コア，6 コアそれぞれでヘッダ解析処理を実行したときの 1 コアの実行時間を基準とした速度向上率のグラフを示す．図 9 より，ヘッダ解析処理全体で従来の 1 コア上での逐次の実行に対して，2 コアで 1.51 倍，4 コアで 2.32 倍，6 コアで 3.02 倍の速度向上を確認した．

ここで実行時間の内訳を分析するため，コア数の増加に対するヘッダ解析処理の実際の実行時間，ペナルティが皆無の場合の実行時間，及びメモリアクセスのペナル

ティを計測した結果を図 10 に示す．メモリアクセスのペナルティについては，CACHEABLE 領域へのアクセス及び NONCACHEABLE 領域へのアクセスのペナルティをそれぞれ計測した．なお図 10 の実行時間は，1 コア時の実際の実行時間に正規化したものである．

図 10 では，ペナルティが皆無の場合の実行時間及び CACHEABLE 領域のアクセスペナルティはコア数の増加に対して実行時間が減少しているが，NONCACHEABLE 領域のアクセスペナルティは 2 コアで増加しており，4 コア，6 コアでもほとんど減少していないことが確認できる．この結果から，ヘッダ解析処理における NONCACHEABLE 領域へのアクセス要因について調査を行った．

2 コアで NONCACHEABLE 領域のアクセスペナルティが増加している原因は，マルチコアでの並列実行を行うためにコア間で共有が必要になった変数の多くを M3 メモリの NONCACHEABLE 領域に配置していることである．共有変数を NONCACHEABLE 領域に配置したのは，M3 メモリにおいて CACHEABLE 領域へ配置した上でキャッシュフラッシュを行った際のペナルティと比較し，サイクル数において優位性が確認できたためである．さらに，NONCACHEABLE 領域へのアクセス要因を調査した結果，NONCACHEABLE 領域へのアクセスを頻繁に行っている箇所は，本プログラム中に実装された機能分散によるマルチコアでの実行を容易にするための機構であることが分かった．この機構はレイヤ 2 信号処理以外の処理にも共通して使用されているものである．そのため，さらなる速度向上を実現するためにはこの機構において NONCACHEABLE 領域へのアクセスを除去する，あるいはこの機構を使用しない方式を新たに検討することが必要である．

次に，レイヤ 2 信号処理のサブレイヤのうち，MAC レイヤ内の処理である SCH 終端処理の評価結果について述べる．図 11 に 1 コア，2 コア，4 コア，6 コアそれぞれで SCH 終端処理を実行したときの 1 コアの実行時間を基準とした速度向上率のグラフを示す．図 11 より，SCH 終端処理全体で従来の 1 コア上での逐次の実行に対して，2 コアで 1.56 倍，4 コアで 2.88 倍，6 コアで 3.53 倍の速度向上を確認した．

ここで図 10 と同様に，コア数の増加に対する SCH 終端処理の実際の実行時間，ペナルティが皆無の場合の実行時間，及びメモリアクセスのペナルティを計測した結果を図 12 に示す．

図 12 では，ペナルティが皆無の場合の実行時間はコア数の増加対して減少しているが，CACHEABLE 領域及び NONCACHEABLE 領域のアクセスペナルティは 2 コアで増加している．また 4 コア，6 コア実行時の CACHEABLE 領域及び NONCACHEABLE 領域のアクセスペナルティの減少はほとんど見られないが，NONCACHEABLE 領域

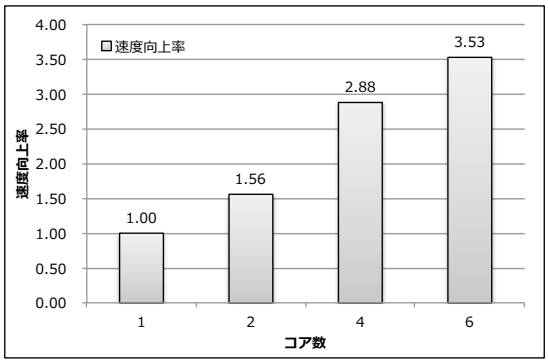

図 11 MAC レイヤ SCH 終端処理の並列化評価結果

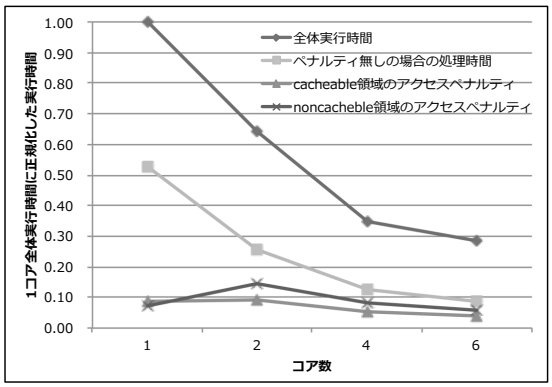

図 12 コア数の増加に対する SCH 終端処理の実行時間

のアクセスペナルティは特に減少していない．この結果から，CACHEABLE 領域及び NONCACHEABLE 領域のアクセスペナルティが 2 コアで増加している原因，及び SCH 終端処理に含まれる NONCACHEABLE 領域へのアクセス要因について調査を行った．

CACHEABLE 領域のアクセスペナルティが増加している原因は，並列実行時の共有変数にサイズの大きい変数が存在したため，DDR の CACHEABLE 領域に配置したことである．また，NONCACHEABLE 領域のアクセスペナルティが増加している原因は，ヘッダ解析同様，共有変数の多くを M3 メモリの NONCACHEABLE 領域に配置したことである．さらに，NONCACHEABLE 領域へのアクセス要因を調査した結果，NONCACHEABLE 領域へのアクセスを行っている箇所は OSCAR コンパイラによる並列化対象外の処理であるメモリ制御を行う OS 関数内であることが分かった．

次に，レイヤ 2 信号処理のサブレイヤのうち，RLC レイヤ内の処理である PDU 受信処理の評価結果について述べる．図 13 に 1 コア，2 コア，4 コア，6 コアそれぞれで PDU 受信処理を実行したときの 1 コアの実行時間を基準とした速度向上率のグラフを示す．図 13 より，PDU 受信処理全体で従来の 1 コア上での逐次の実行に対して，2 コアで 1.69 倍，4 コアで 2.75 倍，6 コアで 3.14 倍の速度向上を確認した．

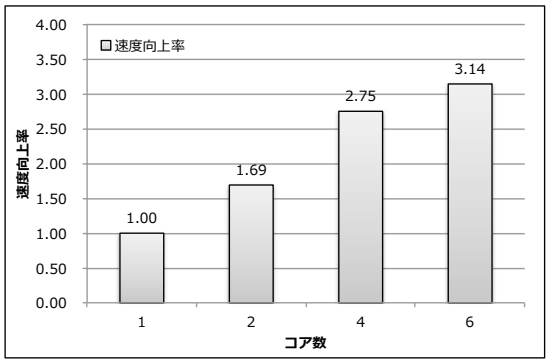

図 13 RLC レイヤ PDU 受信処理の並列化評価結果

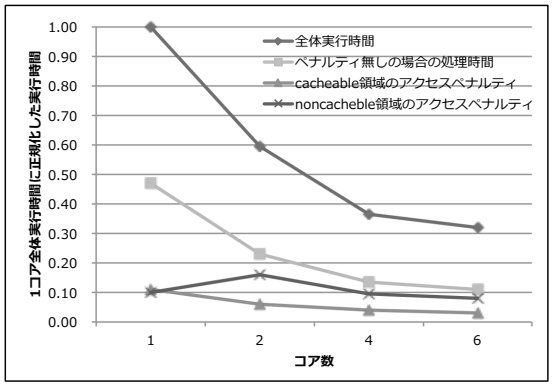

図 14 コア数の増加に対する PDU 受信処理の実行時間

ここで図 10，図 12 と同様に各コア数利用時の実行時間の内訳を計測した結果を図 14 に示す．図 14 では，ペナルティが皆無の場合の実行時間及び CACHEABLE 領域のアクセスペナルティはコア数の増加に対して実行時間が減少しているが，NONCACHEABLE 領域のアクセスペナルティは 2 コアで増加しており，4 コア，6 コアでもほとんど減少していないことが確認できる．この結果から，NONCACHEABLE 領域のアクセスペナルティが 2 コアで増加している原因，及び PDU 受信処理に含まれる NONCACHEABLE 領域へのアクセス要因について調査を行った．

NONCACHEABLE 領域のアクセスペナルティが 2 コアで増加している原因は，ヘッダ解析処理及び SCH 終端処理同様，共有変数の多くを M3 メモリの NONCACHEABLE 領域に配置したことである．さらに，NONCACHEABLE 領域へのアクセス要因について調査した結果，SCH 終端処理同様，NONCACHEABLE 領域へのアクセスを行っている箇所は OSCAR コンパイラによる並列化対象外の処理であるメモリ制御を行う OS 関数内であることが分かった．

以上より，OSCAR コンパイラによりヘッダ解析処理，SCH 終端処理，及び PDU 受信処理の各処理ともにコア数の増加とともに性能向上が得られることを確認した．

6. まとめ

本稿では，LTE 無線基地局におけるレイヤ 2 信号処理アプリケーションに対して OSCAR 自動並列化コンパイラを用いて並列化を行い，freescale MSC8156 マルチコア DSP 上で性能評価を行った結果を記した．並列性抽出のためのわずかのリストラクチャリングを実施後に並列化したプログラムを MSC8156 上で評価した結果，MAC レイヤ内の処理であるヘッダ解析処理では従来の 1 コア上での逐次の実行に対して，2 コアで 1.51 倍，4 コアで 2.32 倍，6 コアで 3.02 倍の速度向上率が確認できた．また，SCH 終端処理では従来の 1 コア上での逐次の実行に対して，2 コアで 1.56 倍，4 コアで 2.88 倍，6 コアで 3.53 倍の速度向上率が確認できた．さらに，RLC レイヤ内の処理である PDU 受信処理では従来の 1 コア上での逐次の実行に対して，2 コアで 1.69 倍，4 コアで 2.75 倍，6 コアで 3.14 倍の速度向上が確認できた．この結果から，freescale マルチコア DSP 上での LTE 無線基地局信号処理アプリケーションにおける OSCAR 自動並列化コンパイラによる並列化が有用であることが確認できた．

参考文献

[1] Karam, L. J., AlKamal, I., Gatherer, A., Frantz, G., Anderson, D. V., Evans, B. L. et al.: Trends in multicore DSP platforms, *Signal Processing Magazine, IEEE*, Vol. 26, No. 6, pp. 38–49 (2009).
[2] Kasahara, H., Obata, M. and Ishizaka, K.: Automatic coarse grain task parallel processing on smp using openmp, *Workship on Lan- guages and Compilers for Parallel Computing*, pp. 1–15 (2001).
[3] 服部武 and 藤岡雅宣: ワイヤレス・ブロードバンド HSPA+/LTE/SAE 教科書, インプレス R&D (2009).
[4] Ishizaka, K., Obata, M. and Kasahara, H.: Coarse Grain Task Parallel Processing with Cache Optimization on Shared Memory Multiprocessor, *Proc. of 14th International Workshop on Languages and Compilers for Parallel Computing (LCPC2001)* (2001).
[5] Obata, M., Shirako, J., Kaminaga, H., Ishizaka, K. and Kasahara, H.: Hierarchical Parallelism Control for Multigrain Parallel Processing, *Lecture Notes in Computer Science*, Vol. 2481, pp. 31–44 (2005).
[6] Shirako, J., Oshiyama, N., Wada, Y., Shikano, H., Kimura, K. and Kasahara, H.: Compiler Control Power Saving Scheme for Multi Core Processors, *Lecture Notes in Computer Science*, Vol. 4339, pp. 362–376 (2007).
[7] Mase, M., Onozaki, Y., Kimura, K. and Kasahara, H.: Parallelizable c and its performance on low power high performance multicore processors, *Proc. of 15th Workshop on Compilers for Parallel Computing* (2010).
[8] freescale: MSC8156 Reference Manual, http://cache.freescale.com/files/dsp/doc/ref_manual/MSC8256RM.pdf.
[9] freescale: freescale Website, http://www.freescale.com/ja/webapp/sps/site/prod_summary.jsp?codeMSC8156.
[10] freescale: SmartDSP OS API Reference Manual, http://cache.freescale.com/files/soft_dev_tools/doc/ref_manual/SmartDSP_OS_API_Reference_Manual.pdf.