様式Ｃ－１９

科学研究費助成事業（科学研究費補助金）研究成果報告書

平成２５年　５月３１日現在

機関番号：１４３０３
研究種目：基盤研究(C)
研究期間：2010～2012
課題番号：22500046
研究課題名（和文）　動的スカラ展開によるオンザフライ並列化に関する研究
研究課題名（英文）　A Research on the On-the-Fly Parallelization by a Dynamic Scalar Expansion
研究代表者
柴山　潔（SHIBAYAMA　KIYOSHI）
京都工芸繊維大学・工芸科学研究科・教授
研究者番号：70127091

研究成果の概要（和文）：　本研究では，スレッド間の逆依存と出力依存の関係を除去する手法として，ベクトル化コンパイラで用いられるスカラ展開を一般化したメモリリネーミングを提案している．メモリリネーミングでは，並列実行する各スレッドで共有する変数に対して，スレッドごとに格納場所を用意する．これにより，スレッド間の逆依存と出力依存の関係を除去して，スレッドレベルの並列性抽出の可能性を向上させることができる．シミュレーションによる評価の結果，評価した大部分のプログラムにおいて，スレッド間の依存関係によるハザードの半分以上を，本手法により除去できることを確認した．

研究成果の概要（英文）：　We propose a memory renaming as a generalization of the scalar expansion techniques which are employed in vectorizing compilers. The memory renaming expands memory area of a variable dynamically according to the number of threads which execute at the same time. This method can remove any anti-dependency and output-dependency among threads and increase the opportunities to extract thread-level parallelism. Our simulation results show that this method removes more than half of hazards caused by the dependencies among threads in most programs which we investigated.

交付決定額

（金額単位：円）

	直接経費	間接経費	合　計
２０１０年度	700,000	210,000	910,000
２０１１年度	1,100,000	330,000	1,430,000
２０１２年度	900,000	270,000	1,170,000
年度			
年度			
総　計	2,700,000	810,000	3,510,000

研究分野：情報工学
科研費の分科・細目：情報学・計算機システム・ネットワーク
キーワード：(1) 並列処理 (2) スカラ展開 (3) 並列化コンパイラ (4) プロセッサ

１．研究開始当初の背景

コンピュータの性能向上に関する研究の歴史は古く，これまでに様々な技術が開発されてきた．特に1990年以降は，コンピュータアーキテクチャの分野ではマシン命令レベルの並列性を活用して高速化を図る技術が盛んに研究され，半導体技術の進歩と相まって大きな発展を遂げた．その後，この命令レベル並列処理による性能の限界が見え始めると，次第に，命令レベルよりも粗粒度のス

レッドレベルの並列性の活用へと技術的着目点が移行した．その結果，今日では，パソコン用のコンピュータに搭載するマイクロプロセッサにおいても，複数のプロセッサ（コア）を設けてスレッドレベル並列処理を行うマルチプロセッサ構成を採用することが一般的となっている．

しかし，このマルチプロセッサ構成の潜在的な能力を十分に引き出すことは容易ではない．命令レベル並列処理においては，並列化コンパイラ技術によってプロセッサの能力を引き出すことが比較的容易であるのに対して，スレッドレベルの並列性を抽出して適切にスケジューリングを行うことは，一般には，コンパイラが行うのはもちろん，プログラマ自身が行うのでさえ困難である．マルチコアのマイクロプロセッサが市場に登場した数年前には，「スレッドレベル並列処理を本格的に研究する時代」の始まりであると叫ばれたが，実際には，従来からの並列処理研究の延長線上を脱しきれない状況が続いている．

命令レベルの並列処理では，マシン命令間の依存関係をチェックし，互いに依存しない複数の命令を同時に実行する．マシン命令間の依存関係は，主に，個々のマシン命令がどのレジスタを使用するかを調べることでチェックする．一般的なプロセッサの場合，プロセッサ内部に備えられているレジスタは高々30個程度であり，各マシン命令がどのレジスタを使用するのかをチェックすることは特に困難ではない．一方，スレッドレベルの並列処理では，一連のマシン命令の逐次実行を単位（スレッド）として，それが同時に実行可能であるか否かを判別しなければならない．すなわち，1個のスレッドが使用するレジスタが多くなることだけではなく，どのメモリアドレスにアクセスするのかもチェックしなければ，並列実行可能であるかどうかを判別できない．

命令レベル並列性に比べて，スレッドレベル並列性を抽出することが格段に難しい理由は， i) レジスタの数が高々30個程度と有限であることと比較すると，メモリは格段に大容量であるために，スレッド間の依存関係をチェックするだけで空間的かつ時間的なオーバヘッドが発生する； ii) どのメモリアドレスにアクセスするかが実行時でなければ決まらない場合があり，そのため，コンパイラだけでは並列化が不可能である；の2点である．

以上の問題があまりにも深刻なため，一般的なプログラムに対してコンパイラによるスレッドレベルでの並列化を適用することは困難であるとされていた．

２．研究の目的

本研究に先立って，単一スレッド実行を前提としてプログラミングして作成したソースプログラムに対して，これをコンパイラによって並列化する場合に生じる並列化の阻害要因について調査した．その結果，単一の変数の使いまわしによってスレッド間の依存関係が生じている場合が非常に多く，これを取り除けば，多くのプログラムでスレッド間の依存関係を取り除くことができることを見付けた．このような種類の依存関係は，ベクトル化コンパイラの代表的な技術であるスカラ展開(scalar expansion)によって取り除くことができる．

しかし，ベクトル化コンパイラのスカラ展開技術の適用範囲は狭く限られており，一般のプログラムに対して広く用いることはできない．そこで，本研究では，スカラ展開のアイデアを一般化して，任意のプログラムに対して，かつ，実行時に(on-the-fly)，適用する動的スカラ展開方式を開発することで，スレッド間に存在する見かけ上の依存関係を取り除き，並列性の抽出可能性を大幅に向上させる．具体的な目標は以下の通りである．

(1) 「スカラ展開を適用するのが効果的な変数はどれか」を動的にすなわち実行時に検出・判別する技術を開発・提案する．
(2) 動的に変数をスカラ展開する仕組みを開発・提案する．本研究ではスカラ変数を必ずしもベクトル変数に展開するのではなく，マルチスレッド実行に応じて適切な形式およびメモリ量でメモリ領域を割り当てる．
(3) 並列処理効果を最大限に引き出すために，上記(1)(2)の技術をオンザフライで（実行時に）最適に制御する技術を確立する．
(4) シミュレーションによる性能評価を行い，本研究の効果を検証する．

３．研究の方法

(1) プログラムの並列実行時に，a) メモリ上のどのデータに対してスレッド間の依存関係を破壊するアクセス（メモリバイオレーション）が発生したのかを検出する；b) そのデータをスカラ展開することによって並列実行が可能になるかどうかを判別する；の各方法について検討する．

(2) 動的スカラ展開を行う方式について，その実装をも考慮して検討する．スカラ展開のためのメモリ領域の確保，マシン命令からそのメモリ領域を参照するための仕組み，などが具体的な検討項目であり，実施コストを考慮しながら検討を進める．

(3) オンザフライ並列化によって可能となるスレッドの並列実行のための基本的な制御機能について検討する．

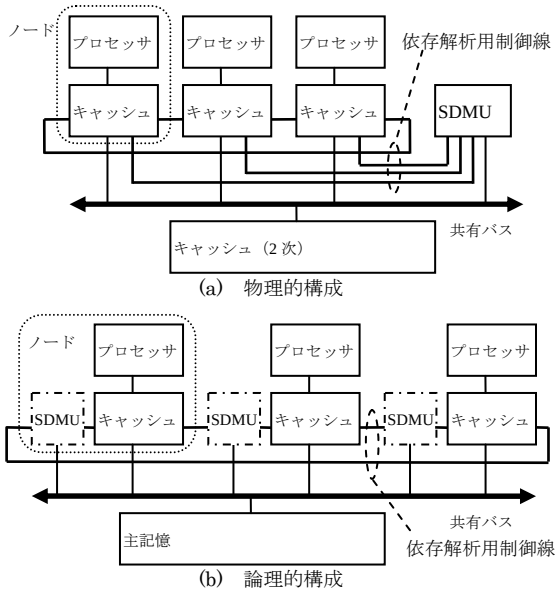

図１　システム構成

(4) 上記(1)～(3)の結果をもとに，性能評価のための時間シミュレータを開発する．

(5) (4)で開発するシミュレータによって性能評価を行い，本研究の有効性を検証する．

４．研究成果

(1) スカラ展開を一般化し，オンザフライでスレッドレベル並列化を行う具体的な方式として，メモリリネーミングを開発した．

本方式を用いるマルチプロセッサシステムの構成を図1に示す．スヌープキャッシュプロトコルを利用してスレッド間の依存解析を行うための専用機構を1次キャッシュに設け，依存関係を破壊するハザードを検出した場合には，それに関連する投機スレッドのみをアボートして，最初の命令から再実行する．各プロセッサの1次キャッシュにおいて，スレッド間で共有する変数の別バージョンを格納することで，メモリ上のデータに対してリネーミングを行い，スレッド間の逆依存と出力依存を除去する．

1次キャッシュにはコピーバック方式を採用し，共有バス側からはその状況に応じてあたかも1次キャッシュまたは2次キャッシュであるかのように動作する投機データ管理ユニット(SDMU; Speculative Data Management Unit)を設けることで，キャッシュあふれに対応する．SDMUは，1次キャッシュから追い出される投機データのメモリアドレスを別のアドレスにマッピングして，2次キャッシュ（または2次キャッシュ以降の階層のメモリ）に記憶し，共有バス上ではSDMU内に記憶しているかのように振る舞う．2次キャッシュ以降の階層のメモリに記憶している投機データは，アプリケーションプログラムからは別のアドレスのデータとして可視となるが，逆に，投機データをバックアップするための領域をアプリケーションのアドレス空間内にあらかじめ確保しておかなければならない．

表1　アクセス状態の遷移（ライン単位）

現状態 (W's, FR, O)	次状態 (W's, FR, O)		
	自Read	自Write	先行Write
($^{\forall}$0, 0, 0)[注1]	($^{\forall}$0, 1, 0)	($^{\exists}$1, 0, 0)	—
($^{\forall}$0, 1, 0)	($^{\forall}$0, 1, 0)	($^{\exists}$1, 1, 0)	ABORT
($^{\exists}$1, 0, 0)	($^{\exists}$1, 0, 0), ($^{\exists}$1, 1, 0)	($^{\exists}$1, 0, 0)	($^{\exists}$1, 0, 0), ($^{\exists}$1, 0, 1)
($^{\exists}$1, 1, 0)	($^{\exists}$1, 1, 0)	($^{\exists}$1, 1, 0)	($^{\exists}$1, 1, 0), ABORT
($^{\exists}$1, 0, 1)	($^{\exists}$1, 0, 1), ($^{\exists}$1, 1, 0)	($^{\exists}$1, 0, 1)	($^{\exists}$1, 0, 1)

注1) 初期ミスによってキャッシュにロード（アロケート）した直後の内部初期状態であり，実際にはこのような状態は存在しない．

1次キャッシュから追い出された投機データについても，スレッド間の依存解析の対象としなければならないので，その処理をSDMUが担当する．従って，SDMUは，物理的には，図1(a)に示すように1台であるが，論理的には，図1(b)に示すように各1次キャッシュと仮想的に対を構成し，1次キャッシュの容量を拡張する機能を提供する．

(2) メモリ上のデータに対するスレッド間の依存解析を行うための依存解析機構を設計した．

1次キャッシュの各ラインに以下のアクセス状態ビットを設ける．

① W (Written)ビット：ライン内の各バイト領域に対応付けて設け，そのバイト領域に書き込みを行ったことを示す．
② FR (First Read)ビット：ライン中の少なくとも1個の変数に対して，そのスレッドにおける最初のアクセスが読み出しであることを示す．
③ O (Obsolete)ビット：ライン中のまだ書き込みを行っていない変数のメモリアドレスに対して，先行スレッドが書き込みを行ったことを示す．

プロセッサが各自の1次キャッシュに書き込みを行うとき，対応するWビットをセットする．また，Wビットがセットされていない領域から読み出す場合，FRビットをセットする．Oビットについては，自プロセッサからのアクセスではなく，先行スレッドを実行しているプロセッサが書き込みを行ったことをバススヌーピングによって検出し，ハザードが発生していないことを確認する．ハザードが検出されず，かつ，その書き込みアドレスに対応するWビットがセットされていなければ，Oビットをセットする．

各1次キャッシュは，プロセッサ側からリード要求とライト要求を，また，共有バス側

からロード要求と依存解析要求を受け付け，アクセス状態ビットを用いて各ラインの状態を管理する．その状態遷移を表1にまとめてある．

(3) メモリ上のデータに対するバージョン管理機構を設計した．

確定スレッドが終了すると，隣接するプロセッサで実行中の投機スレッドをコミットし，確定スレッドとして実行を継続する．コミットされたデータをキャッシュに残したまま，順次，投機スレッドがコミットされて行く状況では，確定データについても複数のバージョンが存在することになる．本方式では，投機データ間のバージョンの前後関係はリング接続の制御線を用いることで固定するが，確定データについては，投機実行中からアクセスしていたか，あるいはコミット後に初めてアクセスしたかによって，必ずしも順序を固定できない．そこで，本方式では，最新の確定バージョンを保持することに責任を持つキャッシュが必ず1個だけ存在するように制御する．そのため，1次キャッシュのラインごとに以下の状態ビットを設ける．

① L (Loaded)ビット：各プロセッサ（ノード）に対応付けて設け，後続の投機スレッドを実行するノードのキャッシュがこのラインをロードしたことを示す．

② P (Predecessor)ビット：複数個存在する確定データの中で，最新バージョンではないことを示す．

他のノードからロードされるとき，Lビットをセットする．しかし，その後，ロードしたノードで実行中の投機スレッドがアボートする場合もあり得る．このとき，そのノードは共有バスを用いてアボートしたことを全ノードに通知する．その通知を受けて，各キャッシュでは，Lビットを一斉にリセットする．また，投機スレッドをコミットする場合も，同様に共有バスを用いて通知するので，各キャッシュでは，コミットしたノードに対応するLビットがセットされているラインに対して，そのPビットをセットする．

本方式では，ライン単位でかつバイト領域へのアクセスを考慮して依存関係を管理する．従って，LビットとPビットを用いて共有バス上で最新バージョンのラインから古いバージョンのラインの順に再帰的に検索し，Wビットを参照しながらマージする．高々ノード数分の連続バスサイクルを要するが，マージデータが完成した時点でそれより古いバージョンのラインを一斉に無効化することによって時間短縮を図る．その後，マージされたラインのデータに対するアクセスは，2次キャッシュ（メモリ）アクセスと同様に扱うことができる．

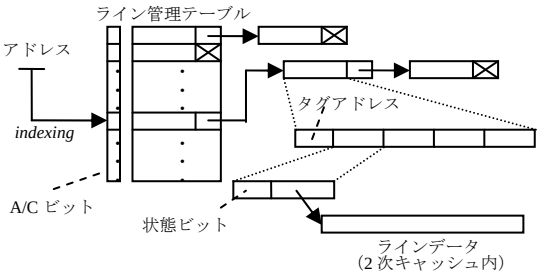

図 2　SDMU におけるライン管理

(4) 投機データ管理ユニット(SDMU)におけるラインデータの管理機構と，その処理の最適化方式を設計・開発した．

1次キャッシュのライン置換において，追い出さなければならない投機データは，そのラインの状態ビットも含めSDMUに転送してバックアップする．SDMU内においてそれらのバックアップデータを管理するためのデータ構造を図2に示す．ライン管理テーブルはダイレクトマッピング方式のキャッシュと同様の検索機能を備え，マッピングコンフリクトが発生する場合は，連結リストを用いて管理する．ライン管理テーブルの各エントリはそのアドレスに対応するすべてのノードの1次キャッシュの状態ビットとデータを記憶する．ただし，ラインデータ自体は2次キャッシュに保存する．1次キャッシュからラインを追い出す場合は，SDMUは2次キャッシュであるかのように振る舞い，また，そのラインデータを再び1次キャッシュにロードする場合は，他のノードの1次キャッシュのように振る舞うことで，スヌープキャッシュプロトコルに対する変更を最小限に留める．また，SDMUは2次キャッシュではないので，同一の投機データは，1次キャッシュかSDMUのいずれか一方にのみ存在する．

スレッドのコミットやアボートについては，その都度，共有バスを用いてSDMUにも通知する．このとき，SDMUで管理しているすべてのデータに対して無効化や確定処理を行うと，SDMUが性能上のボトルネックとなりかねない．そこで，これらの処理をいずれかの1次キャッシュのライン置換が発生するまで遅延させて，時間的に分散処理するために，ライン管理テーブルの各エントリを各ノードに対応させて， i) アボートの通知があったことを示すA (Abort)ビット； ii) コミットの通知があったことを示すC (Commit)ビット；を設ける．それぞれの通知があった場合にそのノードに関するこれらのビットを一斉にセットし，ライン置換等の処理を行う際に，例えばAビットがセットされていれば，連結リストをたどりながら該当ノードのラインを無効化した後，Aビットをリセットする．なお，コミットやアボートの通知を検知した際，すでにA,Cビットのいずれかがセ

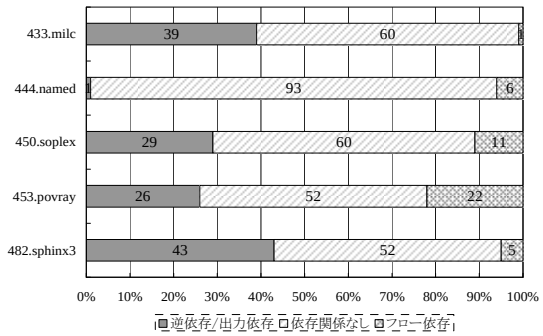

図３　依存関係に関する命令の分類

ットされている場合には，変更は行わない．その理由は，例えばAビットがセットされている状態でコミットの通知を受けた場合，Aビットがセットされてから新たにラインが追加されてはいないので（もしも，追加されていれば無効化の処理を行って，Aビットもリセットしているはずである），連結リスト中に確定すべきラインデータは存在しないからである．

(5) シミュレータを作成し，SPEC CPU 2006ベンチマーク中のプログラムを対象とする性能評価を行った．

スレッドとして実行するすべての命令を，a) 逆依存および出力依存によるハザードが生じる命令； b) フロー依存によるハザードが生じる命令； c) 依存関係によるハザードが生じない命令（ロード/ストア命令以外の命令も含む）；に分類して，それぞれの割合を調べた結果を図3に示す．これによって，逆依存または出力依存によるハザードが多く（最大約43%）生じることが判明し，メモリリネーミングによってそれらを取り除くことにより，並列性抽出の機会が劇的に増加することが確認できた．

５．主な発表論文等

（研究代表者，研究分担者及び連携研究者には下線）

〔雑誌論文〕（計 11 件）

① 平田博章，藤井崇弘，藤皓平，森田清隆，布目淳，柴山潔．スレッドレベル並列投機実行のためのメモリリネーミング機構．第 11 回情報科学技術フォーラム講演論文集，査読有，Vol.1，pp. 31-34，2012.

② 藤皓平，布目淳，平田博章，柴山潔．動的可換性解析によるスレッドレベル並列投機実行方式，第 11 回情報科学技術フォーラム講演論文集，査読無，Vol.1，pp.267-268，2012.

③ 安井寛幸，布目淳，平田博章，柴山潔．基本ブロック単位の命令実行パス予測方式，第 11 回情報科学技術フォーラム講演論文集，査読無，Vol.1，pp.269-270，2012.

④ 中務国男，山田徹，布目淳，平田博章，柴山潔，スタックスマッシング攻撃の正確な検出機構を備えたプロセッサアーキテクチャ，第 10 回情報科学技術フォーラム論文集，査読有，Vol.1，pp.63-66，2011.

⑤ 藤井崇弘，森田清隆，布目淳，平田博章，柴山潔，スレッドレベル並列化のためのメモリリネーミング，第 10 回情報科学技術フォーラム論文集，査読無，Vol.1，pp.385-388，2011.

⑥ 赤坂謙二郎，布目淳，平田博章，柴山潔，電子商取引サイトにおける応答待ち時間の短縮を目的としたデータベース参照のスケジューリング方式，第 10 回情報科学技術フォーラム論文集，査読有，Vol.1，pp.113-116，2011.

⑦ 布目淳，平田博章，柴山潔，IPv6 環境におけるネットワーク認証のためのマルチキャストフィルタリングイーサネットスイッチ，第 10 回情報科学技術フォーラム論文集，査読有，Vol.4，pp.1-4，2011.

⑧ A.Nunome，H.Hirata，M.Fukuzawa，and K.Shibayama，Development of an E-learning Back-end System for Code Assessment in Elementary Programming Practice，SIGUCCS Fall 2010 Conference，ACM，査読有，pp.181-186，2010.

⑨ 平田博章，山田徹，布目淳，柴山潔，マシン命令レベルでのプログラム実行モニタリングによる手続き呼び出し関係の正確な検知方式，第9回情報科学技術フォーラム論文集，査読有，Vol.1，pp.87-90，2010.

⑩ 森田清隆，布目淳，平田博章，柴山潔，スレッドレベル並列化のためのスレッド間依存関係の分類，第9回情報科学技術フォーラム論文集，査読有，Vol.1，pp.81-86，2010.

⑪ 赤坂謙二郎，布目淳，平田博章，柴山潔，データベース参照のスケジューリングによる電子商取引サイトの最適化，第 9 回情報科学技術フォーラム論文集，査読無，Vol.1，pp.407-408，2010.

〔学会発表〕（計 2 件）

① 藤井崇弘，森田清隆，布目淳，平田博章，柴山潔，スレッドレベル並列投機実行のためのメモリリネーミング機構，電子情報通信学会 2012 年総合大会 学生ポスターセッション，於・岡山大学，2012 年 3 月 22 日．

② 安井寛幸, 布目淳, 平田博章, 柴山潔, 基本ブロック単位の命令実行パス予測方式, 電子情報通信学会 2012 年総合大会 学生ポスターセッション, 於・岡山大学, 2012 年 3 月 22 日.

６．研究組織

(1) 研究代表者

柴山　潔（SHIBAYAMA　KIYOSHI）

京都工芸繊維大学・工芸科学研究科・教授

研究者番号：70127091

(2) 研究分担者

平田　博章（HIRATA　HIROAKI）

京都工芸繊維大学・工芸科学研究科・准教授

研究者番号：90273549

布目　淳（NUNOME　ATSUSHI）

京都工芸繊維大学・工芸科学研究科・助教

研究者番号：60335320

(3) 連携研究者

（　　）

研究者番号：