

自動並列化・低消費電力化された複数アプリケーションに対する マルチコア用ダイナミックスケジューリング手法

後藤 隆志[†] 武藤 康平[†] 平野 智大[†] 見神 広紀[†] 高橋宇一郎^{††}
井上 栄^{††} 木村 啓二[†] 笠原 博徳[†]

[†] 早稲田大学理工学術院 〒169-8555 東京都新宿区大久保 3-4-1

[†] 富士通株式会社

E-mail: †{tgoto,kmuto,hirano,hiroki,kimura,kasahara}@kasahara.cs.waseda.ac.jp

あらまし 本稿では、マルチコアを搭載したスマートフォン端末において、コンパイラにより自動並列化及び低消費電力化された複数のアプリケーションを実行する際に、全体の実行時間の短縮あるいは各アプリケーション毎に設定されたデッドラインを守りつつ電力削減を達成するダイナミックスケジューリング方式について提案する。本スケジューリング手法では、コンパイル時に指定した各アプリケーションの並列実行時の利用コア数に応じた実行時間や消費電力、及びデッドラインを用いて、3種類の方式に基づくスケジューリングを行う。ARM 4 コアの端末上で動画コーデックアプリケーションを対象に評価を行い、FIFO 方式と比べ速度向上率で 18.5%、電力削減率で 28.8% の結果が得られた。

キーワード スケジューリング, 並列化アプリケーション, マルチコア, ARM, 低消費電力化, 高速化

Dynamic Scheduling Algorithm for Automatically Parallelized and Power Reduced Applications on Multicore Systems

Takashi GOTO[†], Kohei MUTO[†], Tomohiro HIRANO[†], Hiroki MIKAMI[†], Uichiro

TAKAHASHI^{††}, Sakae INOUE^{††}, Keiji KIMURA[†], and Hironori KASAHARA[†]

[†] Faculty of Science and Engineering, Waseda University 3-4-1 Ookubo, Shinjuku-ku, Tokyo, 169-8555 Japan

[†] Fujitsu Limited

E-mail: †{tgoto,kmuto,hirano,hiroki,kimura,kasahara}@kasahara.cs.waseda.ac.jp

Abstract This paper proposes a dynamic scheduling algorithm for multiple automatically parallelized or power reduced applications on a multicore smart devices to gain higher performance and lower power consumption within the application's deadline. This scheduling algorithm uses the information such as time, power, deadline and number of cores for each application, and is composed of three type of scheduling. Using media codec applications as a benchmark, the proposed scheduling gained 18.5% speedup and 28.8% power reduction compared to FIFO scheduling.

Key words Scheduling, Parallelized Application, Multicore, ARM, Power Reduction, Acceleration

1. はじめに

近年、スマートフォンやタブレットなどのスマートデバイスは急速に普及しており、求められる性能も高くなっている事から、これらの端末に NVIDIA Tegra3 [1], Qualcomm Snapdragon [2], Samsung Exynos [3] などのマルチコアが広く用いられている。このような高性能化に加え、より広い場面でス

マートデバイスが用いられることから、長時間充電無しで利用可能とするための低消費電力化が求められている。

これらのマルチコアに対しては、アプリケーションの並列化を行う事で性能向上を得ることが出来る。また、リアルタイム制約アプリケーションに対しては DVFS 及びクロックゲーティングを行うことでマルチコアを活用した低消費電力化も可能となる [4]。

しかし、これらの並列化されたアプリケーションは、利用するコア数の変化により、処理速度と消費電力値が変わるため、複数の並列化されたアプリケーションを実行する場合には、その時々々の負荷状況や消費電力値を加味したスケジューリングを行う必要がある。そこで本稿では、自動並列化された使用コア数の異なる複数のアプリケーション群に対し、マルチコア上で消費電力の削減と性能向上を得るためのダイナミックスケジューリング手法について提案する。

本稿は、第2章で並列化に用いた OSCAR 自動並列化コンパイラについて述べ、第3章でスケジューリングの対象モデルについて、第4章で具体的なスケジューリング方式について、第5章で評価結果について説明する。

2. OSCAR コンパイラによる高速化・低消費電力化アプリケーション

本章で、スマートフォン上で実行する各アプリケーションを並列化するために使用される OSCAR コンパイラを説明する。

OSCAR (Optimally SCheduled Adavanced multiprocessor) 自動並列化コンパイラは一般的な並列化コンパイラのループのみの並列化のみならず、関数・基本ブロック間の並列性を利用する粗粒度並列化、ステートメントレベルの近細粒度並列化を階層的に利用するマルチグレイン並列化を行う。

OSCAR 自動並列化コンパイラは、逐次プログラムである C コードや Fortran のコードを基本ブロック (BB)、ループブロック (RB)、サブルーチンブロック (SB) で構成される、マクロタスクと呼ばれる粗粒度タスクに分解する。次に、OSCAR コンパイラはこれらのマクロタスクのコントロールフローとデータ依存関係を表現したマクロフローグラフ (MFG) を生成する。その後、コンパイラは MFG から MT 間の並列性を最早実行可能条件解析により抽出した結果をマクロタスクグラフ (MTG) として生成する [5]。

MT がサブルーチンコールやイタレーション間並列化ができないループである場合、OSCAR コンパイラは階層的にその MT の中に MT を生成する。また、ループの並列化可能なループは、ループを複数分割し各分割ループをそれぞれ粗粒度並列タスクとして定義する。これらのマクロタスクは各レイヤの階層型マクロタスクグラフの並列性を考慮して階層的に構成されるプロセッサグループ (PG) に割り当てられる。

MTG が条件分岐を持つ、あるいは実行不確定性が高いプログラム部分には動的スケジューリングするスケジューリングプログラムを生成し、並列化プログラムに埋め込むことにより低オーバーヘッドの動的スケジューリングを可能としている。また、通常コンパイル時には静的スケジューリングにより PG に割り当てられる [6]。

最速実行モードの場合、OSCAR コンパイラはプログラム実行時間が延長しないように、DVFS、クロックゲーティングおよびパワーゲーティングの適用を管理する [7]。同様に、MTG のデッドラインが与えられ、デッドラインまで十分な待機時間がある場合、OSCAR コンパイラは消費エネルギーの合計を最小になるよう、MT に DVFS を適用するか、あるいはデッド

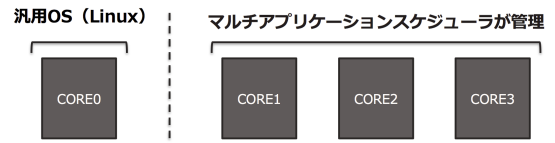


図1 OSCAR マルチアプリスケジューラーの対象モデル

表1 並列アプリ情報の例

PE 数	1PE	2PE	3PE	1PE	2PE	3PE
処理時間 [s]	10.0	6.0	4.0	15.0	15.0	15.0
平均消費電力 [W]	—	—	—	1.2	0.7	0.5
デッドライン時間 [s]	20	20	20	20	20	20

ラインになるまでの待機時間、クロックゲーティングおよびパワーゲーティングを適用する。このモードを本稿ではデッドラインモードと呼ぶ。

動画再生のように、電力最適化された MTG がデッドラインを繰り返し守るような場合、リアルタイム制御モードと呼ぶ。今回の評価する電力削減アプリケーションはリアルタイム制御モードによるものである。

3. 対象とする複数並列化アプリケーションの並列実行モデル

本章では並列化されたマルチアプリケーションの並列実行モデルについて述べる。

3.1 コアパーティショニング

提案するスケジューラは OSCAR コンパイラによって並列化されたアプリケーションを対象としており、OS やその他バックグラウンドプロセスについては Linux によるスケジューリングを適用する。その上で、本稿で提案するスケジューラが意図するスケジューリングを行うために、複数コアのパーティショニングを行い、並列化アプリケーションを動かす。図1はコアのパーティショニングの例である。

図1ではコア1からコア3はスケジューラが管理するコア群とし、コア0には汎用 OS 及びその他通常アプリケーション動作させる。この並列アプリケーション専用コア群の用意により、汎用 OS のバックグラウンドプロセスの割り込みの影響を受けずに OSCAR 並列（電力制御）アプリケーションを実行できる。

3.2 マルチアプリケーションの実行モデル

本スケジューラが対象とするアプリケーションは第2章で述べた OSCAR コンパイラによって自動並列化されたアプリケーションであり、1つのアプリケーションに対して、使用コア数が異なる複数の並列化プログラムの集合となる。これらの各アプリケーションに対し、使用するコア数毎の実行時間や消費電力等の並列アプリケーション情報を用いて、適切なプロセッサへ並列アプリケーションのスケジューリングを行う。

表1は、任意のタスク T_n に対する並列化されたアプリケーション情報の例である。並列化アプリケーション情報は自動並列化を適用する各 PE 数に対して実行処理時間、平均消費電力、デッドライン時間などの情報を持つ。スケジューラは表1で示

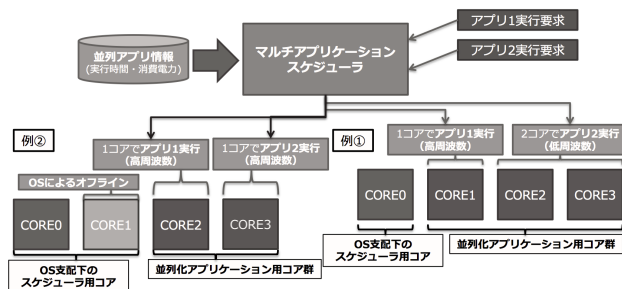


図 2 マルチアプリケーションスケジューラのイメージ図

すような静的な情報を用いてタスクのスケジューリングを行う。これらの情報はスケジューラが起動時に読み込む情報とする。

図 2 はマルチアプリケーションスケジューラの基本的な割り当てイメージを示す。スケジューラはアプリケーションの実行要求が届け次第、並列化アプリ情報を用いてスケジューリングを行う。その際に、実行するアプリケーションの順序や、各アプリケーションの使用コア数を決定し、CPU に割り当てる。例 1 では、アプリ 1 に対して CORE1 を割り当て、アプリ 2 に対して CORE2, CORE3 を割り当てて並列化による電力削減を行う。例 2 の場合は、アプリ 1, アプリ 2 共に 1 コアで実行させている。

4. 提案する OSCAR マルチアプリケーションスケジューラのスケジューリング方式

本章では、提案するスケジューラの実装にあたり、対象モデルとなるコアのパーティショニングの実装について述べる。さらに、並列化された複数コア利用のアプリケーション群に対する具体的なスケジューリング方式として、全体の速度向上を行う応答性重視スケジューリング、デッドラインを守りつつ電力の削減を重視したデッドライン優先低消費電力化スケジューリング、上記 2 方式を負荷量に応じて遷移する複合スケジューリングの 3 方式について説明する。また、提案するスケジューラの実装にあたっては、OSCAR マルチアプリケーションスケジューラとして行った。以降、本スケジューラを OMAS として述べる。

4.1 cgroup を用いたコアパーティショニング

3.1 節で述べた、OMAS が利用するコアのパーティショニングの環境構築にあたっては、Linux カーネルの機能であるコントロールグループ(cgroup)を用いて実現した。具体的には、スケジューラが管理するコア群を OMAS グループとして作成し、管理するコア番号を設定した。その上で、OMAS が対象とするアプリケーションを OMAS グループ上で動作させることで、OMAS が管理するコア上で動作させることができる。その他コアについては、SYSTEM グループとして設定し、既存の Linux スケジューリングの元に OS 及びアプリケーションが実行される。この OMAS 専用コア群の用意により、汎用 OS のバックグラウンドプロセスの割り込みの影響を受けずに OSCAR 並列(電力制御)アプリケーションを実行できる。

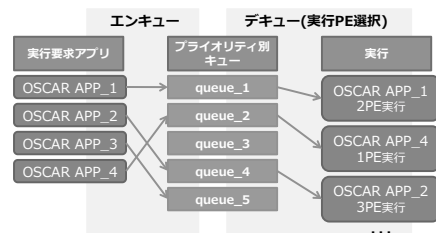


図 3 プライオリティ別キュー

4.2 キューの構造とエンキュー・デキュー

多種類のタスクを扱う場合、スケジューラはどのタスクを優先的に実行するか知る必要がある。そのためタスクにはプライオリティを割り当て、プライオリティが高い順にタスクを選択し、スケジューラが適切なコアグループ割り当てる。方式別でプライオリティの割り振りは異なるが、キューの構造は一環してプライオリティ別キュー (Multilevel Queue) を用いる。図 3 にその構造とエンキュー・デキュー処理フローを示す。

OMAS に対しアプリケーションの実行要求が到着した場合、まずはエンキューとしてこのプライオリティ別キューに挿入される。そして、デキュー処理が行われる際に、対象アプリケーションの並列化アプリケーション群(使用コア数が異なる)からコア数を指定して実行が行われる。

この時、エンキュー処理におけるアプリケーションの対応するプライオリティと、デキュー処理における対象アプリケーションとコア数の決定において、OMAS は方式別のアルゴリズムに沿ってスケジューリング処理を行う。

4.3 応答性重視スケジューリング

応答性重視スケジューリング方式とは、複数のタスクを出来る限り速く処理できるようにスケジューリングする方式である。具体的には、OSCAR コンパイラで並列化されたアプリケーションをタスクとして適切なコアグループに割り当て、全体の実行時間と各タスクの実行時間を短縮させる方式である。本方式は LPT (Longest Processing Time) アルゴリズム [8] をベースとして、各タスクの実行処理時間の情報を用いて複数 OSCAR アプリケーションの効率的な同時実行を実現する。以下に本方式の内容を記載する。なお、タスク T_i は利用 PE 数における実行時間 $t_{i(mPE)}$ を情報として持つものとする。

(1) LPT アルゴリズムに基づき、各タスク情報の 1PE の実行処理時間を比較してプライオリティを決定する。より大きい実行処理時間から順番に高いプライオリティを割り当てる。

(2) 実行要求が来た各タスク T_i に対して、対応するプライオリティ別キュー内に挿入する。

(3) T_i の実行する PE 数を決定する。空いているコアの数で場合分けを行う。

(3-a) 空いているコア数が 2 以上の場合：タスク T_i の PE 数 α の実行処理時間の差 α を計算する。

$$mPE \text{ と } (m+1)PE \text{ の差 } \alpha_m = 1 - \frac{t_{i(m+1)PE}}{t_{i(mPE)}} \quad (1)$$

4.3 節では、全アプリケーションの実行時間を短縮する応答性重視スケジューリングについて、4.4 節では、デッドラインを満たしつつ、消費電力を削減するデッドライン優先低消費電力化スケジューリングについて述べた。本節で述べる複合スケジューリングは、上記 2 方式を状況によりモード遷移しながらスケジューリングする方式である。遷移においては、2 方式毎にキューリストを持たせ、エンキュー先、デキュー先の順番で方式を変えていく。この方式の状態遷移を示したものが図 4 である。まず、初期の状態ではデッドライン優先低消費電力スケジューリングで動作し、低消費電力化を図る。この時、4.4 節で述べた必要 CPU 使用率 α が 1 を超えた場合に、応答性重視スケジューリングモードに遷移する。遷移中は、デキュー先は

表 2 MPEG2 デコーダの並列アプリケーション情報

PE 数	1PE	2PE	3PE	1PE	2PE	3PE
処理時間 [s]	3.99	2.32	1.75	7.5	7.5	7.5
平均消費電力 [W]	—	—	—	0.60	0.46	0.40
デッドライン時間 [s]	20.0	20.0	20.0	20.0	20.0	20.0

デッドライン優先低消費電力スケジューリングのキューから行うが、エンキュー先を応答性重視スケジューリングのキューに変更する。デキュー先のキューが全て無くなった時に応答性重視スケジューリングで動作する。応答性重視スケジューリングにおいて全てデキュー先のキューが無くなり次第、デッドライン優先低消費電力スケジューリングに戻る。この手法により、負荷量が小さく、 α が 1 未満である場合は、デッドラインを守りつつ電力の削減を行うが、 α が 1 を超えた時、つまり負荷量が高くなり、デッドラインを全て満たすことが厳しくなった状態である時に、応答性重視スケジューリングへの遷移を開始することで、出来る限り全てのキューを早く終わらせ、負荷量の削減を行う。

5. OSCAR マルチアプリスケジューラの評価

本章では、第 4 章で提案した各スケジューリングの方式についての評価環境、及び結果について述べる。

5.1 評価端末

本稿では、電力測定が可能なマルチコアでかつ Android プラットフォームで動作する ODROID-X2 を使用した。ODROID-X2 は Samsung の Exynos4412 のチップを積んだ開発ボードである。Exynos4412 は Samsung 社が開発したチップで ARM 社の最大周波数 1.7GHz の Cortex-A9 が 4 つと、1MB の共有 L2 メモリ、2GB のデュアルチャネルの LPDDR2 RAM が搭載されている。ボード改変により、CPU と CPU の動力源コントローラーとして動作をする PMIC(Power Management IC) の間の電力が測定可能である [10]。

5.2 評価アプリケーションと並列アプリ情報

この節では、評価に用いるアプリケーションの概要について説明する。

5.2.1 MPEG-2 Decoder

MPEG-2 デコーダは標準ビデオデコードアプリケーションである。OSCAR 自動並列化コンパイラにより並列化を行い、アプリケーション内デッドライン (1 フレーム毎) を 60FPS (1 フレームあたり 15.0[ms]) として設定し電力削減を適用している。入力ファイルとしては、352x240 の解像度で 450 フレームである。MPEG2 Decoder の並列アプリケーション情報を表 2 に記載する。

5.2.2 MPEG-2 Encoder

MPEG-2 エンコーダは標準ビデオエンコードアプリケーションである。OSCAR 自動並列化コンパイラにより並列化を行った。本アプリケーションに対しては電力制御は適応していない。入力ファイルとしては、352x240 の解像度で 450 フレームである。MPEG2 Decoder の並列アプリケーション情報を表 3 に記載する。

表 3 MPEG2 エンコーダの並列アプリケーション情報

PE 数	1PE	2PE	3PE
処理時間 [s]	22.21	13.35	10.26
平均消費電力 [W]	—	—	—
デッドライン時間 [s]	30.0	30.0	30.0

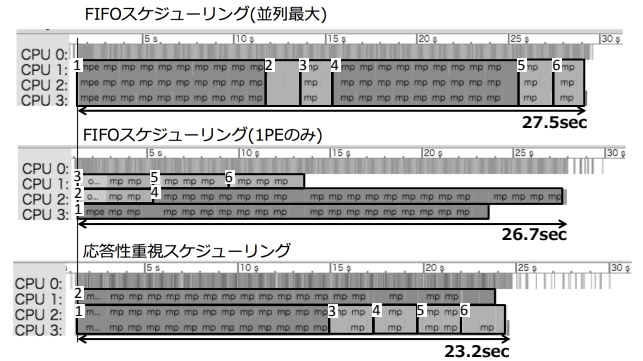


図 5 FIFO 及び応答性重視スケジューリング結果

5.3 応答性重視スケジューリングの評価結果

応答性スケジューリングの評価においては、FIFO スケジューリングと、要求した全アプリケーションの実行時間を比較することで行った。なお、FIFO スケジューリングは最大コア数のみを選択する方式と、1PE のみを選択する方式どちらも計測した。実行要求は、MPEG2Encoder を 1 回、MPEG2Decoder を 2 回、再度 MPEG2Encoder を 1 回、MPEG2Decoder を 2 回の順で同時に入力した。

それぞれのスケジューリング結果は、端末上でアプリケーションが実行している様子を、Android 標準の CPU 負荷分析ツールである systrace を用いることで計測した。結果を図 5 に示す。縦軸は各 CPU コア、横軸は時間を示す。黒線で囲われている部分が対応する CPU で動作する 1 アプリケーションであり、番号がデキュー順を示している。

最大コア数での FIFO スケジューリングでは 27.5 秒、1PE のみの FIFO スケジューリングで 26.7 秒、応答性重視スケジューリングで 23.2 秒となった。また、各スケジューリング方式におけるコアの利用度を算出した結果、それぞれ 100.0%、76.5%、98.4% となった。これらから、応答性スケジューリングにより全コアを活用しつつ、結果として、最大コア数 FIFO スケジューリングと比べて 18.5%、1PE の FIFO スケジューリングと比べて 15.0% の速度向上が得られた。

5.4 デッドライン優先低消費電力化スケジューリングの評価結果

デッドライン優先低消費電力化スケジューリングの評価においては、FIFO スケジューリングと、要求したアプリケーション実行時の消費電力を比較することで行った。実行要求については、応答性重視スケジューリングと同じキューイングで行ったが、負荷量を変化させるためにエンキュー間隔の時間を 6 秒、4 秒、2 秒に変化させて評価した。この時の systrace 結果を図 6 にて示す。

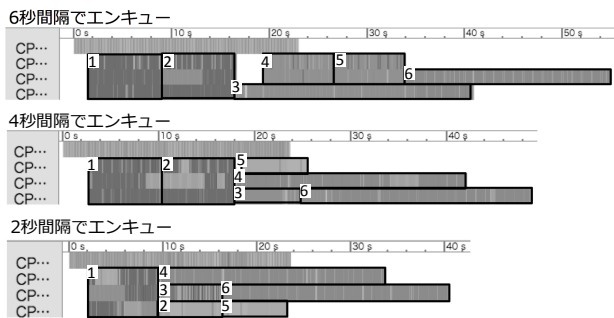


図 6 デッドライン優先低消費電力化スケジューリング結果

6 秒間隔でのエンキューでは、エンキューに余裕があるため CPU 使用率 α も 0.33 未満で推移し、常に可能な限り最大並列数を選択することで電力削減していることがわかる。また、この時デッドラインを満たせている。

4 秒間隔のエンキューでは、序盤は余裕があり 3PE が選択され低消費電力化となったが、後半は必要 CPU 使用率 α が 0.5 を超えたため、デッドラインを満たすため全て 1PE 実行となった。

2 秒間隔のエンキューでは、エンキュー間隔が短く負荷率が高いため、序盤から必要 CPU 使用率 α が 0.5 を超え、1PE で実行されることでデッドラインを満たした。

それぞれのエンキュー間隔における消費電力を、MPEG2Decoder の消費電力値から算出すると、6 秒間隔のエンキューでは 1.71W、4 秒間隔のエンキューで 2.00W、2 秒間隔のエンキューでは 2.20W となる。FIFO スケジューリングで全アプリケーションが 1PE で実行された場合も同様に算出すると、MPEG2Decoder のみで 2.40W となることから、エンキュー間隔に余裕がある場合では、FIFO1PE スケジューリングと比べて MPEG2Decoder について約 28.8% の電力削減が得られた。

5.5 応答性・リアルタイム制約複合スケジューリングの評価結果

複合スケジューリングの評価においては、アプリケーションは MPEG2Decoder に限定した上で、初期段階で 6 回エンキューすることで高負荷状態とし、10 秒毎に 3 回のエンキューをまとめて 3 セット繰り返すことで、合計 15 回のエンキューを行った。

この時の systrace 結果を図 7 にて示す。1～6 までは低消費電力化スケジューリングされるが、この段階で必要 CPU 使用率 α は 1 を超えるため状態遷移に入る。そのため、その後の 7～12 までは応答性重視スケジューリングとなっている。12 までデキューした後は低消費電力化スケジューリングに戻り、13～15 が実行されている。結果として、低負荷時は低消費電力化スケジューリングで行い電力削減するが、負荷が高まりデッドラインを満たすことが厳しくなった場合に応答性重視スケジューリングに遷移することで、キューを早く処理していることがこの結果から確認出来た。

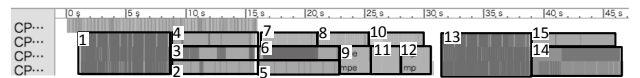


図 7 応答性・リアルタイム制約複合スケジューリング結果

6. おわりに

本稿では、マルチコアスマートフォンを対象に、コンパイラにより自動並列化及び低消費電力化された複数のアプリケーションを実行する際の手法として、並列アプリケーション情報を用いたダイナミックスケジューリング方式について述べた。

本手法を動画コーデックアプリケーションの実行をサンプルとして評価を行い、全体の実行時間及び各アプリケーションの実行時間を短縮させる応答性重視スケジューリングでは、FIFO スケジューリングと比べ 18.5% の速度向上が得られた。低消費電力化デッドライン優先スケジューリングでは、1PE 実行との比較でデコーダアプリに対し 28.8% の電力削減が得られた。また、負荷量により 2 方式を遷移する応答性・リアルタイム制約複合スケジューリングにおいては、低負荷時には低消費電力化、高負荷時には高速化のスケジューリングが行える事が確認できた。

文 献

- [1] NVIDIA Corporation, “Whitepaper Variable SMP (4-PLUS-1) - A Multi-Core CPU Architecture for Low Power and High Performance”.
- [2] QUALCOMM Inc., “Snapdragon S4 Processors : System on Chip Solutions for a New Mobile Age,” 2012.
- [3] Samsung Electronics Co., Ltd., “White Paper of Exynos 5,” April 2011.
- [4] K. Kimura, M. Mase, H. Mikami, T. Miyamoto, J. Shirako, and H. Kasahara, “Oscar api for real-time low-power multicores and its performance on multicores and smp servers,” Languages and Compilers for Parallel Computing, vol.5898, pp.188–202, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010.
- [5] H. Kasahara, M. Obata, and K. Ishizaka, “Automatic coarse grain task parallel processing on smp using openmp,” Languages and Compilers for Parallel Computing, vol.2017, pp.189–207, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2001.
- [6] M. Obata, J. Shirako, and H. Kaminaga, “Hierarchical parallelism control for multigrain parallel processing,” Languages and Compilers for Parallel Computing, pp.31–44, 2005.
- [7] J. Shirako, N. Oshiyama, Y. Wada, H. Shikano, K. Kimura, and H. Kasahara, “Compiler Control Power Saving Scheme for Multi Core Processors,” Lecture Notes in Computer Science, vol.4339, pp.362–376, 2007.
- [8] T.F. Gonzalez, O.H. Ibarra, and S. Sahni, “Bounds for lpt schedules on uniform processors,” SIAM J. Comput., vol.6, no.1, pp.155–166, 1977.
- [9] C.L. Liu and J.W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” J. ACM, vol.20, no.1, pp.46–61, Jan. 1973.
- [10] H. Yamamoto, T. Hirano, K. Muto, H. Mikami, T. Goto, K. Kimura, and H. Kasahara, “OSCAR Compiler Controlled Multicore Power Reduction on Android Platform,” Languages and Compilers for Parallel Computing, pp.155–168, 2013.