計算機アーキテクチャ 147−31
ハイパフォーマンス 89−31
コンピューティング
（2002. 3. 8）

# MediaBenchにおけるマルチグレイン並列性の解析

櫻澤 秀樹† 岩井 啓輔†† 阿部 剛‡ 天野 英晴‡

†日立製作所 ††防衛大学校 ‡慶應義塾大学理工学部

現在、慶応義塾大学天野研究室ではマルチグレインでの並列処理を目的としたマルチプロセッサシステムであるASCA(Advanced Scheduling oriented Computer Architecture) の設計が行われている．従来ASCAシステムの性能評価には科学技術計算を対象としてきたが，今後の評価にはメディア処理も考慮にいれている．そのため今回はメディア処理での代表的なベンチマークプログラムであるMediaBenchの解析を行い、メディア処理でのマルチグレイン並列処理の有効性の分析を行う．

# Analysis of Multigrain Parallelizm in MediaBench

Hideki Sakurazawa† Keisuke Iwai†† Tsuyoshi Abe‡ Hideharu Amano‡

†Hitachi, Ltd. ††National Defence Academy ‡Keio University

Currently, a design of the multiprocessor system ASCA purposes in parallel computation in Multigrain in Keio University Amano laboratory. We have been intending Science calculation for the performance evaluaion of an ASCA system, but will take media processing into account in future evaluation. Therefore we analyze MediaBench which is a representative benchmark program by media processing this time and inspect the effectiveness of Multigrain parallel computation by media processing.

## 1 はじめに

最近のLSI技術の発展により，マルチメディア処理用システムLSIにもかなり大規模な並列処理機能を搭載することが可能となった．従来，マルチメディア処理用LSIは一部にSIMD による大規模並列性の利用が可能であったが，主として命令レベル並列性(Instruction Level Parallelism)を利用する方式が多かった．これは，従来のマルチメディア用システムLSIが，チップ自体に比較的小規模な並列性を利用する機能以外持ち合わせていなかったことが一つの原因である．一方で，大規模並列処理用科学技術計算機用に並列性を抽出し，利用する技術が開発されている．このうちの一つの手法であるマルチグレイン並列処理[2]は，対象となるプログラムの様々なレベルの並列性を抽出して，利用することが可能な方法であるが，今まで大規模な科学技術計算にのみ適用されてきた．

そこでマルチメディアアプリケーション用ベンチマークであるMediaBenchにマルチグレイン並列処理の手法を試みる場合，どの程度有効であるかを調べるため，本論文ではそれぞれのベンチマークに対し特に粗粒度，中粒度での並列性に焦点を当て分析を行った．

## 2 マルチグレイン並列処理手法

マルチグレイン並列処理手法とは，疑似代入文ブロック(BPA)やループ(RB)，サブルーチン(SB)間の並列処理手法である粗粒度並列処理手法（マクロデータフロー処理手法）[3]，ループレベルの並列処理である中粒度並列処理手法，BPA内部の並列性を利用する近細粒度並列処理手法[4]を階層的に組み合わせて効率良く並列処理を行なう手法である．以下

それぞれのレベルの並列処理手法について述べる．

### 2.1 マクロデータフロー処理(粗粒度並列処理)

粗粒度並列処理では，プログラムはマクロタスク(MT)に分割される．MTはブロック途中への飛び込みが無いステートメントの集合からなるブロックである．その形状からBPA，RB，SBに分けられる．

ここで生成されたMTはプロセッサエレメント（PE）のグループであるプロセッサクラスタ（PC）に割り当てられ，それぞれのMT間の並列性を利用し，マクロデータフロー処理が行なわれる．

階層的にMTを生成した後，各階層でMT間のコントロールフロー解析，データフロー解析を行ない，その結果からマクロフローグラフ(MFG)を作成する．MFGではMT間の依存関係を表しているが並列性は表していない．MFGからMT間の並列性を抽出するために，コントロールフロー，データフローを考慮に入れ各マクロタスクがもっとも早く実行可能になる条件(最早実行可能条件[6])を求める．それを示したものがマクロタスクグラフ(MTG)である．このMTGを用い条件分岐などの実行時不確定性の存在する階層ではダイナミックスケジューリング，それ以外ではスタティックスケジューリングにより各MTをPCに割り当て実行していく．

### 2.2 ループ並列化(中粒度並列処理)

PCに割り当てられたMTがDo-all可能なRBである場合，このRBはPC内のプロセッシングエレメント(PE)によって中粒度すなわちイタレーションレベルで並列化される．

### 2.3 近細粒度並列処理

MTがBPAやループ並列化できないループであった場合，MT内部はステートメントレベルで並列化され，PC内部のPEによって並列処理される．

このような近細粒度タスク集合のPC内PEへの割り当てには，データ転送時間を考慮したヒューリスティックアルゴリズムであるCP/DT/MISF法あるいはDT/CP[6]法を用いる．これはコンパイル時にスケジューリングを行なうスタティックスケジューリングである．

## 3 Mediabench Conponents

Mediabench1.0はC言語で記述された19のアプリケーションから構成される．これらのアプリケーションは全て公に使用可能であり，汎用的なプロセッサで広く使用されている．表1に各アプリケーションの概要をまとめる．

表 1: MediaBench Conponents

| application | 内容 |
|---|---|
| ADPCM | 音声をデジタルデータに変換する方式の一つ． |
| G.721 | CCITTにより規定された32kbpsADPCM． |
| EPIC | 画像データの圧縮方式の一つ． |
| PEGWIT | 公開暗号鍵を用いた暗号化ツール． |
| MPEG2 | 映像データの圧縮方式の一つ． |
| GSM | デジタル携帯電話に使われている無線通信方式の一つ． |
| PGP | RSA，IDEAを使った電子メール暗号化ツール． |
| Ghostscript | Postscriptに関するフィルタ系プログラム． |
| JPEG | 静止画像データの圧縮方式の一つ． |
| Mesa | OpenGLに近い3Dグラフィックスライブラリ． |
| RASTA | 音声処理プログラム． |

## 4 解析の手順

マルチグレイン手法での並列性の抽出を行う際に行った手順について述べる．解析に用いたマシン環境はSun Ultra enterprise 450，OSはSolalis2.6であり，コンパイラにはgccを使用した．コンパイルの際のオプションにはMediaBench提供のMakefileにプロファイルの為の-pgオプションを加えたものを用い，実行の際にはMediaBenchに用意されたシェルスクリプト，入力データを用いた．
実行後プロファイラによって収集されたランタイム情報より実行時間の多くを占める関数を選択，そのマクロフローグラフを作成し，それによって示されるMT間の依存関係を解析することで並列性を分析した．

## 5 MediaBenchの解析

### 5.1 ADPCMの解析

ADPCMにはrawcaudioとrawdauioの2つのプログラムが含まれる．図1にそれぞれのプロファイル結果を示す．

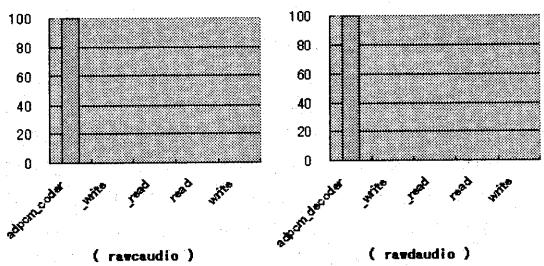


図 1: ADPCM profile

図 1から分かるようにエンコード側ではadpcm_coderが，デコード側ではadpcm_decoderがほぼ100%の実行時間を占める．またこれらの関数はmain内のループから呼び出される．よって，並列化

にあたってadpcm_coder内部での並列性と，呼び出し側であるmainのループのイタレーションレベルでの並列性を検討した．rawcaudio，rawdaudioともにほとんど同じような特徴のMFGであったため，ここではrawcaudioについてのみ述べる．

adpcm_coderの実行時間は一回の呼び出しで0.2ms，全呼び出し回数は148回なので呼出し側のループをイタレーション間で並列化出来ればその効果は大きいと思われるが，このループ内にはファイルI/Oがあること，さらに各イタレーション間にデータの依存関係が存在するため並列化することは出来ない．

adpcm_coderの内部処理を見ると，多くのifブロックで構成され，かつ全てのタスクが1，2ステートメントからなるBPAであり，複雑なデータ依存関係を持っている．これら一つ一つのBPAは非常に小さいため，粗粒度並列性は全く期待できない．しかし，これらのBPAはほとんどが単純な演算からなる代入文のため三項演算子を用いて再定義することで若干の近細粒度並列性が見込める．

## 5.2 G.721の解析

G.721にはencode，decodeのプログラムが含まれる．図2にそれぞれのプロファイル結果を示す．

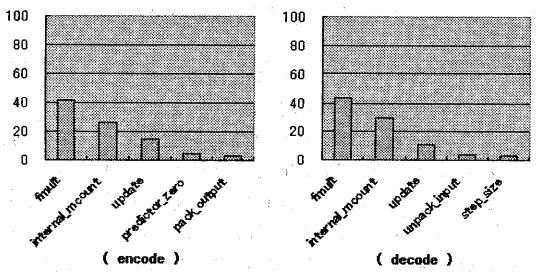


図 2: G721 profile

internal_mcountはプロファイラの評価用の関数であり，アプリケーションの本来のパフォーマンスとは関係がないためここでは評価しない．

プロファイル結果をみるとencode，decodeともfmultが最も多くの実行時間を占めているが，この関数自体は数ステートメントからなる小規模なBPAである．だが，呼出し回数が1180160回と多いため，合計で最も高い割合になっている．そこでfmultの並列性を呼び出し側から検討する．fmultはpredictor_pole，predictor_zeroから呼ばれる．また，この2つの関数はg721_encoderから呼ばれている．まず，トップモジュールのg721_encoder(図3)からみていく．

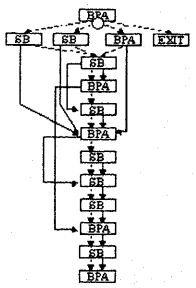


図 3: g721_encoderのMFG

この関数は最も多くの実行時間を占めるfmultを呼び出す関数predictor_zero(呼び出し回数6回)，predictor_pole(2回)を含む．まず，この二つの関数を並列化することを検討する．この二つの関数を含むマクロタスクにはデータの依存関係があり，そのままでは並列化することが出来ない．これに対しては単純なテンポラリ変数を用いることで並列化が可能である．g721_encoderを上記の方法で変換したものが図4のMTGである．

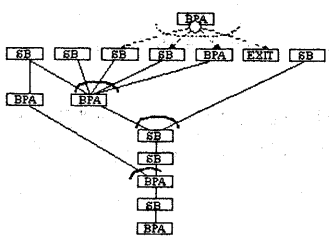


図 4: g721_encoderのMTG

次にpredictor_zero(図5)をみる．この関数ではループ前のSB，ループ内のSBは共にfmultを呼び出し，戻り値の加算を行っている．そのため，g721_encoderの時と同様にテンポラリ変数を用いることにより並列化できる．それにより並列化したものが図6のMTGである．

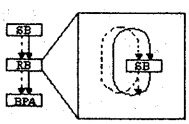


図 5: predictor_zeroのMFG

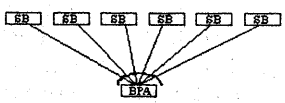


図 6: predictor_zeroのMTG

predictor_pole(図7)ではfmultを呼び出すSBが2箇所あるが，これらのSB間にデータ依存関係が無いため容易に並列化が可能である．

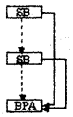


図 7: predictor_poleの MFG

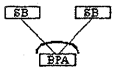


図 8: predictor_poleの MTG

## 5.3 EPICの解析

EPICにはepic, unepicの2つのプログラムが含まれる．図9にそれぞれのプロファイル結果を示す．

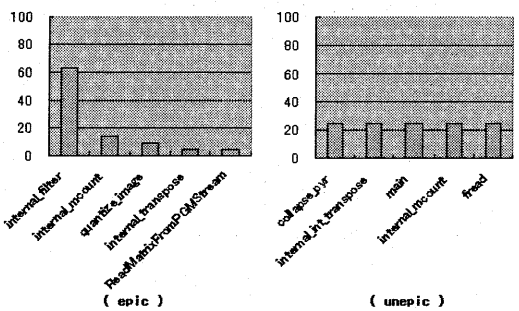


図 9: EPIC profile

epicではinternal_filterが処理時間の大半を占めている．図10にinternal_filterのMFGを示す．

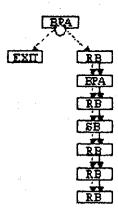


図 10: internal_filterのMFG

そのままではほとんど並列性はみえないが，この関数には次のような特徴がある．各RBは非常に似た構造をしていて，各RBで次のイタレーションの初期値を決定している．そして次のイタレーションはfor文の条件と単純な加算のみで計算される．これらの特徴を考慮に入れプログラムを変換すると図11の様なMTGを作成することができる．ただ，これはあくまでプログラムに手作業で変更を加えた場合であり，コンパイラによる自動並列化と手動並列化で大きく差が出ると考えられる．

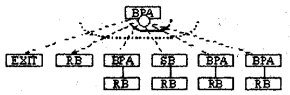


図 11: internal_filterのMTG

unepicで上位を占めているcollapse_pyrはinternal_filterの逆にあたるプロセスでありデータの復号を行う．collapse_pyrのマクロフローグラフはinternal_filterのものにかなり似た傾向があり，同様な並列化が可能である．

## 5.4 PEGWITの解析

暗号化，復号化の2つの処理のプロファイルを行った．図12にそれぞれのプロファイル結果を示す．

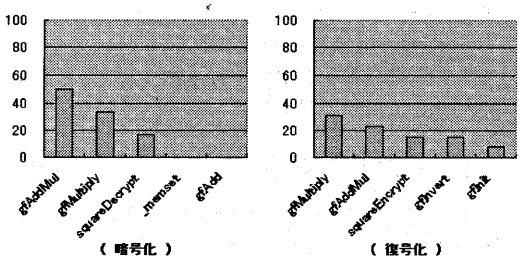


図 12: PEGWIT profile

どちらの処理でもgfAddMul，gfMultiplyが共通して実行時間の多くを占める．gfAddMul自体はほとんど時間はかかっていないが全体で41210回呼ばれるため，その呼び出しを並列化できると効果は大きいと思われる．gfAddMulは関数gfInvert(図 13)内のループで2回続けて呼ばれており，かつデータ依存が無いため2並列動作が可能である．ただしそのループ自体はgoto文を含むためイタレーション間での並列化は出来ない．

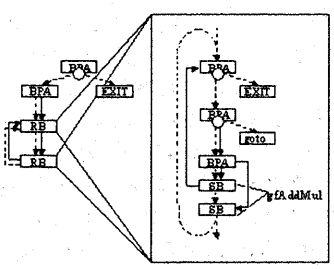


図 13: gfInvertのMFG

gfMultiply(図 14)内のRBはイタレーションレベルで並列化可能なので，中粒度並列性を引き出せる．

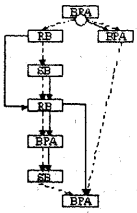

図 14: gfMultiply の MFG

## 5.5 MPEG2 の解析

MPEG2 には mpeg2encode, mpeg2decode の 2 つのプログラムが含まれる. 図 15にそれぞれのプロファイル結果を示す.

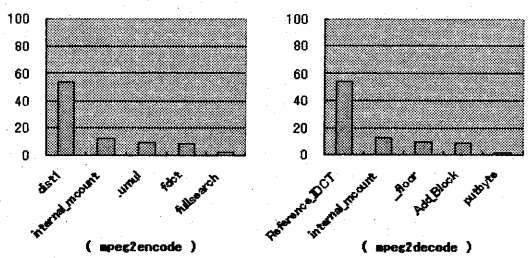


図 15: MPEG2 profile

mpeg2encoder では dist1, mpeg2decoder では Reference_IDCT がほとんどの実行時間を占める. dist1 のループはリダクションループであり容易に並列化が可能である. Reference_IDCT の二つのループはそれぞれリダクションループを含む三重ループである. これらのループの各イタレーション回数は 8 回, 最内側のループ内で 64 要素の配列計算を行う 512 回転のループである. これらに対してはループ崩壊などの技法が容易に適用できる.

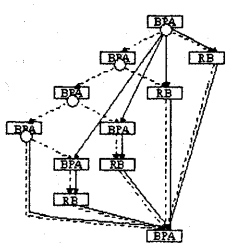

図 16: dist1

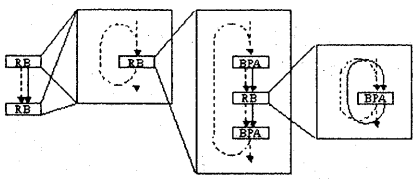

図 17: Reference_IDCT

## 5.6 GSM の解析

GSM には toast, untoast の 2 つのプログラムが含まれる. 図 18にそれぞれのプロファイル結果を示す.

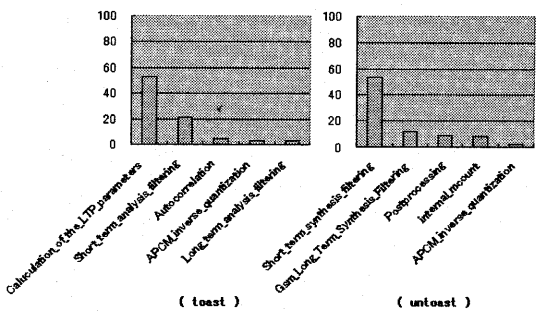


図 18: GSM profile

Calcuration_of_the_LTP_parameters のタスクグラフは図 19のようになる. この関数は基本的にシーケンシャル処理であり, 粗粒度並列性は見られないが, いくつかの RB は doall であり中粒度並列性がある. Short_term_analysis_filtering,Short_term_synthesis_filtering は共にシンプルな 2 重ループ構造だがイタレーション間にデータ依存が存在する.

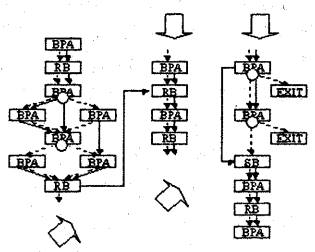

図 19: Calculation_of_the_LTP_parameters の MFG

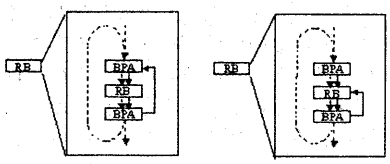

図 20: Short_term_analsis_filtering(左側) の MFG , Short_term_synthesis_filtering(右側) の MFG