社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－SLDM－124 (1)
2006／3／16

並列化MiBenchを利用したチップマルチプロセッサの評価

田辺 靖貴† 住吉 正人† 天野 英晴†

† 慶應義塾大学大学院 理工学研究科，〒 223-8522 横浜市港北区日吉 3-14-1

あらまし 汎用性，柔軟性に優れた Chip-MultiProcessors(CMP) は近年，組み込み分野でも注目されている．このような CMP では処理を各コアで並列実行する事により，低周波数，低消費電力を実現可能である．組み込み分野で利用されるアプリケーションには画素ブロック毎といったように，データ並列性に優れるアプリケーションが多い．そこで組込み分野のベンチマークである MiBench suite を並列化し，組み込み分野向けの CMP を想定したシミュレータ上でこれを実行した．その結果，CMP において組み込み分野で利用されるアプリケーションを，データ並列性を利用し並列実行した場合に，コア数に応じた性能向上が得られる事が確認できた．本稿では，その結果について報告を行う．

キーワード 組み込みプロセッサ，チップマルチプロセッサ，ベンチマーク，キャッシュコンシステンシ管理

Evaluation of Embedded Chip Multi-Processors using Parallelized MiBench Suits

Yasuki TANABE†, Masato SUMIYOSI†, and Hideharu AMANO†

† Graduate School of Science and Technology, Keio University,
3-14-1 Hiyoshi, Kohoku, Yokohama, Kanagawa, 223-8522 Japan

Abstract Today, even embedded systems are required to process complicated programs with large size data considering both performance and energy consumption. Chip multi-processors (CMPs) have received an attention as one of the most promising candidates. Since a lot of applications used in embedded systems have a large degree of data parallelism, CMPs have a potential to enhance the performance by executing these applications in parallel. In this paper we parallelized three applications from MiBench embedded system benchmark suits using data parallelism extracted from the applications and executed these parallelized applications on a simulator that modeled CMPs. And we demonstrate that CMPs can achieve a good execution performance for the applications used in an embedded system.

Key words Embedded Systems, Chip Multi-Processors, Benchmark, Cache Cosistency

1. はじめに

近年，ホモジーニアスな構成のオンチップマルチプロセッサ (Chip Multi-Processors: CMP) の組込み分野での利用が注目されている．

スケーラビリティに優れるホモジーニアスな構成の CMP は，必要とされる性能に応じ搭載コア数を変更することにより，幅広い製品への対応が可能である．また，並列実行により周波数を上げずに高い性能が得られることから，適切な周波数と電源電圧を設定することで，消費電力と性能を広い範囲で制御することができる．

また，DSP や画像プロセッサなど特定アプリケーション実行用のコプロセッサを搭載したヘテロジーニアスなプロセッサに比べ，汎用性とプログラミング開発環境で優れており，様々なアプリケーションや，機能の追加に柔軟に対応可能である等の特徴も持つ．

ここ数年，半導体メーカー各社は組み込み用途をターゲットとしたホモジーニアスな構成の CMP の利用を実際に始めている．

ARM 社と NEC エレクトロニクスが連携して開発した MPCore [1] は ARMv6 のコア 1〜4 個をシングルチップに納めた，非対称および対称のマルチプロセッシングのどちらにも対応可能な CMP で，民生エレクトロニクス，車載機器，モバイル機器等の高性能・低消費電力製品をターゲットとしている．

富士通の，4 つの 8-Way VLIW のプロセッサコアを搭載したシングルチップマルチプロセッサ FR1000 [2] [3] は，並列デコードにより，約 3W 程度という低消費電力での MPEG-2 MP@HL の復号を可能としている

このような CMP では，いくつものアプリケーションを各コアで個別に実行し，プロセス毎のスループットを向上させる方法と，1 つのアプリケーションの処理を各コアに分散し，並列実行することにより，要求される性能を満たす方法のどちらを

用いることもできる．

このうち，1つのアプリケーションの処理を各コアで並列に実行する場合，その並列実行の方法としては，アプリケーションをタスクに分割し，各コアにタスクを割り当てて実行する方法 (タスク分割) と，画素のブロック毎といったようなデータの並列性を利用し，それぞれ別のデータブロックに対し，各コアで同一の処理を行う方法 (データ並列) の二つが代表的である．

組み込み分野で利用されるアプリケーションでは，画素ブロック毎，暗号処理単位毎などといったように，特にデータ並列性が得やすいアプリケーションが多く，データ並列性を利用しやすい傾向にある．このため，ホモジーニアスな構成の組み込み分野向けの CMP を利用し，データ並列性を利用し効率の良い実行性能を得る事は，多くのアプリケーションにおいて有効であると考えられる．しかし，CMP はサーバ用途のアプリケーションである SPLASH-2 [4] などのベンチマークでは頻繁に評価されてきていたが，組み込み用途のベンチマークでの評価があまり行われていない．これはデータ並列性を利用して並列化された組み込み用途のベンチマークが今まで存在しなかったことが大きい．組み込み用途のベンチマークは，利用メモリ量が制限される等大規模な科学技術計算が多くを占める SPLASH-2 などとは異なった性格があり，プロセッサ台数やスヌープキャッシュプロトコルが性能に及ぼす影響も異なることが予想される．

そこで，本研究では，組込み分野におけるマイクロプロセッサの評価ベンチマーク MiBench suite [5] より，JPEG 圧縮，エッジ検出プログラム，Blowfish 暗号化について，データ並列性を利用した並列化を行った．本稿では，シミュレータと，この並列化したベンチマークを利用し，CMP 構成の組み込みプロセッサにおいて，組み込み分野で利用されるアプリケーションをデータ並列性を利用し実行する場合の実行性能について評価を行った．

以降，本稿では，2. 章において MiBench suite の並列化手法を取り上げる．次に 3. 章で評価環境とシミュレータの説明を行い，4. 章で並列化した MiBench を用いて評価する．最後に 5. 章で結論および今後の課題を述べる．

2. MiBench suite の並列化

MiBench suite は EDN Embedded Microprocessor Benchmark Consortium (EEMBC) をモデルとしたベンチマーク集である．組込み分野で用いられるアプリケーションの多様性を反映し，35 の組込みプログラムを，Automotive and Industrial Control, Consumer Devices, Office Automation, Networking, Security, Telecommunications の 6 カテゴリに分類し提供している．各プログラムのソースコードは標準の C 言語で記述されており，コンパイラサポートのあるプラットフォームであれば容易に使用することが可能である．

本研究では，CMP 構成の組み込みプロセッサで，組み込み分野で利用されるアプリケーションを，データ並列性を利用し各コアで実行した場合の，並列処理性能を検証する事を目的に MiBench suite の提供するベンチマークのうち Consumer Devices のカテゴリから JPEG 圧縮，Automotive and Industrial Control のカテゴリから SUSAN Edges，および Security のカテゴリから Blowfish のプログラムを並列化した．

本章では，並列化の際のプログラミングモデルについて触れ，次に，並列化したプログラムの説明及びその手法を述べる．

2.1 並列化モデル

プログラムの並列化は，今回は，ホモジーニアスな構成の CMP を想定したため，各プロセッサコアから同一のメモリ空間を利用できる共有メモリモデルを用い，2.2 節に述べるように粗粒度の並列性をプログラムから抽出し，並列化を行った．

並列化に際し，自動並列化コンパイラ等を利用する方法もあるが，今回は，できる限り粗粒度の並列性を抽出した場合における並列処理性能を検証する事を目的とし，手作業で並列化を行なうこととした．また，各プロセッサ間の同期には，共有メモリ空間上に設けた同期変数を利用したバリア同期を利用し，排他制御には Test&Set を利用した．

2.2 並列化手法

ここでは，各アプリケーションの動作について説明し，その並列化手法を述べる．

2.2.1 JPEG 圧縮

JPEG 圧縮は，フルカラー画像を対象とした非可逆の画像圧縮アルゴリズムである．入力画像に対して色変換，離散コサイン変換 (DCT)，量子化，エントロピー圧縮の処理を行うことで圧縮する．圧縮の最小単位は MCU (Minimum Coded Unit) と呼ばれ，一連の処理は MCU 毎に繰り返し行われる．

本研究では，JPEG 圧縮の処理を次の 3 つのステージに分け，それぞれのステージに関して並列化を行った．

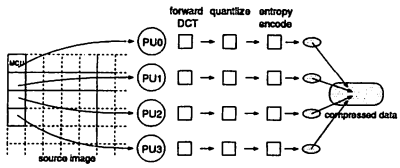

図 1 JPEG 圧縮の並列化

- Stage0 (初期化): 圧縮パラメータの設定や量子化テーブルとハフマン符号化テーブルの生成を行う．生成されるパラメータやテーブルは以後の処理で書き換えられることはないが，圧縮の過程で頻繁に参照されるものであるので，各コア毎にローカルメモリ上に生成する事とした．また，初期化に必要なサイクル数はアプリケーション全体からみると微々たるものであるので，並列化による高速化を行わない事とした．
- Stage1 (圧縮): DCT，量子化，ハフマン符号化を行う．本研究では図 1 のように各コアに異なる MCU を割り当て，MCU 毎に同時に圧縮処理を進めることにより性能向上を図った．単純に並列化すると，DCT で得られたデータの DC 成分を量子化する際に，隣接する MCU の DC 成分との差分を取る必要から待ち合わせが必要となる．そこで，この依存関係による性能低下を避けるため，量子化の時点では DC 成分の差分を取らず，全ての MCU が圧縮された後でこれを解決することにより待ち合わせによるロスを少なくした．
- Stage2 (データの収集): 各コアのローカルメモリ上で圧縮されたデータを，全 MCU の圧縮が完了した後に共有メモリ領域にコピーしてマージすることで，最終的な JPEG 圧縮画像データを出力する．

2.2.2 SUSAN Edges

SUSAN Edges [6] は画像の輪郭抽出を行う，エッジ検出アルゴリズムのプログラムである．このプログラムは，画像の各ピクセルに対して以下の処理を行うことでエッジを検出する．まず，注目するピクセルを囲むように円形のマスクを配置する．そのマスク内で中心の輝度と近いピクセル数を計算する．このピクセル数は USAN (Univalue Segment Assimilating Nucleus) と定義されている．閾値から USAN の大きさを減じるとエッジが強調された画像が得られ，さらにモーメントを算出しエッジ方向を検出し，非極大抑制を行う．

本研究では，処理を次の 4 つのステージ (Stage0〜Stage3) に分け，図 2 に示したように Stage1 と，Stage3 について，画素ライン毎に各コアで並列に処理が行われるように並列化を行った．

- Stage0 (初期化): エッジ検出の準備段階で，共有領域の

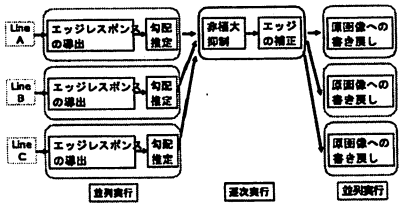

図 2 SUSAN Edges の並列化

確保や USAN 値の算出に用いる重み付けの LUT 生成を行う．LUT の生成は乗算 6 回，除算及び指数関数の演算を 512 回のループで行う．このループ間にはデータ依存がなく容易に並列化可能である．しかし，このステージの実行に要するサイクル数はプログラム全体からみると少ないので並列化による高速化を行わない事とした．

- Stage1 (エッジの強調): このステージで行う計算はマスクごとに独立して算出できることに着目し，入力画像データをコア数で分割して並列処理を行うことによる性能向上を図った．

共有メモリ上の原画像から各画素の輝度データを読み出し，USAN 値の算出やエッジ方向検出及びエッジ強調を行い，共有領域に書き込む．コア間でのデータ依存関係は無いが，共有メモリからの読み出しと書き込みが頻繁に発生する．

- Stage2 (エッジの補正): 検出されたエッジに対して非極大抑制を行い，エッジを補正する．このステージは隣接画素間のデータ依存関係により左上端画素から右下端画素まで逐次的に処理する必要があり，条件によっては既に処理済みのピクセルまで再帰的に戻る場合があるので，並列化を行わず単一コアが処理する事とした．
- Stage3 (原画像へ重ね合わせ): 共有メモリからエッジ画像データを読み出し，それを共有メモリ上にある原画像データ上に書き込むことでエッジ画像を原画像に重ね合わせる．このステージは Stage1 と同様の方法で並列した．

2.2.3 Blowfish encrypt

Blowfish [7] は鍵拡張とデータ暗号化という 2 つの部分からなる．はじめに暗号化に用いるサブ鍵を鍵拡張により生成し，次にブロック単位の平文を繰り返し暗号化する．

blowfish の暗号化には，鍵の利用方法に応じ，いくつかの手法があるが，この手法の一つである ECB モードは，各ラウンドが，そのラウンドの前のラウンドの出力に依存しないという特徴がある．この ECB モードでは，各ラウンド間の依存がないため，平文の各ブロックが独立して暗号化が可能であり，比較的容易に並列処理が行える．このため，本研究でも ECB モードでの暗号化を取り上げ並列化を行った．

本研究では ECB モードを用いた Blowfish の暗号化処理を次のようにステージに分け，並列化を行った．

- Stage0 (鍵拡張): 鍵拡張は Blowfish のアルゴリズムを利用して行われる．必要なサブ鍵を全て生成するためには合計 521 回転するが，サブ鍵の生成は直前に生成したサブ鍵を用いて行う必要があり，並列化による実行時間の短縮が見込めないので並列化しない事とした．また，生成した鍵は暗号化時に頻繁に読み出されるので，高速にアクセス可能なローカルデータとして各コアに持たせる事とした．
- Stage1 (暗号化): データ暗号化は図 3 のように 16 回転する関数からなる．関数 F は図 4 のように入力データを 4 分割し，S ボックスを参照しその出力に対して OR や XOR をとる．暗号化の各ラウンドは鍵依存の転置 1 つと鍵とデータの両方に依存する置換で構成される．共有メモリに置かれた平文に対して，前もって鍵拡張で生成しておいたサブ鍵の配列を用いる．ECB モードでは，各ラウンド間の依存がないため，平文の各ブロックが独立して暗号化が可能である．このため，本ステージでは，平文の各ブロック毎に各コアが並列に処理を実行する事とした．
- Stage2 (データの収集): 各コアがローカルメモリに生成した暗号データを共有メモリ上に集めて暗号文を完成させる．

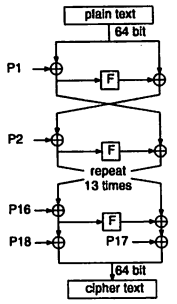

図 3 Blowfish 暗号化

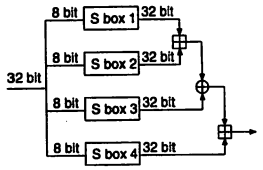

図 4 関 数 F

3. 評価環境

並列化した MiBench を動作させる環境として，多くのコアを搭載した場合の性能評価を可能とし，アプリケーション実行時の挙動を詳細に解析するため，シミュレータを実装した．シミュレータは，ホモジーニアスな構成の CMP として，一般的なアーキテクチャを想定し，ISIS-SimpleScalar [8] を利用して実装した．本章では，この ISIS-SimpleScalar およびこれを用いて構築したシミュレーション環境について述べる．

3.1 ISIS-SimpleScalar

ISIS-SimpleScalar は，主として並列計算機をターゲットとした計算機シミュレータのための C++言語によるクラスライブラリである ISIS に，SimpleScalar が提供する out-of-order 実行，分岐予測などをサポートした sim-outorder モデルを組み込み，マルチプロセッサシステムに対応させたシミュレーション環境である．C++を採用したことによりシステム全体をクラス単位で効率良く管理でき，しかも C 言語と同様に高速に動くプログラムが容易に作成可能である．ISIS では計算機内部のプロセッサやメモリなどの機能ブロックがユニットと呼ばれるクラスライブラリとして実装されている．

3.2 ターゲットアーキテクチャ

バス結合型は最も基本的なマルチプロセッサアーキテクチャであり，現在開発されている組み込み CMP の多くもバス型を採用している．そこで，今回の評価でも図 5 のようにバスにより，各コアおよび，オンチップの RAM が接続された CMP をモデルとして，ISIS-SimpleScalar を用いてシミュレータを構築した．

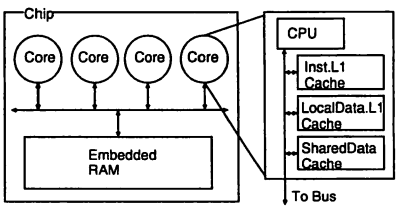

図 5 シミュレーションターゲットアークテクチャモデル

各コアは SimpleScalar の sim-outorder モデルを組み込みマルチプロセッサシステムのシミュレーションに対応させたモデルを用い，図 5 のような構成となっている．プロセッサコアの命令発行幅は最大 4 命令同時発行に設定した．また，プロセッサコアの動作周波数については，組み込み分野で用いる場合には，低消費電力化のため負荷に応じて周波数を変更したりする

場合があるが，今回は可変とはせず，全てのコアが，600MHz で駆動される事とした．

ローカルデータおよび命令は各コア毎の L1 および，オンチップ RAM の領域を利用した L2 キャッシュでキャッシュされる事とした．L2 でミスした場合には，オフチップのメモリにアクセスされる事とし，ローカルデータへのアクセスレイテンシには，表 1 に示したレイテンシを用いる事とした．

共有データ用の領域としては，各コアからバスを介してアクセスされる共通のオンチップ RAM 領域を利用する事とし，本稿の評価では，評価条件を簡略化するため，共有データはすべて，この RAM 領域に格納できる事とした．また，各コアは，バスをスヌープすることによりコンシステンシ管理された共有データ用のキャッシュを持つ事とした．共有データアクセスのレイテンシは表 1 に示した通りである．ただし，バスにて衝突が発生した場合等は，それを考慮して，より多いレイテンシとなる．

キャッシュ間のコンシステンシ管理の手法としては，更新されたデータのメモリへの反映方法に着目し，共有データが更新されるたびにオンチップ RAM 上のデータも更新され，他のコアが保持しているキャッシュラインは無効化される Write-through 方式，および，MESI プロトコルを利用し，更新は各コアのキャッシュ上で行われ，オンチップ RAM へのデータの反映はキャッシュからラインが追い出される場合と，他のキャッシュがラインを更新する前に行われる Write-back 方式の 2 通りを実装し，評価に用いた．

各コア，およびオンチップの RAM を接続するバスは，チップ内にわたるグローバルな配線となる事を考慮し，プロセッサコアの周波数の半分の 300MHz で動作する事とした．ただし，M32R [9] のようにパイプライン構成のバスを用いる事を想定し，アービトレーション，スヌープ，データアクセス，データ転送を行う 4 つのステージのパイプライン構成を取り，クロック毎に 1 つの要求を受け取れる事とした．

また，各コア間の同期のための変数および，Test&Set 用の変数は，各コアでキャッシュ可能とし，変数を更新したコアからのバスへ無効化要求を発行する事により，各コアへ変更が通知されるようにし，同期変数の監視のために，不必要なバスアクセスを発生させないようにした．

表 1　シミュレーション環境

CPU Core	Clock Rate	600 MHz
	Issue Width	4 inst/cycle
Local Inst & Data Cache	L1 Hit Latency	1 cycle
	L2 Hit Latency	6 cycle
	L2 Miss Latency	18 clcye
	L1 Size	128 Byte
	L2 Size	1024 Byte
	Line size	32byte
Shared Data Cache	L1 Hit Latency	1 cycle
	L1 Miss Latency	8 cycle
	L1 Size	8192byte
	Line size	16byte

4. 評　　価

ベンチマーク実行時の各データサイズは表 2 に示した．評価は入力データが共有メモリにセットされた状態から開始した．

前節に述べた通り，共有データキャッシュのスヌーププロトコルは、Write-through 方式と，MESI プロトコルの Write-back 方式の二種類を比較したが，参考のため，各コアで共有データをキャッシュしない場合についても評価を行った．

表 2　入力データサイズ

Application	Format	Size
JPEG 圧縮	8bit Color	512 x 512 pixel
SUSAN Edges	8bit Gray	384 x 288 pixel
Blowfish encrypt	ascii text	19488 Byte

4.1　並列実行時の性能評価

図 6 は JPEG 圧縮について，アプリケーションを繰り返し実行した場合に，1 秒間に何枚の画像の圧縮が可能かをスループットとして，利用するコア数毎，キャッシュ管理プロトコル毎に表わしている．

同様に図 7，図 8 は，SUSAN Edges と，Blowfish でのスループットを表わしている．Blowfish では，1 秒間に何 Byte の入力データを暗号化できるかでスループットを示している．

4.1.1　利用コア数による性能向上

図 6〜図 8 について，利用コア数を増やした場合の性能向上を見ると，どのアプリケーションでも，コア数に応じた良好な性能向上が得られている．

組み込み分野で利用されるアプリケーションでは，処理単位毎に，データ並列性が得られる場合が多い．今回，並列化したアプリケーションでも，JPEG 圧縮では MCU 毎に DCT，量子化，ハフマン符号が行え，SUSAN Edges では画素ライン毎の並列性が抽出でき，Blowfish の ECB モードでは，暗号化を行うブロック毎の並列性が抽出できた．

また，コア間の同期や排他制御も，JPEG 圧縮における圧縮されたデータのメモリへの書き出しといったような一部の例外を除き，DCT や量子化などといったように，比較的大きな処

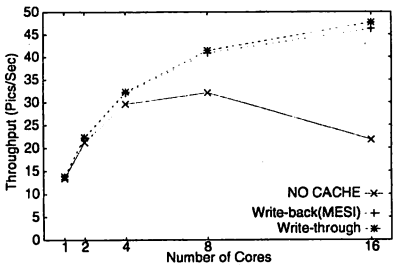

図 6　JPEG 圧縮 並列実行性能

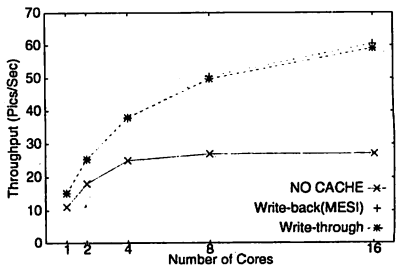

図 7　SUSAN Edges 並列実行性能

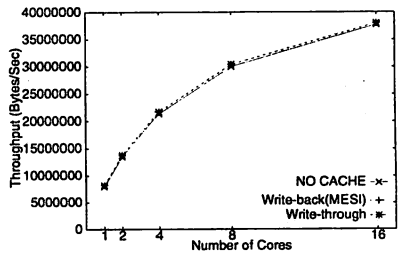

図 8　Blowfish 並列実行性能

理単位毎で済む場合が多く，科学技術計算と違い，同期や排他制御が性能を低下させる要因とはなりにくかった．

このように，理想的なデータ並列性の抽出が可能であり，同期の回数を極端に増すことなく並列化が可能であったため，並列化を行った処理については，ほぼ理想的なスケーラビリティが得られ，コア数に応じた性能向上が得られた．

4.1.2 共有データキャッシュの有無による性能比較

共有データキャッシュを利用した場合と，利用しなかった場合について比較を行うと，JPEG 圧縮，SUSAN Edges ともに，キャッシュを利用した場合には，コア数を増やした場合も性能の向上が得られているが，キャッシュを利用しない場合には，8 コアで性能向上が頭打ちとなってしまっている．特に，SUSAN Edges では，複雑な計算を必要とせず，計算量に対し，共有データアクセスの割合が多いためキャッシュが性能の向上に有効であった．

Blowfish では，今回は ECB モードについて並列化しており，ECB モードでは暗号化ブロック間での依存が無く同期操作以外では共有するデータが無いため，共有データキャッシュの有無は性能に影響を与えなかった．

4.1.3 Write-back 方式と Write-through 方式の比較

今回の評価では，Write-back 方式と，Write-through 方式で比較を行うと，この 2 つでは性能には大きな差が示されなかった．これは，従来のマルチプロセッサシステムにおいて，科学技術分野や，サーバ分野のアプリケーションを実行した場合に Write-back 方式の優位が報告されているのと比べ，特異な点であると言える．

この理由の一つとして，オンチップのメモリを共有データに利用しているため Write-thorugh 方式と Write-back 方式を用いた場合でもキャッシュミス時のレイテンシには差がない点が挙げられる．ボードレベルでプロセッシングノードを接続したようなマルチプロセッサシステムでは，MESI プロトコルを用いてキャッシュ間のラインの転送を行う事により，キャッシュミス時のレイテンシを抑え，性能を向上させる事ができる．しかし，今回は各コアのキャッシュも，キャッシュミスした際にアクセスするメモリも，どちらもオンチップに搭載しており，キャッシュ間転送を行う場合でも，オンチップのメモリからキャッシュラインを取得する場合でも，レイテンシは変わらない．このため，この 2 つの方式で大きな性能差が生じなかった．

ただし，キャッシュ間転送時のレイテンシとオンチップのメモリとのレイテンシに差がある場合でも，オフチップの共有メモリにアクセスするような従来のマルチプロセッサシステムと比べ，CMP では，このレイテンシの差はあまり大きくはならないので，Write-through 方式を利用したとしても性能を大きく低下させる事はないと考えられる．

また，Write-through 方式を用いた場合，書き込まれたデータを共有メモリに反映するために頻繁にメモリアクセスが発生し，性能を低下させる可能性がある．

表 3 は，JPEG 圧縮実行時にバスに発行された Read 要求と，Write 要求の数を，各ステージ毎に比較している．実際に，Write-through 方式を利用した場合の Write 要求数の増加は，各コアが個別に処理していたデータを書き戻す Stage2 で顕著である．しかし，今回はオフチップのメモリと比較して高速にアクセス可能なオンチップのメモリに共有データが格納されるため，共有データの更新が頻発しても性能に与える影響はそれほど多くなく，この Stage2 でも，実行サイクル数で比較すると，7%程度の増加となっているだけであった．

このように，組み込み分野で利用されるアプリケーションを，共有メモリ領域への書き込みのコストが低い CMP において実行する場合には，複雑でなプロトコルを用いずとも，シンプルなプロトコルを利用し，充分な性能を得られる可能性を示した．

表 3 バスへ発行された Read, Write 要求数の比較:JPEG 圧縮 8PU

実行	Write-back		Write-through	
ステージ	Read	Write	Read	Write
Stage0	0	1,048	0	1,054
Stage1	393,224	2,315	393,224	6,117
Stage2	10,935	3,908	10,934	32,898

もっとも，今回の評価は 3 つのアプリケーションにおいて行ったのみであり，画像や動画の伸長などのように，出力データが膨大な場合や，共有データがオンチップメモリに収まらない場合等，Write-through 方式を用いていると性能低下を招くケースも考えられ，今後も検証を行っていく必要はあると言える．

4.2 タスク分割による並列実行との比較

4.1 節では，データ並列実行時に利用するコアを増やすことにより性能向上が得られる事を示した．

1. 章に述べたように，1 チップに複数搭載されるコアの利用方法としては，アプリケーションをタスクに分割し，各コアに分割したタスクを割り当てる方法もある．そこで，データ並列性を利用して並列実行する場合との，比較対象として，JPEG 圧縮について，このタスク分割による並列実行も行った．

タスク分割により実行する場合，一番処理に時間のかかるタスクがボトルネックとなり，利用するコア数を増しても性能が向上しない．もちろん，タスクを，さらに細分化する方法や，タスク内のデータ並列性を利用し複数のコアでタスクを並列実行し，要求される性能を満たす方法も考えられる．しかし，今回は，このような方法を用いる事は，想定せず，全部で 4 つのコアのみを実行に利用する事とし，最も処理量の多いタスクを 1 つのコアに割り当て，その他のタスクを 3 つのコアに分散させる方法を取った．

表 4 は，実際に JPEG 圧縮でどのようにタスクを各コアに割り当てたかを示している．各タスク間でのデータ受け渡しは共有メモリを介して行う事とした．

4.2.1 スループットの比較

図 9 は，キャッシュ利用時に，タスク分割により並列実行した場合と，データ並列性を利用して実行した場合を比較している．図 6 と，同様に，横軸には利用コア数を取り，縦軸にスループットを示しているが，タスク分割時は，最も処理量が多い DCT 変換を行うタスクがボトルネックとなりコア数を増しても性能は変わらないため，4 つのコアで実行した場合のスループットについてのみ示している．

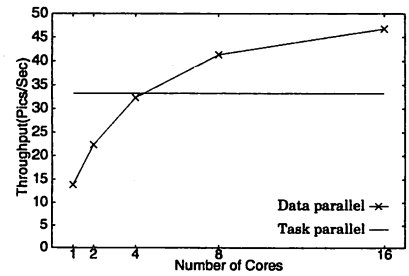

図 9 タスク分割実行とデータ並列実行の比較

タスク分割およびデータ並列で実行した場合と比較すると，4 コア利用時では，僅かにタスク分割の場合のほうが高いスループットを示し，データ並列を利用した場合にはコア数を増やす事によりタスク分割を上回る性能実現している．したがって，このアプリケーションでは，データ並列性を利用した並列処理を用いた方が，利用コア数を増す事で，要求する性能に柔軟に対応することが可能であると言える．

4.2.2 タスク毎の必要サイクル数での比較

表4は，どのように各コアにタスクを割り当てたかを示すと同時に，分けられた各タスクを実行するのにタスク分割時に何サイクルを必要としたか，および，そのタスクと同様の処理を行うのに，4つのコアでデータ並列性を利用し実行した場合に何サイクルを必要としていたかを比較して示している．ただし，タスク分割時は，各コアは各タスクの処理以外にもタスク間の通信も行う必要があり，データ並列実行時には，各コアが処理を行う画素ブロック全てについて，AC成分の符号化を行ってから同期を行うといったように，タスク間通信や，同期のために必要なサイクル数を含めて比較を行うと必ずしも正当な比較とならない．このため，ここでは，このような，同期や通信のために必要なサイクル数は除外している．

表4 JPEG圧縮のタスク分割

Task	タスク分割 実行時 (cycle)	データ並列 実行時 (cycle)
色変換とサンプリング	8,138,708	8,218,643
DCT	14,686,085	3,658,911
量子化	6,818,592	1,635,640
ハフマン符号化	6,739,602	2,266,940

表4で，タスク分割実行時とデータ並列実行時の，各タスク毎の必要サイクル数を比較すると，色変換とサンプリングでは，今回は，MCU単位の並列化を行ったため，MCU生成前の色変換についてはデータ並列性を利用した並列化を行っていない．このためデータ並列実行時とタスク分割実行時で性能にはそれほど差がない．DCT，量子化については，データ並列性が理想的に抽出でき各コアが個別に動作する事ができるため，タスク分割時と比較し，ほぼ利用コア数に応じた性能の向上を得られている．ハフマン符号化については，データ並列実行時にはデータの並び順に共有メモリへの書き込みを行う必要があるため，利用コア数に比例した性能向上とは行かないものの，タスク分割時の約1/3のサイクル数で処理を行えた．

このように，データ並列性を利用し並列実行する場合には，データの共有関係によっては性能向上が妨げられる場合があるが，今回並列化を行った部分については，ほぼ理想的な並列性が抽出でき，タスク分割と比較しても各タスクを効率良く実行できた．

また，タスク実行時に全体の性能を決定する最も処理量の多いタスクは，DCTだった．今回，タスク分割実行時と，データ並列実行時のスループットを比較すると，僅かに，タスク分割実行時のスループットがデータ並列実行時を上回っていた．このため，他のタスクについてはデータ並列実行を行わずとも，DCTについてのみデータ並列性を利用して並列実行を行うことによっても，性能向上を実現できる可能性がある．

5. 結論および今後の課題

本研究では，組込み分野におけるマイクロプロセッサの評価ベンチマークMiBench suite[5]より，JPEG圧縮，エッジ検出プログラム，Blowfish暗号化について，データ並列性を利用した並列化を行い，ホモジーニアス構成のCMPを想定したシミュレータ上で，これを動作させ評価を行った．

その結果，今回は3つのアプリケーションにいて並列化を行ったのみではあるが，組み込み分野のアプリケーションでは比較的，データ並列性に優れており，並列実行によりコア数に応じた性能向上が得やすく，利用コア数の変更により要求される性能に柔軟に対応できる事を示した．

また，今回の評価では，比較的高速なオンチップのメモリを利用可能と想定したため，Write-through方式が，Write-back方式と比較しても遜色の無い性能を示した．この結果は，CMPを組み込み分野のアプリケーションに利用する場合には，複雑なキャッシュ管理プロトコルを用いずとも，シンプルなキャシュコンシステンシ管理プロトコルで充分な性能が得られる可能性を示したと言える．

アプリケーションをタスクに分割し，各コアに割り当てて実行した場合において，各タスクを処理するのに必要としたサイクル数を比較し，データ並列性を生かす事により，1つのコアのみでタスクを実行した場合と比較し，効率良くアプリケーションを実行できる事を示した．

今後の予定としては，今回は，JPEG圧縮等の基本的なアプリケーションを選択し並列化を行ったが，それ以外のアプリケーションについて並列化を行う事が挙げられる．特に，処理量が多い動画圧縮伸長などについてなども取り挙げ並列化を行っていきたい．また，今回の評価では，Write-through方式がWrite-back方式と比較して遜色無い結果を示していたが，その他のアプリケーションでも性能低下を招く事がないのか検証を行っていきたい．さらに，タスク分割実行時に，処理に時間のかかるタスクについてデータ並列性を利用した並列化を行い，タスク分割とデータ並列性を利用した並列化を組み合わせ，より効率の良い性能向上が得られるか検証する事なども挙げられる．

そして，今回並列化したベンチマークも含め，将来的には，並列実行性能の検証用の組み込み向けベンチマークとしての公開を検討して行く予定である．

文　献

[1] ARM. Arm developers' guide (arm11 mpcore multiprocessor semiconductor).

[2] T. Shiota, K-I. Kawasaki, Y. Kawabe, W. Shibamoto, A. Sato, T. Hashimoto, F. Hayakawa, S-I. Tago, H. Okano, Y. Nakamura, H. Miyake, A. Suga, H. Takahashi Fujitsu, and Kawasaki. A 51.2GOPS, 1.0GB/s-DMA Single-Chip Multi-Processor Integrating Quadruple 8-Way VLIW Processors. In *ISSCC Digest of Technical Papers*, pp. 18–19, 2005.

[3] 須賀敦浩. 雑誌 FUJITSU, 第56巻, 特集 FR-V 用1チップマルチコアプロセッサ, pp. 292–298. 富士通株式会社 知的財産権本部 情報部 技術開発支援部, 7月 2005.

[4] S.C.Woo, M.Ohara, E.Torrie, J.P.Singh, and A.Gupta. The splash-2 programs: Characterization and methodological considerations. In *Proceedinfs of the 22nd International Symposium on Computer Architecture*, pp. 24–36, 1995.

[5] Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, and Richard B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proc. of IEEE 4th Annual Workshop on Workload Characterization, Austin, TX*, December 2001.

[6] Stephen M. Smith and J. Michael Brady. Susan - new approach to low level image processing. *Int. J. Comput. Vision*, Vol. 23, No. 1, pp. 45–78, 1997.

[7] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption, Cambridge Security Workshop*, pp. 191–204, London, UK, 1994. Springer-Verlag.

[8] 薬袋俊也, 塙敏博, 田辺靖貴, 天野英晴. Isis-simplescalarの実装. 情報処理学会研究報告, 2004-ARC-160, pp. 29–34, December 2004.

[9] Satoshi Kaneko et al. A 600 mhz single-chip multiprocessor with 4.8 gb/s internal shared pipelined bus and 512 kb internal memory. In *ISSCC Digest of Technical Papers*, pp. 254–255, 2003.