動的システム最適化技術SysteMorphの予備性能評価

江島, 和仁
九州大学

吉松, 則文
福岡県産業・科学技術振興財団　福岡知的クラスター研究所

曽我, 武史
福岡県産業・科学技術振興財団　福岡知的クラスター研究所

村上, 和彰
九州大学

出版情報：電子情報通信学会技術研究報告 CPSY2004-12. 104 (476), pp.59-64, 2004-12. 電子情報通信学会CPSY研究会
バージョン：
権利関係：

動的システム最適化技術 SysteMorph の予備性能評価

江島　和仁†　　吉松　則文††　曽我　武史††　村上　和彰†

† 九州大学　〒 816-8580 福岡県春日市春日公園 6–1
†† 福岡県産業・科学技術振興財団　福岡知的クラスター研究所
〒 814–0001 福岡市早良区百道浜 3–8–33
E-mail: †arch-systemorph@c.csce.kyushu-u.ac.jp

あらまし　コンピュータのハードウェアや，ソフトウェアを動的に最適化する「動的システム最適化技術」が注目されている．筆者らは SysteMorph という適応型動的システム最適化技術の概念を用い，その応用システムの研究を行っている．本論文では動的システム最適化技術によって到達可能な性能向上を評価実験によって示す．動的システム最適化技術による性能向上を対象プログラムの最適化前後のクロックサイクル数の比率で定義する．評価実験の結果から VLIW 型の加速実行部を持つシステムで高い性能向上が期待できるアプリケーションを示す．

キーワード　動的システム最適化技術，オンライン・プロファイリング，加速実行

Preliminary Performance Evaluation of Dynamic/Adaptive/System Level Optimization Technology，SysteMorph

Kazuhito ESHIMA†, Norifumi YOSHIMATSU††, Takeshi SOGA††, and Kazuaki MURAKAMI†

† Graduate School of Information Science and Electrical Engineering, Kyushu University　6-1 Kasuga-koen, Kasuga，Fukuoka, 816-8580 Japan
†† FLEETS，Fukuoka Laboratry for Emerging & Enabling Technology of SoC　3-8-33, Momochihama, Sawara-ku, Fukuoka 814-0001, Japan
E-mail: †arch-systemorph@c.csce.kyushu-u.ac.jp

Abstract　Dynamic optimization technology for a system which dynamically optimizes both software and hardware is gaining attention of computer system researchers. The authors are researching SysteMorph, a feedback directed dynamic and adaptive hardware/instruction set architecture(ISA)/software co-optimization technology. The technology enables to optimize performance, power, and consumption energy for a system dynamically. In this paper the authors describe attainable performance improvement with the SysteMorph technology. Clock cycle counts of benchmark programs by applying the SysteMorph are evaluated using a simulation method. The performance improvement is shown as a rate of the clock cycle count comparing with unoptimized state. As a conclusion the authors show an application group which can achieve large performance improvement in a system using VLIW acceleration unit.

Key words　Dynamic optimization，Online profiling，　accelerate execution

1. はじめに

半導体技術の進歩による再構成可能ロジックの性能向上を背景に，コンピュータシステムを構成するハードウェアやソフトウェアを動的に最適化する技術が注目されている．動的に最適化することにより，システム運用後での規格の変更，それによるシステム特性の最適化が可能になる．また，システム運用後に最適化を行うことで，システム開発の期間やコストを抑えることが可能である．

しかし，動的最適化には未解決の課題も少なくない．システム運用時に最適化を行うには最適化処理を行うことによるシステムへの影響を抑えなくてはならない．また，システムの動作中に最適化可能な箇所を特定し，最適化方法を特定しなければならない．当然ながら，最適化を行う機構も組み込ま

れている必要がある．上記の未解決の課題を解決すべく，我々は SysteMorph [1] [2] [3] [4] という動的システム最適化技術の研究，開発を行っている．SysteMorph とはアプリケーションプログラムの実行中に得られるプロファイル情報に基づいて，ソフトウェア，ハードウェア，命令セットアーキテクチャを変更して，その性能，消費電力等を最適化する技術である．

現在，動的システム最適化技術は静的には不可能な最適化が可能で，高い性能向上が得られるとされている．その前提の下で研究，開発が進められているが，システムを動的に最適化することでどれほどの性能向上が期待できるかの評価は成されていない．筆者は動的システム最適化技術によって得られる，性能向上とアプリケーション特性，およびシステムの動作原理との因果関係について研究を行っている．動的システム最適化のモデルは複数考えられる．それぞれのモデルにおいて期待できる性能向上を評価し，性能向上が現れたアプリケーションの特性を明らかにすることで性能向上とアプリケーション特性，およびシステムの動作原理との因果関係を明らかにする．本論文では，そのための予備評価として複数のベンチマークに対して，動的システム最適化技術によって到達可能な性能向上を示す．

2 章で我々が提案する動的システム最適化技術・SysteMorph の概要を述べ，3 章で本論文で行った，動的システム最適化技術で到達可能な性能向上の評価方法を述べ，4 章で評価結果を示し，5 章で考察する．6 章でまとめと今後の課題を述べる．

2. 動的システム最適化技術 SysteMorph

2.1 SysteMorph の概要

SysteMorph とは, 主にプロセッサ・ベース・システムにおいて, 対象とするアプリケーション・プログラムのプロファイル情報に基づいて, 当該アプリケーションを実行中に以下の要素を変更する．

- ソフトウェア（当該アプリケーション・プログラムのオブジェクト・コード）
- 命令セット・アーキテクチャ（当該アプリケーション・プログラムを実行しているプロセッサの命令セット・アーキテクチャ）
- ハードウェア（当該アプリケーション・プログラムを実行しているプロセッサおよび周辺ハードウェア）のモードならびに構成

これにより消費電力, 消費エネルギーを最適化する, すなわち Feedback-Directed Dynamic Software / Instruction Set Architecture / Hardware Co-optimization 技術である（図 1）．

本技術により，システム LSI の設計期間の短縮化，システム LSI の寿命の長期化，ならびに，システム LSI 運用時の性能，消費電力，消費エネルギーの最適化を狙うことが可能になる．SysteMorph はリコンフィギュラブル・コンピューティングとは異なり, ランタイムにハードウェアおよびソフトウェアを動的に最適化する．

携帯電話を例に SysteMorph の応用を考えると，次のようなシステム構成が考えられる．システムは，機器側（この例では携帯電話端末）と，外部システム（この例では最適化を行うセンター）から構成される．機器側はまず動作中のアプリケーションのプロファイルを収集する．電力を十分に使える充電時に，外部システムに収集したプロファイルを送信する．外部システムは得られたプロファイルから最適化を行う．機器側は，次に外部システムと通信する際に，自身の再構成に関する情報があるかどうかも問い合わせ，それが存在すれば，その情報を基に自身を再構成する．

2.2 オンライン・プロファイラによるホットパス検出

2.2.1 プロファイラの概要

オンライン・プロファイリングは，アプリケーションの振舞いに関する情報をその実行中に収集する技術である．オンライン・プロファイリングはプログラムの実行途中のプロファイル結果に基づき，当該プログラムの残りの実行に対する「最適化のヒント」を導き出す必要がある．高頻度で実行されるパスをホットパスと定義し，ホットパスを検出する．

2.2.2 BH 法によるオンライン・プロファイリング

オンライン・プロファイリングに関して，近年さまざまな研究がされている．Evelyn らは，実行中のアプリケーション中のパス，すなわち連続して実行される基本ブロックの集合について，頻繁に実行されるパスをアプリケーションの実行時に推定する手法として NET(Next Execution Tail) 法を提案している [6]．我々は BH 法というオンライン・プロファイリング手法を提案した [5]．BH 法は NET 法に対して, ホットパス推定手法が異なる．図 2 のような基本命令ブロックおよび基本命令ブロック最終の分岐命令のパス構造であるとき，高い頻度で実行されるループを構成するパスをホットパスとして検出する．プロセッサコアで実行される分岐命令に着目し，分岐アドレスと分岐先アドレスの組の実行回数（図 2 のある基本ブロックから他の基本ブロックへの矢印の通過回数）の履歴を収集する．この実行回数が，ある閾値を超えたパスをホットパスとする．

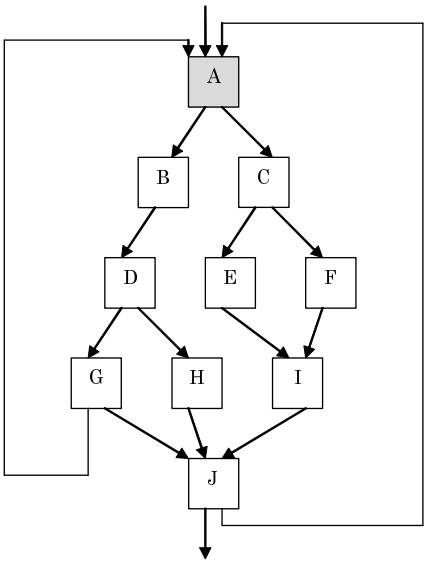

図 2　対象プロファイリングパス構造

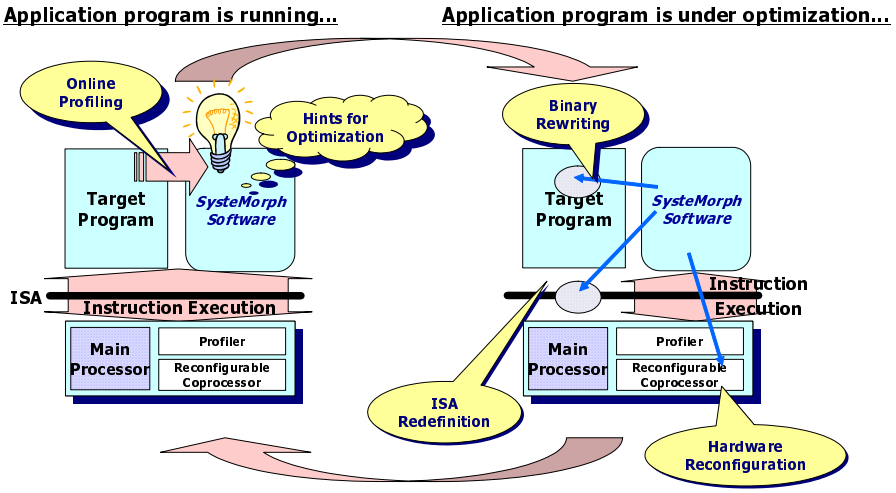

図 1　SysteMorph 概念図

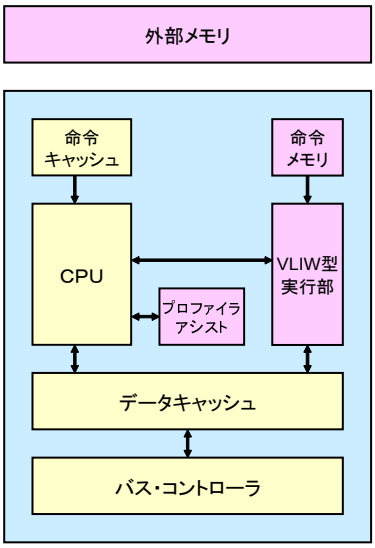

図 3　評価対象とするシステムのモデル

2.3　評価対象とするシステムのモデル

本論文ではホットパスに被覆される命令列を VLIW 型コプロセッサにオフロードする手法を想定する．動的システム最適化技術により到達可能な性能向上を評価するモデルは図 3 のようになっている．メイン・プロッセッサとコプロセッサがあり，基本的にはメインプロセッサで命令を実行する．コプロセッサは VLIW 型の加速実行部を仮定する．オンライン・プロファイラはハードウェアで実現されている．メイン・プロセッサでプログラムを実行し，実行中にプロファイラが実行のトレースからホットパスを検出する．ホットパスのプロファイル情報はプロファイラに蓄積される．蓄積されたプロファイル情報を基に，メイン・プロセッサで実行中にホットパスに該当する命令列を実行しようとすると，コプロセッサを起動し，実行させる特殊な命令を発行する．コプロセッサ起動の命令が発行されると，ホットパスに被覆される命令列をコプロセッサに転送する．ホットパスに被覆される命令列はコプロセッサで並列実行するために再スケジューリングを施される．再スケジューリングされたホットパスに被覆される命令列はコプロセッサで加速実行される．実行が終了すると，メイン・プロセッサと共通のデータキャッシュに転送される．コプロセッサ起動命令の発行，メイン・プロセッサとコプロセッサのデータの転送，コプロセッサ実行のための再スケジューリング，の機能はソフトウェアで実現され，メイン・プロセッサ上で実行される．

本論文では VLIW 型コプロセッサに再スケジューリングを行い，加速実行するモデルを仮定しているが，再スケジューリングの際に，動的ソフトウェアパイプライニング技術を適用させ，性能向上を図るモデルに関しても，研究，開発が進められている [7] [8].

3.　性能評価方法

ここでは 2.3 節で仮定したシステムにより到達可能な性能向上の評価実験について述べる．

3.1　仮　　定

評価にあたり，以下を仮定する．

- 全ての命令は 1 クロックサイクルで実行が完了する
- VLIW 型コプロセッサは 4way である
- 動的最適化によるオーバーヘッドは考慮しない

動的最適化によるオーバーヘッドは以下のものが考えられる．コプロセッサ起動命令の発行，メイン・プロセッサとコプロセッサのデータの転送，コプロセッサ実行のための再スケジューリング，に必要なクロックサイクルである．

3.2　評 価 指 標

本節では性能向上を定義する．対象プログラムに動的システム最適化技術を適用し，ベンチマーク・プログラムを実行した場合の所要クロックサイクル数を C_{opt} とする．動的システム最適化技術を適用せずに，単一のプロセッサ・システムで同一のベンチマーク・プログラムを実行した場合の所要クロックサイクル数を C_{non_opt} とする．ベンチマーク・プログラムの全命令数を N_{ins_all} とする．C_{opt} と C_{non_opt} の比率をもって，動的システム最適化技術の性能向上とする．動的システム最適化技術による性能向上 *speedup* は以下の式 (1) になる．

$$speedup = \frac{\frac{N_{ins\_all}}{C_{opt}}}{\frac{N_{ins\_all}}{C_{non\_opt}}} = \frac{C_{non\_opt}}{C_{opt}} \quad (1)$$

3.3 評価環境

2.3 節で仮定したモデルでの到達可能な性能向上を評価をするために以下の実験を行った．本論文では実行型プロセッサシミュレータとして広く利用されている SimpleScalar [9] を利用した．環境は以下である．

- 命令セットシミュレータ: SimpleScalar version 3.0
- モデル: sim-safe
- 命令セット: PISA
- プロファイラ: 2.2.1 節の BH 法によるプロファイラ
- 実行マシン: Sun Solaris 5.8 (gcc2.95.3)
- ベンチマーク: MiBench [10]，h264 [11]
- プロファイラの設定閾値: 1000

プログラム開始から終了までの全命令をシミュレータで実行した．

3.4 動的システム最適化技術により到達可能な性能向上に関する評価

評価の第一段階として，動的システム最適化技術により到達可能な性能向上を求める．どのような最適化を施しても動的最適化技術ではこれ以上，性能向上が望めないという値である．本節では以下の方法で 3.2 節で定義したクロックサイクル数 C_{opt} を求める．動的システム最適化技術を適用せずに，単一のプロセッサ・システムで同一のベンチマークを実行した場合の所要クロックサイクル数を C_{non_opt} と定義した．比較対照とする単一のプロセッサ・システムは SimpleScalar と同一のシステムを仮定するため，SimpleScalar での実行に所要したクロックサイクルが C_{non_opt} に一致する．

理想的には VLIW 型コプロセッサで実行する際に，ホットパスに被覆される命令列の実行に必要なサイクル数は $\frac{1}{MP}$ 倍になる．MP は VLIW の way 数である．本論文では 4way とした．N_{ins_hot} はホットパスに被覆される命令の総数，N_{ins_all} はシミュレータで実行された全命令の総数とする．全ての命令は 1 クロックサイクルで実行が完了すると仮定し，かつ，動的システム最適化技術を適用しない単一のプロセッサ・システムは SimpleScalar と同一のシステムを仮定するため，SimpleScalar での実行に所要したクロックサイクルが C_{non_opt} に一致し，$N_{ins\_all} = C_{non\_opt}$ となる．

本節の評価で C_{opt} は式 (2) になる．

$$C_{opt} = (N_{ins\_all} - N_{ins\_hot}) + \frac{N_{ins\_hot}}{MP} \quad (2)$$

よって性能向上 *speedup* は式 (3) になる

$$speedup = \frac{C_{non\_opt}}{C_{opt}} = \frac{N_{ins\_all}}{(N_{ins\_all} - N_{ins\_hot}) + \frac{N_{ins\_hot}}{MP}} \quad (3)$$

3.5 VLIW コードをリストスケジューリングで生成した場合の性能向上に関する評価

前節での算出方法はコプロセッサの並列度を完全に使い切ることが可能と仮定している．しかし，実際にはプログラムの破綻が起きないようにデータの依存関係を考慮する必要がある．本節ではデータの依存関係を考慮したスケジューリングを行うことにより，クロックサイクル数 C_{opt} の評価精度を上げる．ホットパスに被覆される命令列に対し，使用するレジスタの依存関係を考慮した再スケジューリングを行う．スケジューリング方法は以下の制約条件を与えたリストスケジューリングである [12]．

- コプロセッサの演算器は 4 個
- 最大で 4 並列の実行が可能
- 演算系の命令については 4 個全てが対応可能
- Load/Store 命令に対応可能な演算器は 4 個のうち 1 個のみ

本節では以下の方法で 3.2 節で定義したクロックサイクル数 C_{opt} を求める．SimpleScalar での実行に所要したクロックサイクルが C_{non_opt} に一致し，$N_{ins\_all} = C_{non\_opt}$ となる．ホットパスは複数出現するので ID として i を割り当てる．ある i 番目のホットパスである $hotpath(i)$ に対し，コプロセッサ実行のための再スケジューリングを施し，新たに発行した VLIW 命令の総数を $N_{vliw(i)}$ とする．$hotpath(i)$ が出現した回数を $N_{hotpath(i)}$ とする．N_{ins_hot} は全てのホットパスに被覆される命令の総数，N_{ins_all} はシミュレータで実行された全命令の総数とする．全ての命令は 1 クロックサイクルで実行が完了すると仮定した．これを全てのホットパスについて和をとるので，C_{opt} は式 (4) になる．

$$C_{opt} = (N_{ins\_all} - N_{ins\_hot}) + \sum_i (N_{vliw(i)} * N_{hotpath(i)}) \quad (4)$$

本節での評価方法で動的最適化を行った場合の性能向上 *speedup* を算出する．*speedup* は式 (5) になる

$$speedup = \frac{C_{non\_opt}}{C_{opt}} = \frac{N_{ins\_all}}{(N_{ins\_all} - N_{ins\_hot}) + \sum_i (N_{vliw(i)} * N_{hotpath(i)})} \quad (5)$$

4. 評価結果

各ベンチマークにおける全実行命令数 (N_ins_all) と，ホットパスに被覆される命令数 (N_ins_hot) を表 1 に示す．表 1 の 4 列目の項目は全実行命令数に占める，ホットパスに被覆される命令数の比率，つまり，$(N\_ins\_hot)/(N\_ins\_all)$ である．

susan_smoothing，tiff2bw，tiffmedian，sha，crc32 ではホットパスに被覆される命令が多く現れた．basicmath，qsort，dijkstra，blowfish，rijndael，adpcm，fft 等のベンチマークではホットパスに被覆される命令列の割合が少なかった．

表 1　全実行命令数とホットパスに被覆される命令数

ベンチマーク	N_ins_all	N_ins_hot	比率
basicmath	31957684	3728335	0.11
bitcount	43558932	28290851	0.64
qsort	38003332	2614593	0.07
susan.corners	975796	509480	0.52
susan.edge	2089854	1408545	0.67
susan.smoothing	25590758	24367500	0.95
jpeg.decode	7334531	5945405	0.81
jpeg.encode	27256338	19703633	0.72
lame3.96	268932495	198950547	0.74
tiff2bw	52098261	51157303	0.98
tiff2rgba	45713854	44307073	0.97
tiffdither	295292634	106559054	0.36
tiffmedian	177831892	164310237	0.92
h264.decode.fb001	179107872	109597076	0.62
dijkstra	54879892	5857198	0.11
patricia	150932582	15824137	0.10
stringsearch	7885456	21920	0.003
blowfish.decode	37054430	4459100	0.12
blowfish.encode	37373983	4731930	0.12
rijndael.decode	37665918	7795535	0.21
rijndael.encode	37744107	7795567	0.21
sha	13200948	12883368	0.97
adpcm.decode	25245400	1592580	0.06
adpcm.encode	30192460	2049760	0.07
crc32	532869380	532159020	0.99
fft	94847975	15094375	0.16

次に表 1 の結果から，3.4 節と 3.5 節で定義した *speedup* を図 4 に示す．susan_smoothing，tiff2bw，tiff2rgba，tiffmedian，sha，crc32 等のベンチマークでは到達可能な性能向上は 3 倍から 4 倍になった．リストスケジューリングで生成した場合の性能向上は上記のベンチマークでは 1.3 倍から 1.7 倍程度になった．逆に basicmath，qsort，dijkstra，blowfish，rijndael，adpcm，fft 等のベンチマークでは到達可能な性能向上ですら 1.1 倍程度で，リストスケジューリングで生成した場合の性能向上はほとんど得られなかった．

5. 考　　察

4. 節での結果から，susan，jpeg，tiff2bw，tiff2rgba，tiffmedian，sha，crc32 等のベンチマークでは特定の箇所が高頻度で実行された．故に，2. 3 で仮定したモデルのように高頻度で実行される箇所をオフロードする評価モデルでは高い性能向上が期待できる．

逆に，basicmath，qsort，dijkstra，blowfish，rijndael，adpcm，fft 等のベンチマークではホットパスに被覆される命令列の割合が少なく，本論文でのプロファイルにより検出された特定の箇所が，高頻度で実行されることはない．2. 3 で仮定したモデルでは高い性能向上は得られにくい．上記のベンチマークで高い性能向上を得るには 2. 3 節とは異なるシステム構成が必要になる．

4. 節での結果から，3. 5 節のスケジューリング手法による性能向上は，全てのベンチマークにおいて到達可能な上限値より，大きく下回る結果となった．単純な並列化による性能向上が少なく，他の最適化を検討するなど，改善の余地があると言える．

また，4. 節の結果から今回対象としたベンチマークを以下の３項目に分類できる．

（1） 到達可能な性能向上，VLIW リストスケジューリングの両方で高い性能向上が現れた

（2） 到達可能な性能向上では高い値が現れたが，VLIW リストスケジューリングでは高い性能向上が現れなかった

（3） 到達可能な性能向上において高い性能向上が現れなかった

(1) に分類されるのは sha，susan_smoothing 等である．ホットパスに被覆される命令列の割合が高く，性能向上の上限値が高いことから，仮定したモデルのように，高頻度で実行される箇所を最適化するモデルに対して，高い性能向上が得られると言える．

(2) に分類されるのは crc32 である．ホットパスに被覆される命令列の割合が高く，性能向上の上限値が高い．しかし，ホットパスに被覆される命令列を VLIW にリストスケジューリングを行うと，高い性能向上が現れない．ホットパス部分の命令レベルの並列性が低いと言えるため，高頻度で実行される箇所はあるが，並列実行で性能向上を図るのは難しいと言える．

(3) に分類されるのは basicmath，qsort，dijkstra，blowfish，rijndael，adpcm，fft 等である．ホットパスに被覆される命令の割合が少なく，ホットパスをコプロセッサにオフロードするモデルでは性能向上を図るのは難しいと言える．

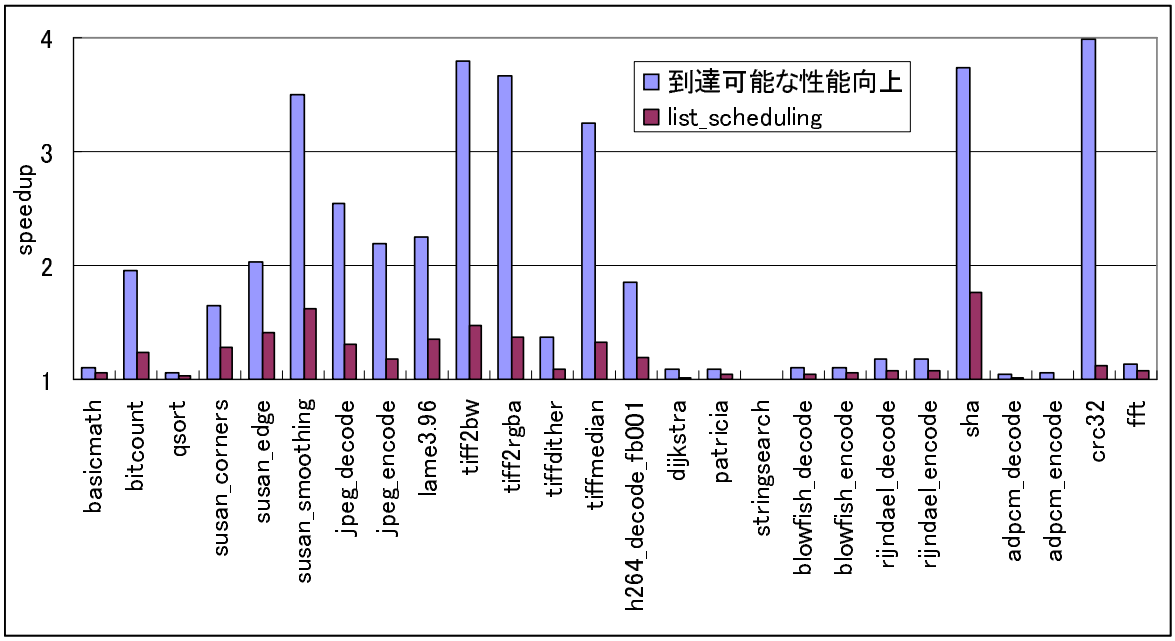

図 4　性能向上の上限値と VLIW コードをリストスケジューリングで生成した場合の性能向上

今回の評価モデルではコプロセッサを VLIW 型としたが，リコンフィギュラブル型など，他のモデルでの性能評価も行う必要がある．

6. おわりに

本論文では，動的システム最適化技術によって到達可能な性能向上を示した．様々な動的システム最適化モデルにおいて，性能向上とアプリケーション特性，およびシステムの動作原理との因果関係を明らかにするための予備評価として行った．評価実験の結果から並列実行型の加速実行部を持つシステムで高い性能向上が期待できるアプリケーションを示した．

今後はアプリケーションの実行アルゴリズムを明らかにし，性能向上とシステムの動作原理に関する調査，考察を行う．また，本論文で想定しなかった他のモデルにおいて期待できる性能向上を評価する．得られた結果から性能向上とアプリケーション特性，およびシステムの動作原理との因果関係を明らかにする予定である．

謝辞　本研究の一部は，平成 13〜15 年度日本学術振興会科学研究費補助金基盤研究 (A)(2)（一般研究）「ハードウェア構成を動的に最適化する「ハードウェア・モーフィング」技術の開発」，平成 14〜18 年度文部科学省知的クラスター創成事業「次世代システム LSI アーキテクチャの開発」に拠る．

文　献

[1] 林田　隆則, 数　勇介, 村上　和彰, “SysteMorph：Functionality Morphing, ” The 4th Summer Workshop on Embedded System Technologies (SWEST4), 2002.

[2] Norifumi Yoshimatsu, Makoto Yoshida, Takeshi Soga, Makoto Shuto, Yasuyuki Tanoue, Yosuke Fujii, Kazuhito Eshima, Takanori Hayashida, and Kazuaki Murakami, “SysteMorph: Dynamic/Online/Adaptive System-Level Optimization for SoC, ” The 7th International Conference on High Performance Computing and Grid in Asia Pacific Region, pp.442-447, Jul. 2004.

[3] Takanori Hayashida, Kazuaki Murakami, “Evaluating Online Hot Instruction Sequence Profilers for Dynamically Reconfigurable Functional Units, ” pp.901-908, IEICE TRANS. INF. & SYST., VOL. E86-D, NO.5, May. 2003.

[4] 曽我　武史, 吉田　真, 吉松　則文, 村上　和彰, “DAP/DNA を用いた SysteMorph プロトタイピング－オンライン・ハードウェア合成手法－, ” 電子情報通信学会技術研究報告, 第 2 回リコンフィギュラブルシステム研究会論文集, pp.22-27, Nov. 2003.

[5] 吉田　真, 曽我　武史, 村上　和彰, 林田　隆則, “DAP/DNA を用いた SysteMorph プロトタイピング, ” 電子情報通信学会技術研究報告, 第 1 回リコンフィギュラブルシステム研究会論文集, pp.221-226, Sep. 2003.

[6] E.Duestenwald and V.Bala, “Software Profiling for Hot Path Prediction: Less is More, ” Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), pp.202-211, Nov. 2000.

[7] 田上　裕之, 村上　和彰, “動的ソフトウェアパイプライニング技術の提案と性能評価, ” The 15th Summer United Workshop on Parallel, Distributed, and Cooperative Processing (SWoPP), 2002.

[8] 藤井　洋輔, 吉田　真, 村上　和彰, “動的システム最適化技術 SysteMorph の一実装例　〜 動的トレースベース・ソフトウェアパイプライニングの実装 〜, ” 電子情報通信学会技術研究報告, デザインガイア リコンフィギャラブルシステム研究会 Dec. 2004.

[9] SimpleScalar LLC, http://www.simplescalar.com/

[10] MiBench, http://www.eecs.umich.edu/MiBench/

[11] h264, ftp://ftp.imtc-files.org/jvt-experts

[12] 中田育夫, コンパイラの構成と最適化, 朝倉書店, 1999 年