Vol.2010-ARC-191 No.2
2010/10/18

OSCAR API標準解釈系を用いたParallelizable Cプログラムの評価

佐藤卓也[†1] 見神広紀[†1] 林明宏[†1]
間瀬正啓[†1] 木村啓二[†1] 笠原博徳[†1]

本稿では種々の組み込みプロセッサ上でOSCAR(Optimally Scheduled Advanced Multiprocessor) コンパイラが並列化したCあるいはFortranプログラムを動作させることを可能とするOSCAR APIを各マルチコア用のライブラリコールに変換するOSCAR API標準解釈系を提案する．このOSCAR API標準解釈系を用いることにより，OSCARコンパイラが出力したプログラムは各コア用のライブラリコール入りCあるいはFortranプログラムになり対象マルチコア内のシングルコア用コンパイラを用いて簡単にバイナリを生成し，各マルチコア上でDMAや電力制御機能を含めて実行することができる．このOSCAR API標準解釈系を用いてOSCARコンパイラにより並列化されたParallelizable Cプログラムの評価を行った．その結果，逐次実行時と比較して，2コア集積のマルチコアであるIBM Power5+ を4基搭載した8コアSMPサーバであるIBM p5 550Qにおいて平均5.61倍，4コア集積のマルチコアであるIntel Xeon 5506プロセッサを2基搭載した8コアSMPサーバにおいて平均4.43倍，SH-4Aコアベースの情報家電用マルチコアRP2の4コアを使用したSMP実行モードにおいて平均3.34倍の性能向上が得られた．

Evaluation of Parallelizable C Programs by the OSCAR API Standard Translator

Takuya Sato,[†1] Hiroki Mikami,[†1] Akihiro Hayashi,[†1]
Masayoshi Mase,[†1] Keiji Kimura[†1]
and Hironori Kasahara[†1]

This paper proposes OSCAR(Optimally Scheduled Advanced Multiprocessor) API Standard Translator. OSCAR API has been developped as an interface between OSCAR compiler, which can parallelize C and Fortran programs, and various embedded multi cores. The OSCAR API Standard Translator translates a parallelized C or Fortran program by OSCAR compiler into a program having runtime library calls for DMA transfer, power control and so on for a target multicore. The parallel processing performance for Parallelizable C programs, which are automatically parallelized by OSCAR compiler, are evaluated on there multicore systems. The evaluation results show that, compared with sequential execution, 5.61 times speedup is achieved on a 8 cores server IBM p5 550Q with 4 dual-core Power5+ processors on average, 4.43 times speedup on a 4 cores server with 2 quad-core Intel Xeon processors on average, and 3.34 times speedup on Renesas/Hitachi/Waseda RP2 with SH-4A cores in SMP execution mode using 4 cores on average, respectively.

1. はじめに

半導体集積度向上に伴った処理性能向上と高い電力効率を実現するため，マルチコアプロセッサが普及している．これらマルチコアプロセッサの有効利用にはプログラムの適切な並列化が必須である．しかし，手動による並列プログラミングは非常に手間がかかり，開発期間の長期化，開発コストの増大へとつながる．そこで，高性能な自動並列化コンパイラの開発が望まれている．このため，筆者らは従来よりOSCAR自動並列化コンパイラを開発している．[1)–3)]

OSCARコンパイラでは，従来のコンパイラで用いられていたループ並列処理に加え，粗粒度タスク並列処理，近細粒度並列処理を組合せたマルチグレイン並列処理を実現しており，プログラム全域にわたる並列化が可能である．またOSCARコンパイラはOSCAR APIにより並列化されたCあるいはFortranを生成可能である．これにより，データのメモリ配置，DMAを用いたデータ転送，電力制御，グループバリア同期，およびリアルタイム制御が可能となり，SMPサーバから情報家電用マルチコアにいたるまで様々なプラットホームでOSCARコンパイラによる自動並列化が適用可能となる．[4)]

OSCAR APIはOpenMP[5)]のサブセットに組込みマルチコアで要求される機能を加えた構成となっており，OpenMPコンパイラを用いてそのままOpenMPサポートマシン上で実行することができる．情報家電向けのマルチコア用のAPI解釈系を用意することにより指示文を各マシン用ランタイムライブラリに変換することでネイティブコンパイラが実行バイナリを生成している．これまでのOSCAR APIを用いたOSCARコンパイラの評

†1 早稲田大学 基幹理工学部 情報理工学科
Dept. of Computer Science and Engineering, Waseda University

価には，情報家電マルチコアでは各マルチコア専用の API 解釈系を用いて評価を行ってきた.[4),6)] しかしながら，より多くのプラットホーム上で OSCAR コンパイラによる並列化や電力制御を利用するために，各々のプラットホームに対して専用 API 解釈系を作成するには時間とコストがかかる．最近では gcc[7)] や ROSE コンパイラ[8)] のように OpenMP が利用可能なオープンソースのコンパイラが存在する．しかしながら，これらのコンパイラの構造を把握し，OSCAR API 指示文を拡張するには多大なコストを要すると予想される．そこで様々なプラットホームで共通に使える OSCAR API 標準解釈系を提案する．

OSCAR API 標準解釈系は API 指示文をランタイムライブラリコールに変換し，これをターゲットプラットホームのランタイムライブラリとともにネイティブコンパイラでコンパイルすることにより，実行バイナリを生成することができる．従来，API 解釈系を作る際には API 指示文の解釈部ならびに C あるいは Fortran のフロントエンドを作る必要があったが，これらの部分はプラットホーム依存度が低い．OSCAR API 標準解釈系としてこれらの部分を用意することにより OSCAR コンパイラ，及び OSCAR API を使用し，簡単かつ開発投資を抑えた新しいマルチコアの開発ができる．本稿では，OSCAR コンパイラによって自動並列化した Parallelizable C[6)] プログラムを OSCAR API 標準解釈系を用いて OSCAR API 指示文をランタイムライブラリコールに変換し，複数のプラットホームで動作することを検証した．

本稿の構成を以下に示す．まず第 2 章では OSCAR コンパイラと OSCAR API について説明し，第 3 章では OSCAR API 標準解釈系の構造及びプログラム変換の様子について述べる．そして，第 4 章では OSCAR コンパイラ，OSCAR API，及び OSCAR API 標準解釈系を用いた評価について述べる．最後に第 5 章でまとめを述べる．

2. OSCAR コンパイラと OSCAR API[1)–4)]

OSCAR コンパイラは C や Fortran 言語で記述された逐次ソースプログラムを入力すると，並列プログラムを自動生成できる自動並列化コンパイラである．

OSCAR コンパイラでは一般的な並列化コンパイラのようにループイタレーションレベルの並列処理を行うのみでなく，ループ・手続き間の粗粒度タスク並列処理，ステートメント間の近細粒度並列処理 を組み合わせたマルチグレイン並列処理，メモリウォール問題に対処するための複数ループにわたるキャッシュあるいはローカルメモリの最適利用を行うデータローカライゼーションが実現されている.[9),10)] さらに，プログラム中の各並列処理部に対する適切なリソース割り当てや，各リソースの周波数・電圧・電源制御による消費電力の自動削減等も実現されている.[11)]

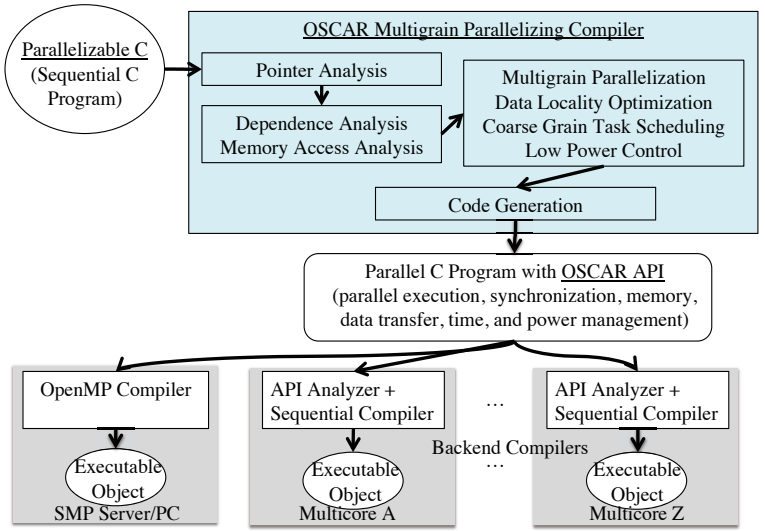

図 1 OSCAR コンパイラと OSCAR API のコンパイルフロー
Fig. 1 Compile flow using OSCAR compiler and OSCAR API

OSCAR API はマルチプロセッササーバで幅広く使われている OpenMP におけるスレッド生成（parallel sections），クリティカルセクション（critical），及びメモリ一貫性維持（flush）の機能に加え，組み込みマルチコアで必要となる電力制御，メモリ管理，メモリ間のデータ転送のような指示文を持っている.[4),12)]

OSCAR コンパイラと OSCAR API のコンパイルフローを図 1 に示す． OSCAR コンパイラにより逐次の Parallelizable C（C において再帰，ポインタ利用に一部制約を加えたもの)[6)] あるいは Fortran プログラムが並列化され，OSCAR コンパイラは OSCAR API 指示文入りのコードを生成する．この並列化コードをサーバ上で動作させるときには OpenMP コンパイラを通すことにより実行ファイルを生成することができる．また，組込みマルチコアで使用する場合には専用 API 解釈系によりランタイムライブラリを含むコードを生成して逐次コンパイラを通すことによっても実行バイナリを生成することができる．

3. OSCAR API 標準解釈系

2 章で述べた OSCAR コンパイラと OSCAR API により，逐次プログラムの並列化や電力制御が自動的に適用できるようになった．しかし，OSCAR API のメモリ配置 API や電力制御 API を利用するにはターゲットとなるプラットホーム用に API 解釈系を用意する必要があり，並列化コンパイラを作るのに比べ極めて低額ではあるがその作成コストがかかる．そこで，プラットホーム依存度が低い指示文やソースプログラムを解釈して変換する部分を OSCAR API 標準解釈系として提供し，多くの組込みプロセッサがあらかじめ持っている，あるいはない場合でも作成が比較的容易なランタイムライブラリをプラットホームごとに用意すれば動作可能とすることでより容易に OSCAR コンパイラによる並列化や電力制御が各マルチコア上でほとんど開発コストなく使用することができる．以下に今回提案する OSCAR API 標準解釈系の構造と変換例について説明する．

3.1 OSCAR API 標準解釈系の構造

OSCAR API 標準解釈系と開発環境の構成を図 2 に示す． OSCAR API 標準解釈系は，OSCAR API を含む C あるいは Fortran プログラムを入力とし，設定ファイルの設定に従いランタイムライブラリ関数を含む C あるいは Fortran プログラムを出力する．

設定ファイルは，アーキテクチャごとの設定が記述でき，分散共有メモリ (DSM) のアドレス，ネイティブコンパイラに対する指示を伝搬可能とする oscar_comment 指示文，データ転送 API 及び電力制御 API の変換，プロセッサコア間で任意のグループによるバリア同期を実現する groupbarrier 指示文，及びモジュール名とモジュール番号の対応付け等に関する設定ができる．

図 3 に設定ファイルの例を示す．この例では情報家電用マルチコア RP2 の設定の一部を示しており，電力制御等で用いるモジュール名に対応するモジュール番号として OSCAR_CPU に 0 を，OSCAR_LDM に 1 を，それぞれ定義している．

出力されたランタイムライブラリ関数を含む C プログラムを各プラットホームでコンパイルし，ランタイムライブラリとリンクすることで実行ファイルを生成することができる．

3.2 ランタイムライブラリコールへの変換

OSCAR API の指示文は C のプログラムでは pragma により記述されるが，これらの OSCAR API の指示文はランタイムライブラリ関数へ変換される．スレッド生成を行う `parallel sections` の変換例を図 4 に示す．

図 4(a) では `parallel sections` 指示文により 4 つのスレッドを生成している．OS-

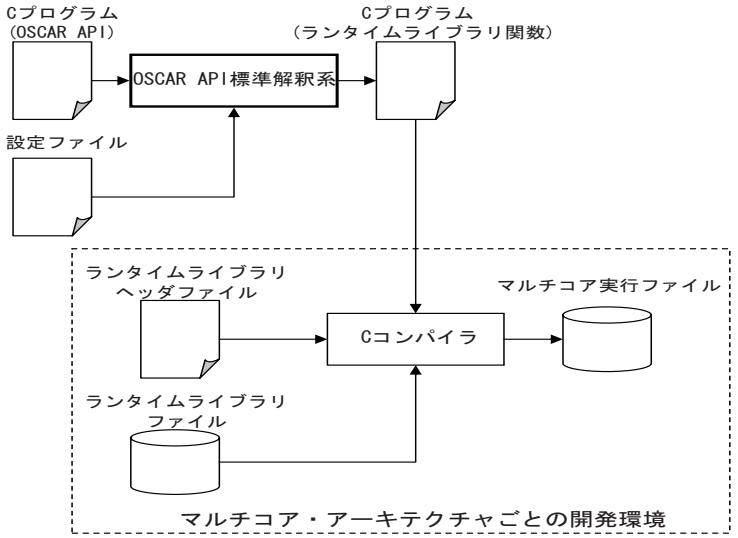

図 2 標準 API 解釈系と開発環境の構成
Fig. 2 Construction of standard API interpretation system and development environment

```
arch:
  -name:RP2
    option:rp2
    module:
     OSCAR_CPU:0
     OSCAR_LDM:1
```

図 3 OSCAR API 標準解釈系設定ファイルの例
Fig. 3 Example of configlation file for the OSCAR API Standard Translation

CAR API 標準解釈系では，スレッドとして実行するプログラム部分を図 4(b) のように `thread_function_00[0-3]` という関数として切り出す．これらの切り出された関数のうち，`thread_function_00[1-3]` は `oscar_thread_create` 関数によりスレッドとして実行し，`thread_function_000` はそのまま通常の関数として実行する．最後に `oscar_thread_join` 関数によりスレッド終了の待ち合わせを行う．

pthread ライブラリが利用可能なプラットホームでは `oscar_thread_create` 及び

```
OMP_NUM_THREADS=4

#pragma omp parallel sections
{
  #pragma omp section
  {
    ...
  }
  #pragma omp section
  {
    ...
  }
  #pragma omp section
  {
    ...
  }
  #pragma omp section
  {
    ...
  }
}
```

(a) OSCAR API 指示文

```
void thread_function_000(void);
void thread_function_001(void);
void thread_function_002(void);
void thread_function_003(void);

int thr1;
int thr2;
int thr3;

oscar_thread_create( &thr1, thread_function_001, 0);
oscar_thread_create( &thr2, thread_function_002, 1);
oscar_thread_create( &thr3, thread_function_003, 2);

thread_function_000();
oscar_thread_join(thr1);
oscar_thread_join(thr2);
oscar_thread_join(thr3);
```

(b) ランタイムライブラリ関数

図 4 OSCAR API のランタイムライブラリへの変換
Fig. 4 Conversion to the run time library of API

oscar_thread_join を pthread_create 及び pthread_join を用いてそれぞれ実装すれば良い．

3.3 メモリ配置属性の指定

thread private 指示文等のメモリ配置指定は，ランタイムライブラリでは実現できない．そのため，これらのメモリ配置指定はネイティブコンパイラ用のメモリ属性指定の記述に変換する必要がある．この変換の様子を図 5 を例として説明する．

```
#pragma oscar onchipshared(var1)
```

(a) OSCAR API 指示文

```
int __attribute((section("OSCAR_SHARED")))var1;
```

(b) gcc 向け変換コード

図 5 onchipshared の変換例
Fig. 5 Conversion example of onchipshared

図 5(a) は onchipshared 指示文により var1 をオンチップ集中共有メモリに配置するように指定している．ネイティブコンパイラとして gcc を用いる場合の変換例を図 5(b) に示す．図 5(b) では__attribute 記述により変数 var1 を OSCAR_SHARED セクションに配置している．対象プラットホーム用リンカの設定ファイルにより OSCAR_SHARED セクションをオンチップ集中共有メモリに配置することで，リンカが変数 var1 をオンチップ集中共有メモリ上に配置する．

4. 性能評価

本章では OSCAR API 標準解釈系が生成したコードが複数のプラットホーム上で動作することを確認する．

4.1 評価方法

OSCAR コンパイラを用いて SPEC2000 より art，equake，SPEC2006 より lbm，hmmer，MediaBench より mpeg2encode および株式会社ルネサステクノロジ提供の AAC エンコード (AACencode) の 6 種類の Parallelizable C コードに対する OSCAR API が挿入された並列化コードを生成した．さらに，これらの OSCAR コンパイラにより自動生成された OSCAR API C プログラムを IBM 8 コアデスクサイドサーバ p5 550Q，Intel 8 コア SMP サーバ Xeon E5506，8 コア組み込みマルチコアルネサステクノロジ／日立製作所／早稲田大学 RP2[13] で OSCAR API 標準解釈系を使用した上で逐次コンパイラにより実行バイナリの生成をする．

得られた実行バイナリを，8 コア搭載の SMP サーバである IBM p5 550Q，Intel の 4 コア CPU である Xeon（Nehalem-EP）を 2 基搭載したサーバ，ルネサステクノロジ，日立製作所，早稲田大学で共同開発した情報家電用マルチコア RP2 における SH-4A を 4 コア構成による SMP モードの 3 つの環境でそれぞれ評価を行った．各環境におけるパラメータ

表 1 評価環境
Table 1 Evaluation environment

System	IBM p5 550Q	Intel Xeon E5506	Renesas/Hitachi/Waseda RP2
CPU	Power5+ (1.5GHz × 2 × 4)	Nehalem-EP (2.13GHz × 4 × 2)	SH-4A (600MHz × 4)
L1 D-Cache	32KB / 1 core	32KB / 1 core	16KB / 1 core
L1 I-Cache	64KB / 1 core	32KB / 1 core	16KB / 1 core
L2 cache	1.9MB / 2 cores	256KB / 1 core	N/A
L3 cache	36MB / 2 cores	4MB / 4 cores	N/A
Native Compiler	IBM XL C/C++ for AIX Compiler V10.1	Intel C/C++ Compiler verion 11.1	SH C Compiler + OSCAR API standard Translation
Compile Option	OSCAR: -O5 -qsmp=noauto	OSCAR: -fast -openmp	

を表 1 に示す．IBM p5 550Q では，1 プロセッサあたり 2 スレッド実行の Simultaneous Multi-Threading(SMT) が可能であるが，本評価では SMT 機能は用いない．

本評価では並列化を実現するために必要なスレッド生成，及び flush 機能についてランタイムライブラリの実装を行った．スレッド生成は pthread ライブラリを利用して実装した．また，flush 機能はインラインアセンブラを利用し，各プラットホームのメモリアクセス順序保証命令に置換した．具体的には Nehalem-EP では mfence，Power5+では sync，SH4 では synco の各命令をそれぞれ使用した．

4.2 評価結果

IBM p5 550Q における処理性能を図 6 に，Intel Xeon E5506 における処理性能を図 7 に，ルネサステクノロジ／日立製作所／早稲田大学 RP2 における処理性能を図 8 にそれぞれ示す．

図中，横軸が各アプリケーションとコンパイル方法を示し，縦軸は逐次コンパイラの 1 コアで実行した場合に対する速度向上率を示す．横軸の各項目内のバーは使用しているプロセッサコア数を示し，それぞれ左から 1 コア，2 コア，4 コア，8 コアとなる．OSCAR API 標準解釈系を使用した場合，IBM p5 550Q では hmmer において 2 コア逐次実行の 1.98 倍，4 コアで 3.90 倍，8 コアで 7.60 倍の速度向上をそれぞれ得ることができた．同様に Intel Xeon E5506 では同じく hanner において 2 コアで 1.56 倍，4 コアで 2.75 倍，8 コアで 7.37 倍の速度向上をそれぞれ得ることができた．平均して，IBM p5 550Q では 8 コア利用により逐次実行の 5.63 倍，Intel Xeon E5506 では 4.58 倍の速度向上をそれぞれ得ることができた．RP2 においても同様に，mpeg2encode では 2 コアで逐次実行の 1.94 倍，4 コアで 3.69 倍の速度向上を得ることができ，平均で 4 コアの利用により逐次実行の 3.34

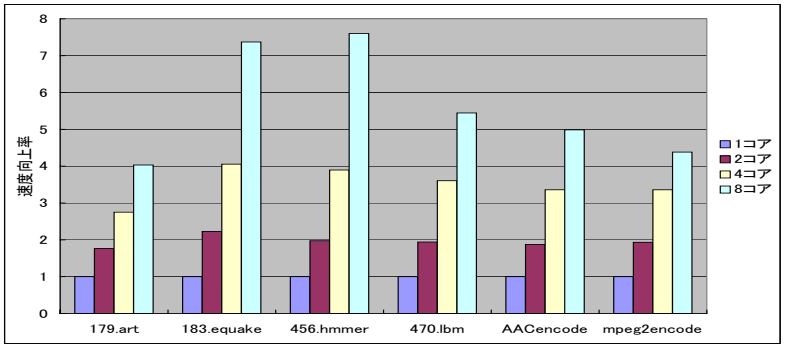

図 6 IBM p5 550Q における標準 API 解釈系の評価結果
Fig. 6 Evaluation results for standard API interpretation system on IBM p5 550Q

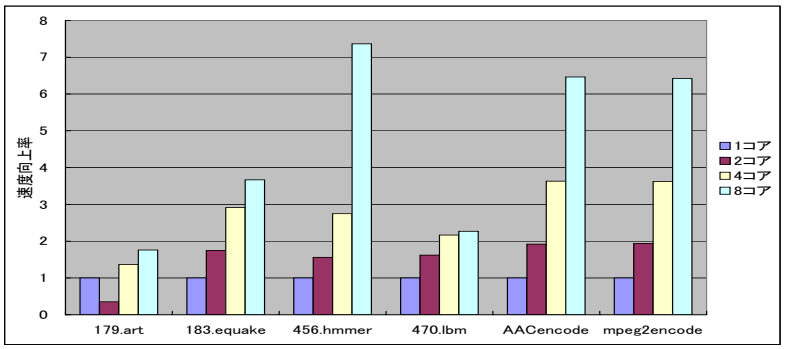

図 7 Intel Xeon E5506 における標準 API 解釈系の評価結果
Fig. 7 Evaluation results for standard API interpretation system on Intel Xeon E5506

倍の性能向上を得ることができた．以上より SMP サーバ上でも組込みマルチコアチップ上でも，OSCAR API 標準解釈系と各プロセッサ用の逐次コンパイラを用いることにより，OSCAR コンパイラの抽出した並列性を利用してコア数の増加に応じたスケーラブルな性能を得られることが確認できた．

この評価により OSCAR API 標準解釈系と各プロセッサ用の逐次コンパイラを用いると

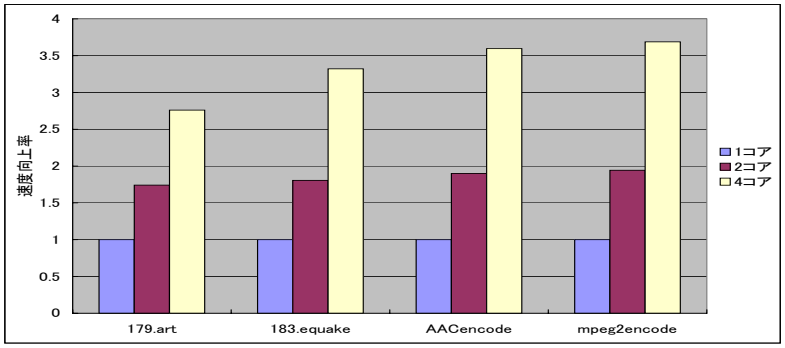

図 8 ルネサステクノロジ／日立製作所／早稲田大学 RP2 における標準 API 解釈系の評価結果
Fig. 8 Evaluation results for standard API interpretation system on Renesas/Hitachi/Waseda RP2

各マルチプロセッササーバ，組込みマルチコアチップ用の並列バイナリを自動生成でき，性能の高い並列処理が行えることが確認できた．

5. ま と め

本稿では OSCAR API の汎用的な解釈系として異なるメーカの SMP サーバ，組込みマルチコア上で OSCAR 自動並列化コンパイラの使用を可能とする OSCAR API 標準解釈系を提案した．複数のプラットホーム上で OSCAR API 標準解釈系を用いて Parallelizable C プログラムの並列化を行い，6 つの C プログラムを 3 種のプラットホーム上で評価をした．評価の結果 SMP サーバである IBM p5 550Q 及び Intel Xeon E5506 及び情報家電用マルチコア RP2 で 6 プログラム平均で 2 コア IBM 1.95 倍，Intel 1.51 倍，RP2 1.85 倍，4 コア IBM 3.50 倍，Intel 2.71 倍，RP2 3.34 倍，8 コア，IBM 5.61 倍，Intel 4.58 倍とコア数の増加に応じたスケーラブルな性能向上を得られることが確認できた．

謝辞 本研究の一部は NEDO“情報家電用ヘテロジニアス・マルチコア技術の研究開発”，および早稲田大学グローバル COE“アンビエント SoC” の支援により行われた．

参 考 文 献

1) Kasahara, H., Obata, M. and Ishizaka, K.: Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP, *Proc. of 13 th International Workshop on Languages and Compilers for Parallel Computing 2000* (2000).
2) Obata, M., Shirako, J., Kaminaga, H., Ishizaka, K. and Kasahara, H.: Hierarchical Parallelism Control for Multigrain Parallel Processing, *Proc. of 15th International Workshop on Languages and Compilers for Parallel Computing* (2002).
3) Kimura, K., Wada, Y., Nakano, H., Kodaka, T., Shirako, J., K.Ishizaka and Kasahara, H.: Multigrain parallel processing on compiler cooperative chip multiprocessor, *Proc. of 9th Workshop on Interaction between Compilers and Computer Architectures (INTERACT-9)* (2005).
4) Kimura, K., Mase, M., Mikami, H., Miyamoto, T., Shirako, J. and Kasahara, H.: OSCAR API for Real-time Low-Power Multicores and Its Performance on Multicores and SMP Servers, *Proc. of The 22nd International Workshop on Languages and Compilers for Parallel Computing* (2009).
5) : OpenMP: Simple, Portable Scalable SMP Programing. http://www.openmp.org/.
6) Mase, M., Onozaki, Y., Kimura, K. and Kasahara, H.: Parallelizable C and Its Performance on Low Power High Performance Multicore Processors, *15th Workshop on Compilers for Parallel Computing* (2010).
7) http://gcc.gnu.org/.
8) http://www.rosecompilre.org/.
9) Yoshida, A., Koshizuka, K. and Kasahara, H.: Data-localization for fortran macro-dataflow computation using partial static task assignment, *Proc. of 10th ACM International Conference on Supercomputing* (1996).
10) 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* (2001).
11) 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 4339* (2007).
12) *Optimaly Scheduled Advanced Multiprocessor Application Program Interface (OSCAR API) version 1.0.* http://www.kasahara.cs.waseda.ac.jp/.
13) Ito, M., Hattori, T., Yoshida, Y., Hayase, K., Hayashi, T., Nishii, O., Yasu, Y., Hasegawa, A., Takada, M., Ito, M., Mizuno, H., Uchiyama, K., Odaka, T., Shirako, J., Mase, M., Kimura, K. and Kasahar, H.: An 8640 MIPS SoC with Independent Power-off Control of 8 CPU and 8 RAMS by an Automatic Parallelizing Compiler, *2008 IEEE International Solid-State Circuits Conference* (2008).