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

2007－ARC－172 (13)
2007－HPC－109 (13)
2007／3／1

マルチコア・SMT プロセッサ上における シェルスクリプト高速化手法

杉田秀[†] 深山辰徳[†] 蛭田智則[†] 當仲寛哲[††] 山名 早人[‡*]

[†]早稲田大学大学院理工学研究科 [‡]早稲田大学理工学術院

[††](有)ユニバーサルシェルプログラミング研究所 [*]国立情報学研究所

概要 本研究では，マルチコア・SMT(Simultaneous Multi-Threading) プロセッサ上でのシェルスクリプト実行の有効性を示すことを目的とする．近年マルチコアプロセッサおよび SMT の技術が注目されている．しかし，並列性を考慮していないプログラムを通常のコンパイラでコンパイルしても，マルチコアプロセッサやマルチスレッドを有効活用することはできない．通常，これらの技術の恩恵を受けるためには，並列化プログラミングが必要であり，自動並列化の技術も数多く研究されている．本稿では，シェルスクリプト自身が持つ並列性に着目し，マルチコアプロセッサ・SMT 環境において，シェルスクリプトの高速化を実現する手法，シェルスクリプトの自動並列化プログラムを提案する．本提案手法を用いて，マルチコアプロセッサ・SMT マシン上でシェルスクリプトの実行を行った結果，手法適用前に比べて 1.4～1.8 倍の速度向上を得ることができた．

A Speed-Up Method for Shell Scripts on Multi-Core and SMT Processors

SHU SUGITA[†], TATSUNORI FUKAYAMA[†], TOMONORI HIRUTA[†], NOBUAKI TOUNAKA[††] and HAYATO YAMANA[‡*]

[†]Graduate School of Science and Engineering, Waseda University

[‡]Faculty of Science and Engineering, Waseda University

[††]Universal Shell Programming Laboratory [*]National Institute of Informatics

Abstract: The purpose of this study is to show the effectiveness of shell script execution on multi-core and/or SMT (Simultaneous Multi-Threading) processors. Recently, multi-core processor and SMT technique have become popular even at home and in business. However, using programs or compilers without consideration of parallelism does not give us the benefits of multi-core and multi-thread. Programmers have to do parallel programming to receive the benefits. Therefore, automatic parallelizing technique has been studied actively. This paper proposes automatic parallelizing scheme for shell script programs on multi-core and/or SMT processors. As a result of the experiment, we have confirmed that the speed-up of automatic parallelized shell script program is 1.4 to 1.8 times in comparison with the original shell script program.

1 はじめに

プロセッサの誕生以来，プロセッサ性能の向上は，主にクロック周波数の向上やパイプラインの細段化によってなされてきた．つまり，単位時間当たりの処理量，IPC (Instructions per clock)の向上を目指して，プロセッサ性能を向上させてきた．しかし IPC の向上のために，製造プロセスの微細化によるリーク電流の増加やトランジスタ数の増加により，消費電力や対コスト比の問題が無視できないものとなってきた．単純にプロセッサ性能を上げることができなくなってきたのである．

そのため，近年複数のプロセッサコアを１つのチップ上に集積するマルチコアプロセッサや，複数のスレッドを同時に処理するマルチスレッドの技術が注目されている．1999 年，IBM 社がデュアルコアプロセッサを搭載した IBM の Power4 が発表されたことをきっかけに，その後様々なマルチコア対応のプロセッサおよび SMT (Simultaneous Multi-Threading) 対応のプロセッサが発表された．Sun Microsystems 社からは，UltraSPARC IV，UltraSPARC T1，Intel 社からは，

Pentium D，Core などが発表された．

しかし，並列性を考慮していないプログラムや通常のコンパイラを用いても，マルチコアプロセッサやマルチスレッドを有効活用することはできない．通常，マルチコアプロセッサやマルチスレッドの恩恵を受けるためには，並列化コンパイラによるプログラムの自動並列化[1,2,3,4,5,6]，あるいは OpenMP や MPI などを使用した並列化プログラミングが必要である．そのため，プログラムの並列化を実現するための研究が多数なされている．

一方，コマンドやシェルの組み込みコマンドなどを組み合わせてプログラミングを行うシェルスクリプトは，業務用バッチ処理などで多用されており，その高速化が望まれている．シェルスクリプト内の各コマンドはプロセスとして動作するため，パイプライン処理[7]を利用することで，並列化が容易に行える特性がある．すなわちシェルスクリプト実行時には，各コマンドがコアやスレッドへ適切に割り当てられることで，容易に処理性能の向上が図れる可能性がある．

そこで，本研究ではシェルスクリプト自体が持つ並列性に着目し，マルチコアプロセッサ・SMT (Simultaneous Multi-threading)環境における，シェルスクリプトの高速化を実現する手法を提案する．提案手法では，シェルスクリプト全体の実行効率を高めるために，スクリプトの細分化を行い，その細分化されたファイルをマルチコアプロセッサ上で並列実行させることで，シェルスクリプトの高速化を実現する．本論文では，提案手法を適用したシェルスクリプトを，様々なマルチコアプロセッサ・SMT 環境のマシン上で実行し，その処理効率を測定することで，マルチコアプロセッサ上でのシェルスクリプト実行の有効性を示す．

以下，第 2 節ではマルチコアプロセッサの概略を説明する．つづいて第 3 節ではシェルスクリプト概要とその並列可能性についての説明をし，第 4 節で提案手法「シェルスクリプトの自動並列化」について解説する．第 5 節では，マルチコア・SMT プロセッサ上でシェルスクリプトの実行を行い，その評価をする．第 6 節で関連研究について述べ，最後に第 7 節で本稿のまとめを行う．

2 マルチコアプロセッサと SMT

本節では，近年注目されているマルチコアプロセッサと SMT 技術について触れる。

2.1 マルチコアプロセッサ

マルチコアプロセッサ（チップマルチプロセッサとも呼ぶ）とは，1 チップ上に複数のプロセッサコアを搭載した CPU のことである．原理は従来のマルチプロセッサと同様，複数のコアで処理を分担する仕組みである．マルチコアプロセッサでは，内部のコア同士がキャッシュを共有しているものと共有していないものがあり，用途によっては性能を大きく左右することがある．例えば，共有キャッシュの場合，コア同士でまったく違うデータを参照していると，互いにキャッシュのデータを追い出してしまうことになり，性能が大きく低下する．しかし，互いに同じデータを使用すれば，メモリではなくキャッシュからデータを取り出すことができ，性能は大きく向上する．非共有キャッシュの場合は，逆の動作をすることになる．したがって，アプリケーションの使い方を考えて，キャッシュ共有／非共有のプロセッサを選ぶ必要がある．

2.2 SMT

SMT (Simultaneous Multi-Threading)とは、単一のプロセッサを複数のプロセッサに見せることで、マルチスレッドに対応した OS において、複数のスレッドを同時に扱うことができるようにする技術である。SMT に対応しているプロセッサを使用すると，OS からそのプロセッサは複数のプロセッサとして認識される．

3 シェルスクリプトと並列化

本節では，本研究の対象プログラムとなるシェルスクリプトについて説明する．また，シェルスクリプト自身が持つ並列性に着目し，シェルスクリプトの並列化についても言及する．

3.1 シェルスクリプト

シェルスクリプトとは，UNIX 等のコマンドやシェルの組み込みコマンドなど（以下「コマンド」と表記）を組み合わせてプログラミングを行い，実行できるようにしたものである．

シェルでは，プログラムの制御やファイルの操作などを簡単に行うことができ，その組み合わせによっては様々な動作を行うプログラムを書くことができる．用途としては，定期的にまとめて一括の処理を行う“バッチ処理”のプログラムを書く際に使用されることが多い．例えば，コンピュータ起動時の環境設定，企業における売上データや受注データの集計処理などに使われる．

3.2 パイプライン処理

シェルにおけるパイプライン処理とは，あるコマンドＡの処理結果をコマンドＢの入力として処理することである．シェルの実行時，左から右，あるいは上から下へ処理結果が流れることからパイプラインと呼ばれている．シェルでパイプライン処理を行うには，バ

ーティカルバー（|）を用いる．具体的な書き方は，

COMMAND A | COMMAND B

のように，コマンドとコマンドの間にバーティカルバーを挿入することである．パイプライン処理における，コマンドAからコマンドBへの処理結果の受け渡しの様子を図1に示す．

コマンドAに対応するプロセス＃１とコマンドBに対応するプロセス＃２はほぼ同時に起動する．そして，各プロセスは次に説明する標準入出力命令等で待ちが発生しない限り，同時に実行される．プロセス＃１は標準出力命令を読み込むと，write()命令によりカーネル内のバッファに対して書き込みを行う．また，プロセス＃２は標準入力命令を読み込むと，read()命令によりカーネル内のバッファから読み込みを行う．その際，プロセス＃１による書き込みとプロセス＃２による読み込みは，同期的に行われる．したがって，プロセス＃１の書き込みがバッファ容量（通常数ＫＢ，ただしシステムにより異なる）を超えると，プロセス＃１は中断される．逆に，バッファに残るデータが０になると，プロセス＃２は中断される．

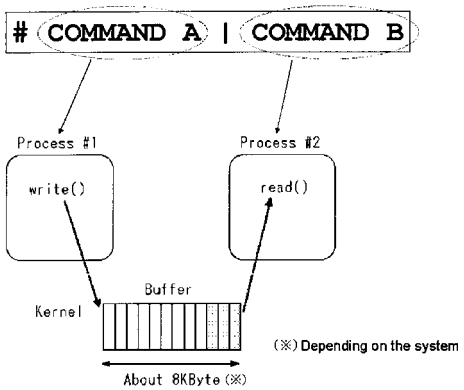

図1 パイプライン処理

マルチコア・SMTプロセッサ上において，プロセス＃１とプロセス＃２は各コアまたはスレッドに割り当てられ，ほぼ同時に実行することができる．そのため，パイプライン処理が張り巡らされたプログラムを，マルチコア・SMTプロセッサ上で動作させることで，コアやスレッドの有効活用をすることができる．

4 提案手法：シェルスクリプトの自動並列化

本節では，シェルスクリプトの実行効率を高めるための手法について説明する．

前節で述べたように，シェルスクリプト中にパイプライン処理が書かれている場合，マルチコア・SMTプロセッサ上においてはその処理は並列実行される．そのため，Ｃ言語やJAVAのプログラムのように，シングルコアからの移植の際にプログラマが並列化プログラミングを意識する必要はない．しかし，並列実行される部分はパイプライン処理，あるいはバックグラウンド処理のある部分のみとなるため，プログラムによっては数%～数十%のスピードアップしか見込めない場合もある．

そこで，本研究ではスクリプトを複数のファイルに分割し，分割ファイルを同時実行することで，全体の処理効率を高めるための手法を提案し，その手法を実現するプログラムを開発した．

4.1 提案手法イメージ

提案手法の全体のイメージを図2に示す．提案手法では，対象となるシェルスクリプトを読み込み，一定のルールに従いスクリプトファイルの分割を行う．ファイル分割後，分割ファイル同士の依存関係を考慮しながら，分割ファイルの実行順序を決定する．その際，相互に依存関係がない分割ファイル同士は，パイプライン処理を利用した同時実行ができるようにする．ユーザは，最終的に出力されるシェルスクリプトを実行するだけで，元のスクリプトを実行するよりも短時間で効率よく，スクリプトの結果を得ることができる．

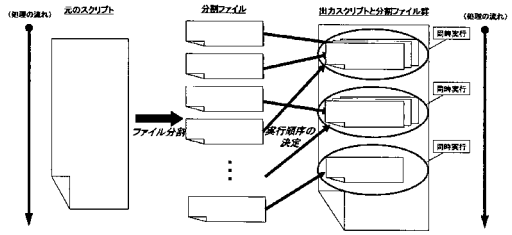

図2 提案手法イメージ

4.2 提案手法詳細

提案手法の全体の流れを図3に示す．はじめに，対象となるシェルスクリプトを指定し，ファイルの読み込みを開始する．ファイルの読み込みは１文字ずつ行い，空白や改行，その他指定文字（# や > など）を区切りにして文字列を取得し，以下の処理を行う．

1. リダイレクション（>, >>）による書き込みの発生するファイルの名前と行番号を取得し，ファイル名リストに加える．
2. ファイル名リストと一致する文字列を検索し，ファイルの再参照による依存関係を調べる．
3. 【分割ルール】に従いファイルの分割ポイントを決定し，ファイルを分割する．その際，同時に実行が不可能なファイルの番号も記憶しておく．
4. 分割ファイルの同時実行不可能ファイル番号を

参照し，同時実行可能ファイルを選び出し，分割ファイルの実行順序を決定する．

5. 変換後の実行シェルスクリプトとして，分割ファイルの実行順序を記した主スクリプトと，元のファイルを分割した分割スクリプトを生成する．

【分割ルール】

・基本的には，ファイルへの書き込みが行われたところまでをひとまとまりとし，その行を分割ポイントとする．

・ただし，例外として次の制御文中でファイルの書き込みが行われても分割を行わず，その制御文の終端行を分割ポイントとする．

1. if 文（if ～ fi 部分）
2. for 文（for ～ done 部分）
3. while 文（while ～ done 部分）
4. ヒアドキュメント書き込み中（<< "name" ～ "name"部分）

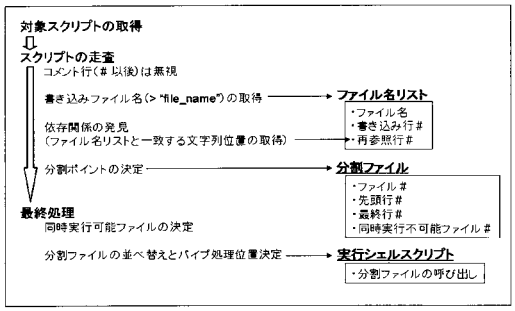

図 3　提案手法の流れ

5　評価実験

本節では，複数のマルチコアプロセッサ・SMT (Simultaneous Multi-Threading)マシン上での，シェルスクリプト実行結果を示す．実験の目的は，マルチコア・SMT プロセッサ上でのシェルスクリプト実行の有用性，および提案手法である「シェルスクリプトの自動並列化」の有効性を示すことである．

5.1　実験環境

本実験で用いたマシンの構成を表 5.1 に示す．

5.2　評価実験

実験は，業務処理プログラムを対象に行った．なお，実行時間の測定には bash の time コマンドを使用した．特に断りがない限り，実行時間は REAL の示す値とする．

【time コマンド】

・REAL：コマンド実行の実時間

・USER：ユーザ CPU 使用時間（コア総計）

・SYSTEM：システム CPU 使用時間（コア総計）

表 1　使用マシン構成

名称	SPARC T1	SPARC IV+	power5	Xeon 共有	Xeon 非共有	Opteron
マシン名	Sun Fire T2000	Sun Fire V490	IBM p5	Dell Precision 490	Dell Precision 490	PowerEdge SC1435
CPU	UltraSPARC T1 1.2GHz	UltraSPARC IV+ 1.5GHz	Power5 1.65GHz	Xeon 5110 1.6GHz	Xeon 5060 3.2GHz	Opteron 1.8GHz
実 CPU 数	1	2	1	2	2	2
論理 CPU 数	32	4	4	4	8	4
キャッシュ共有	あり	あり	あり	あり	なし	なし
メモリ量	16GB	16GB	2GB	16GB	16GB	4GB
HDD	SAS 10000rpm 73GB×2	FC 10000rpm 146GB×2	Ultra320 SCSI 10000rpm 73GB	Serial ATA 7200rpm 80GB	Serial ATA 7200rpm 80GB	SAS 15000rpm 73GB
パイプサイズ	5KB	5KB	32KB	4KB	4KB	5KB
OS	Solaris 10	Solaris10	AIX 5L	Red Hat Enterprise linux 4 WS	Red Hat Enterprise linux 4 WS	Red Hat Enterprise linux 4 WS

業務処理プログラム実験の目的は，実際の企業データとバッチ処理プログラムを用いたとき，マルチコア・SMT の恩恵をどの程度受けられるのかを調べることである．

業務処理プログラムは，データの解凍やテキスト変換などの基幹処理および帳票の作成をするプログラムである．プログラムはすべてシェルスクリプトで書かれている．業務処理プログラムは，大きく分けて次の３つのパートに分けられており，(1), (2), (3)の順番で実行される．

(1)　汎用データのテキスト形式変換および分割・ソート（*DATAMASTER*, 111 行）

ホストコンピュータで作成された汎用データを解凍し，EUC テキストファイルに変換する．その後，販売情報，在庫情報，仕入情報，初回仕入情報，物流センター間振替情報，各店舗間振替情報，差益情報，逆伝票情報に分解し，各生成ファイルをソートする．

(2)　商品勘定実績部門別帳票作成（*KANJO*, 1086 行）

各生成ファイルを集計，編集して，販売チャネル毎の各項目の予算比，昨年比を求め，エクセル形式の一帳票(勘定実績表)にまとめる．

(3)　商品勘定実績部門別詳細帳票作成（*KANJO.DEPA*, 3324 行）

上記の勘定実績表（*KANJO*）の各商品部門版を作成する．

今回の実験では，株式会社良品計画の実データを使用した．元データのファイルサイズは約１５０ＭＢ，解凍後に扱う総データは約４ＧＢである．また，最終的に出力される帳票データは約２５０ＫＢとなる．シングルコア，マルチコア・SMT（自動並列化適用前），マルチコア・SMT（自動並列化適用後）における，スクリプトの処理の実行時間を図 4 に示す．

自動並列化プログラムに通すことで，各スクリプトは以下の数のファイルに分割された．

- *DATAMASTER* : 27 ファイル

- *KANJO*：80 ファイル
- *KANJO.DEPA*：188 ファイル

これらの分割ファイルを，相互の依存関係を侵さないように同時実行する．同時実行ファイルは最高で *DATAMASTER* プログラムが 18，*KANJO* プログラムが 24，*KANJO.DEPA* プログラムが 70 ファイルになる．

実験の結果，通常のシェルスクリプトをマルチコア・SMT プロセッサ上で動作させるだけで，並列効果を得ることができた．また，自動並列化プログラムを使用すると，さらに並列効果を得ることができることが分かった．

DATAMASTER プログラムでは，もともと処理の重い部分がパイプライン処理で記述されているため，自動並列化プログラム適用前でも，平均使用論理 CPU 数が 1.5～3.1 と並列効果が高い．また，自動並列化プログラム適用後は，平均使用論理CPU数1.5～3.5となり，Power5を除いて実行時間ベースで1.2～1.5倍のスピードアップが図れた．なお，Power5 は途中でメモリエラーが生じ，自動並列化プログラムの恩恵を受けることができなかった．これは，Power5 のみメモリ容量が 2GB だったため，十分なメモリを確保できなかったことに起因する．

KANJO プログラム，*KANJO.DEPA* プログラムでは，自動並列化プログラム適用前は，平均使用論理 CPU 数が 1～1.3 と，ほとんど並列効果が現れなかった．しかし，自動並列化プログラム使用後は平均使用論理 CPU 数が 2～3.7 にまでなり，実行時間ベースで 1.8～3.0 倍のスピードアップが図れた．

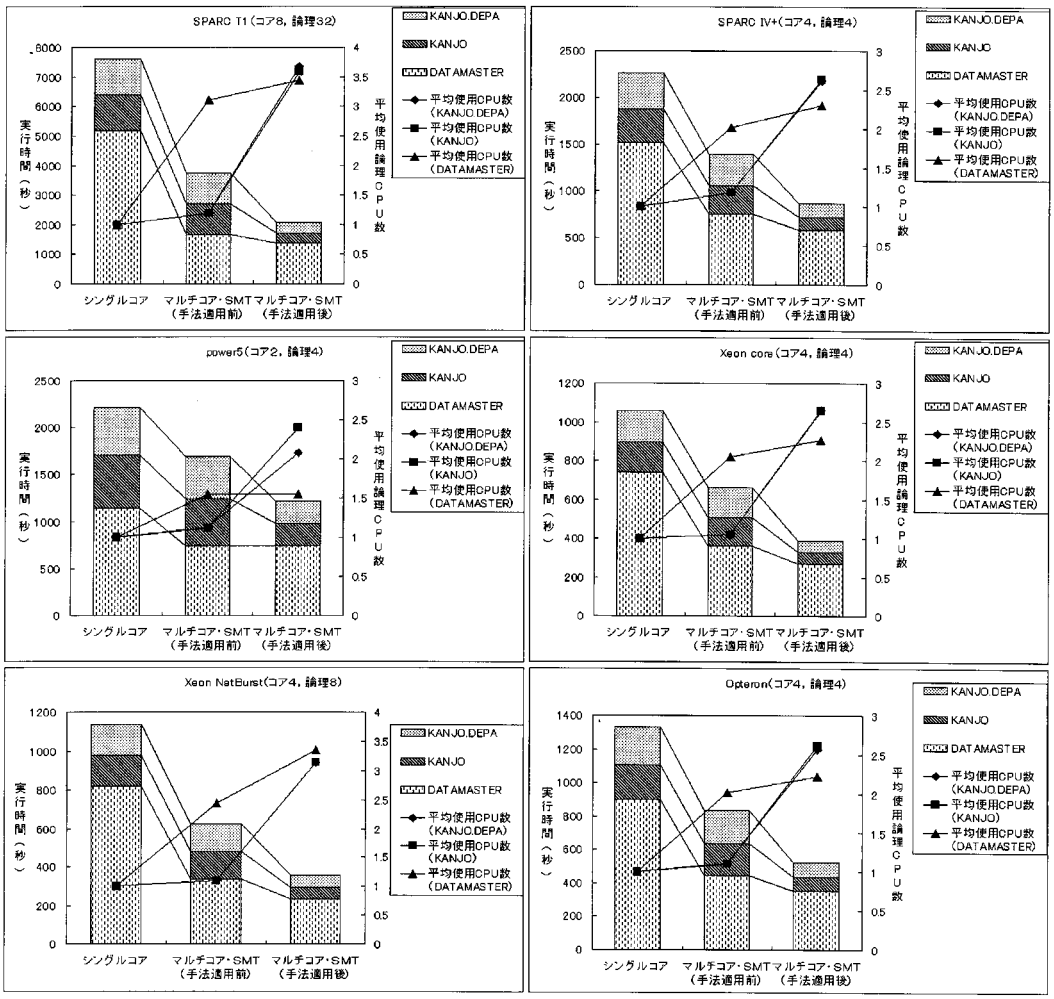

図 4 業務処理プログラム実行結果

マシン間で比較すると，論理 CPU 数の多い SPARC T1 と Xeon NetBurst マシンにおいて，KANJO プログラムの平均使用 CPU 数がそれぞれ 3.6，3.1，KANJO.DEPA プログラムそれぞれ3.7，3.1 となり，高い並列効果が得られた．

業務処理プログラム全体の実行効率を見ると，シェルスクリプトの自動並列化により，シングルコアに比べて，SPARC T1 では 3.7 倍，SPARC IV+では 2.6 倍，Power5 では 1.8 倍，Xeon Core では 2.7 倍，Xeon NetBurst では 3.2 倍，Opteron では 2.6 倍のスピードアップが図れた．また，シェルスクリプトの自動並列化適用前と比べても，SPARC T1 では 1.8 倍，SPARC IV+では 1.6 倍，Power5 では 1.4 倍，Xeon Core では 1.7 倍，Xeon NetBurst では 1.8 倍，Opteron では 1.6 倍のスピードアップが図れた．

6 関連研究

本研究は，シェルスクリプトを対象にした自動並列化の技術である．プログラムの自動並列化コンパイラについては，現在までに様々な研究がなされている．

自動並列化コンパイラの歴史は，１９７０年代に始まった．プログラムの実行時間の多くがループ処理であるという観点から，自動並列化コンパイラの研究は，長らくループ処理の並列化を中心に行われた[1,2,3]．しかし，ループ並列化の研究は，３０年にわたる研究によりほぼ成熟期に達しており，今後の大幅な性能向上は難しいと言われている[4]．そこで，近年自動並列化コンパイラの研究では，ループ並列化に加えて新たな並列化手法を組み込んだ研究がなされている．

[4,5,6]は，マクロデータフロー処理の技術を並列化コンパイラに導入している．マクロデータフロー処理とは，単一プログラム中のループ，サブルーチン，ブロック間の並列性を利用した，従来の並列処理よりも粗粒度の（マクロ）並列処理である．[4]では，RB (Repetition Block)，SB (Subroutine Block)，BPA (Block of Pseudo Assignment Statements)の３種類のマクロタスクを生成する．RB は最外殻ループ，SB はサブルーチン，BPA は並列性を考慮して融合または分割されたブロックである．これらの生成されたマクロタスク間の依存関係を解析し，並列性を考慮しながら実行コードを生成する．

また，[4]はメモリ利用の最適化技術を導入している．マルチコア・SMT プロセッサでは，メモリアクセスのレイテンシをカバーするためにキャッシュの有効利用が必要である．プログラムの解析時，データがキャッシュに収まるようにループとデータを分割する．また，一度キャッシュに載せたデータを複数のループで使用できるように，実行順序を調整する．

7 おわりに

本稿では，シェルスクリプト自身が持つ並列性に着目し，マルチコア・SMT プロセッサ環境において，シェルスクリプトの高速化を実現する手法，シェルスクリプトの自動並列化プログラムを提案した．その結果，マルチコア・SMT プロセッサ上での業務処理プログラム実行において，提案手法適用前に比べて，1.4～1.8 倍のスピードアップが図れた．

謝辞 本研究を行うにあたり、データの提供をしていただいた株式会社良品計画には感謝いたします。本研究の一部は、21 世紀 COE プログラム「プロダクティブ ICT アカデミア」および，科研費基盤 B「ヘルパースレッドを用いたマルチスレッディングプロセッサのための高速化技術研究」によるものである．

参考文献

[1] Z. Li, P. Yew, and C. Zhu: An Efficient Data Dependence Analysis for Parallelizing Compilers, In IEEE Trans. of Parallel and Distributed Systems, Vol.1, No.1, pp.26-34, 1990.

[2] R. P. Wilson, R. S. French, C. S. Wilson, S. P. Amarasinghe, J. M. Anderson, S. W. K. Tjiang, S-W. Liao, C-W. Tseng, M. W. Hall, M. S. Lam, and J. L. Hennessy: SUIF: an infrastructure for research on parallelizing and optimizing compilers, ACM SIGPLAN Notices, pp31-37, 1994.

[3] W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, T. Lawrence, J. Lee, D. Padua, Y. Paek, B. Pottenger, L. Rauchwerger, and P. Tu: Advanced Program Restructuring for High-Performance Computers with Polaris, IEEE Computer, pp.78-82, 1996.

[4] H. Kasahara, M. Obata, K. Ishizaka, K. Kimura, H. Kaminaga, H. Nakano, K. Nagasawa, A. Murai, H. Itagaki, J. Shirako: Multigrain automatic parallelization in Japanese Millennium Project IT21. Advanced Parallelizing Compiler, In IEEE International Conf. on Parallel Computing in Electrical Engineering, pp.105-111, 2002.

[5] H. Tanabe, H. Honda and T. Yuba: Macro-Dataflow using Software Distributed Shared Memory, IEEE International Conf. on Cluster Computing, 2005.

[6] M. W. Hall, S. P. Amarasinghe, B. R. Murphy, S-W. Liao, and M. S. Lam, "Interprocedural parallelization analysis in SUIF," In ACM Trans. on Programming Languages and Systems, Vol. 27, No. 4, pp.662-731, 2005

[7] Cory Isaacson, "Software Pipelines – An Overview," on White Paper, Rogue Wave Software.