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

2007－ARC－173 (16)
2007／6／1

ALU Cascadingを行う動的命令スケジューラ

尾形 幸亮† 姚 駿† 三輪 忍† 嶋田 創† 富田 眞治†

† 京都大学大学院情報学研究科
† 京都大学大学院法学研究科

あらまし ALUの出力を別のALUの入力につなぎ，1クロック・サイクル中にデータ依存関係にある命令列を複数実行する，ALU Cascadingという手法がある．この手法をスーパスカラ・プロセッサに適用する場合，ALU Cascading可能な組を同時にwakeup可能な命令スケジューラが必要となる．本論文では，このALU Cascadingを行える命令スケジューラについて提案を行う．提案する命令スケジューラをSPECint95を用いて評価した結果，2段のALU Cascadingを行うと，IPCが平均で6.6%向上するという結果になった．

The Dynamic Instruction Scheduler for ALU Cascading

Kosuke OGATA†, Jun YAO†, Shinobu MIWA†, Hajime SHIMADA†, and Shinji TOMITA†

† Graduate school of Informatics, Kyoto University
† Graduate school of Law, Kyoto University

Abstract There's a technique called ALU cascading which executes several instructions under data dependency relationship in one clock cycle. Such execution is achieved by concatenating the output of the ALU into the input of the other ALU. To implement this technique to current superscalar processor, we have to prepare instruction scheduler which can wakeup pair of cascaded instructions simultaneously. In this paper, we propose the instruction scheduler which enables ALU cascading. The evaluation result shows that the IPC of SPECint2000 improves by 6.6% in average with 2 level ALU cascading.

1. はじめに

プロセッサの性能を向上させる時の問題点の1つに，データ依存関係にある命令列の実行をいかに高速化するかという点がある．この高速化のため，現在のプロセッサでは結果フォワーディングという技術が使われている．これは，データ依存関係にある命令列を実行する際，先行する命令の実行結果をレジスタ・ファイルを介さずに後続の命令に渡す技術である．これにより，データ依存関係にある命令列の演算を連続して実行することができる．このような命令列が，演算を1クロック・サイクルで完了できる単純な演算命令からなる場合，命令のスループットは1クロック・サイクルあたり1命令となる．

このスループットをさらに向上させる提案として，ALU Cascadingがある．これは，図1のように，あるALUの出力を別のALUの入力につなぎ，データ依存関係にある命令を1クロック・サイクルで実行するものである．

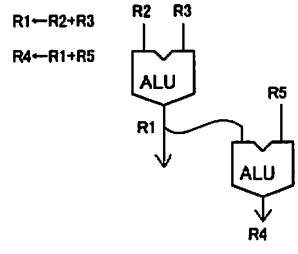

図1 ALU Cascading

このALU Cascadingの適用例として，過去にはベクトル・プロセッサ[6]やメディア・プロセッサ[7]への適用例があったが，一般的なプロセッサへの適用例は少なかった．これは，一般的なプロセッサではALUでの演算時間を基準にクロック・サイクル時間を決定することが多いため，クロック周波数を落とさないとALU Cascadingを使った実行を1クロック・サイクルで行えないからである．しかし，近年では，プロセッサの負荷に応じてク

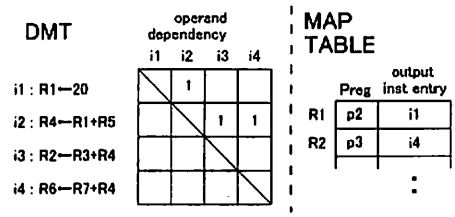

図 2 DMT の行列表とマップ表

ロック周波数を低下させ，消費電力を削減する研究が多数行われている．そのため，クロック周波数を低下させた状況で ALU Cascading を適用することが提案されている [3], [10].

文献 [3], [10] のように，スーパスカラ・プロセッサに ALU Cascading を適用する場合，命令ウィンドウに入っている命令から ALU Cascading を行う組を選択することになる．過去の研究では，限定された命令の組に対して ALU Cascading を行う命令スケジューラは提案されているが [3]，任意の命令の組に対して ALU Cascading を行う命令スケジューラは提案されていない．もちろん，通常の命令スケジューラにおいても，1 クロック・サイクル中に wakeup-select を複数回行えば，任意の命令の組に対する ALU Cascading は可能である．しかし，消費電力やクロック・サイクル時間が増大する点から，wakeup-select を複数回行うというのは好ましくない．

本論文では，DMT(Dependency Matrix Table) [5] をベースとし，1 回の wakeup-select で任意の命令の組に対する ALU Cascading を行う命令スケジューラを提案する．本スケジューラの実現のため，ベースとなる DMT の依存行列表 (以下，行列表) とレジスタ・マップ表 (以下，マップ表) に拡張を加えるが，いずれも，クロック・サイクル時間を増大させる必要がない形で拡張を行っている．

本論文の構成は以下の通りである．まず 2 節では DMT の構成及び動作について述べ，次に 3 節で，DMT を拡張した ALU Cascading を行うための仕組みについて述べる．4 節ではプロセッサ・シミュレータによる評価結果を示し、IPC が向上することを示す．そして 5 節で関連文献について述べ，最後に 6 節でまとめを述べる．

2. Dependence Matrices Table(DMT)

DMT は，命令発行ステージにおいて命令の wakeup-select 動作を高速化するために考案された手法である．DMT は，一般的な命令スケジューリング論理における CAM のタグ部に代わって配置される．その論理的な構造は図 2 左のようになっており，行列の縦横のサイズは命令ウィンドウのエントリ数となっている．行列の各要素は 1 ビットの SRAM である．行列の縦横に振られた番号は命令ウィンドウ内における命令番号に対応している．また，行列の行方向は依存元となる命令番号，列方向は依存先となる命令番号を表しており，命令 m の演算結果が命令 n のソース・オペランドとなっているとき，m 行 n 列にフラグが立つ．例えば，図 2 では，1 行 2 列目にフラグが立っているが，これは命令 i1 のディスティネーションが命令 i2 のソース・オペランドとなっているためである．2 行 3 列目および 2 行 4 列目にある 2 つのフラグも，同様に i2 と i3，および i2 と i4 の依存関係を示している．上記の DMT 更新作業は，一般的な命令スケジューリング論理と同様に wakeup と select によって行われる．wakeup では，前サイクルで select された命令の命令番号の行を全て 0 にし，それによって生成された全てが 0 の列について，対応する命令の ready ビットを立てる．select は従来の命令スケジューラと同様，命令ウィンドウにおいて ready ビットが立っている命令の中から発行すべき命令を選択する．また，DMT を用いた命令スケジューラでは，図 2 右のように，output inst entry という列が追加される．output inst entry は，その論理レジスタ番号に結果を出力する最新の命令の，命令ウィンドウにおける番号を表す．output inst entry は命令デコード時に書き込まれ，2.1 節で示すように DMT の更新に使用される．

2.1 DMT の動作アルゴリズム

図 3 に示す例を用いて DMT の動作を説明する．図 2(a) の初期状態では，命令 i1,i2 がすでに命令番号 1,2 として DMT に登録されている．i2 は i1 に依存しているため，DMT の 1 行 2 列目に 1 が立っている．以下，i3,i4 を新たにデコードし，命令番号 3,4 として DMT に登録し，命令スケジューリングを行う様子を示す．なお，デコード幅と発行幅は 2 命令とする．

(1) 命令デコードとマップ表読み出し

i3,i4 のソース論理レジスタ番号で図 3(a) の初期状態のマップ表を検索し，物理レジスタ番号を得る．同時に，まだ値が生成されていない論理レジスタについては，その物理レジスタに値を出力する命令の命令番号を得る．図の例では，i3,i4 のソースの R4 のみ命令番号を得る．また，同時にデコードされた命令の間のデータ依存関係をチェックし，必要があれば，後続の命令に先行する命令の命令番号を渡す．i3,i4 間には依存関係はないので，この部分による命令番号の追加はない．

(2) マップ表の更新

i3,i4 のデコードにより，i3,i4 に DMT の命令番号 3,4，および，物理レジスタ p2,p6 が割り当てられたとする．この時の動作は，図 3(b) に示すように，マップ表の R2,R6

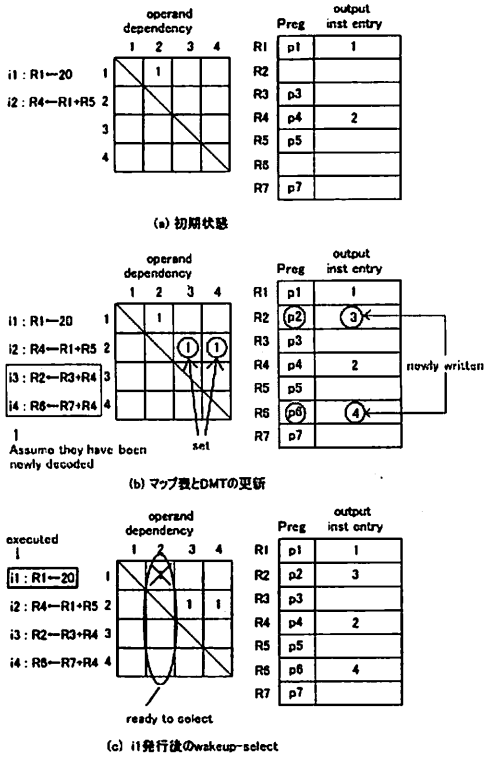

図 3　DMT の動作アルゴリズム

のエントリに割り当てられた物理レジスタとその物理レジスタに結果を出力する i3,i4 の命令番号を書き込むことになる．

(3) DMT の更新

(1) のマップ表読み出しにより，命令番号 3 のエントリに書き込まれる i3 は命令番号 2 に依存し，命令番号 4 のエントリに書き込まれる i4 は命令番号 2 に依存していることが分かった．そのため，DMT の 2 行 3 列と 2 行 4 列に 1 を立てる．図 3(b) に更新後の DMT を示す．

(4) i1 の発行後の wakeup-select

1 が発行されると，図 3(c) のように DMT の 1 行目のエントリ全てに 0 が書き込まれる．これにより，2 列目が全て 0 になるため，i2 が ready となる．select において，ready な命令は i2 のみのため，i2 のみが select される．なお，発行された i1 のマップ表の output inst entry は，発行と同時にクリアされる．

なお，次のサイクルの wakeup-select では，2 行目に全て 0 が書き込まれ，3 列目と 4 列目が全て 0 となり，i3,i4 が ready となる．

3. DMT を用いた ALU Cascading

図 3(b) で例に示している命令列では，i2→i3, i2→i4 の ALU Cascading が可能である．この例の場合，i2 が ready

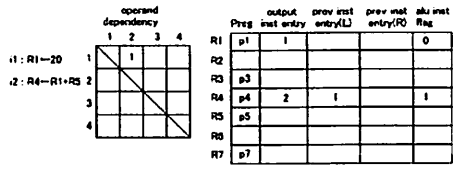

図 4　拡張されたマップ表

となると同時に i3,i4 も ready になるような構成を取れば，i2 と i3，i4 を同時に select し，ALU Cascading ができる．このように，2 つ前のデータ依存先の命令で，後続の命令の wakeup を行うことが，DMT を用いた ALU Cascading の命令スケジュールの概要である．DMT を用いて ALU Cascading を行う意義は，1 クロック・サイクルあたり 1 回の wakeup-select で ALU Cascading が可能なことである．従来の命令スケジューラで ALU Cascading を実現するためには，wakeup-select を 1 クロック・サイクル内で 2 回行わなければならず，消費電力やクロック・サイクル時間の問題から ALU Cascading の実現は難しかった．そこで，本稿で提案する方式では，ALU Cascadig に必要な wakeup-select 回数を 1 回に抑え，上記の問題を克服することができる．以下，DMT を用いた ALU Cascading を行うためのハードウェアの拡張とその動作を説明する．

3.1　マップ表の拡張

ALU Cascading の実装は，3 節で述べた DMT のアルゴリズムを拡張する形で行う．まず，図 4 のようにマップ表を ALU Cascading 用に拡張し，prev inst entry(L), prev inst entry(R) および alu inst flag を追加する．prev inst entry は，output inst entry の示す命令が持つソース・オペランドが，どの命令の出力に由来するかを記憶する．たとえば図 4 では，論理レジスタ R4 に結果を出力する命令は i2 であるため，マップ表において R4 に対応する output inst entry には 2 と記憶されている．ここで，更に i2 の左ソース・オペランドが i1 の出力に依存しているので，マップ表の同じ行において prev inst entry(L) に 1 と記憶されている．なお，prev inst entry は命令が持てるソース・オペランドの数だけ必要である．ALU inst flag は，その論理レジスタに結果を出力する命令が ALU 演算命令であることを示すフラグである．このフラグが 1 の場合は 2 つ前のデータ依存先の命令で wakeup を行い，それ以外の命令は従来と同様に 1 つ前のデータ依存先の命令で wakeup を行うようにする．

3.2　DMT を用いた ALU Cascading のアルゴリズム

2 段の ALU Cascading を行う場合の，DMT への登録やマップ表の更新を，図 4 および図 5 の例を使って説明する．2 節と同様に，図 4 の初期状態では命令 i1,i2 がす

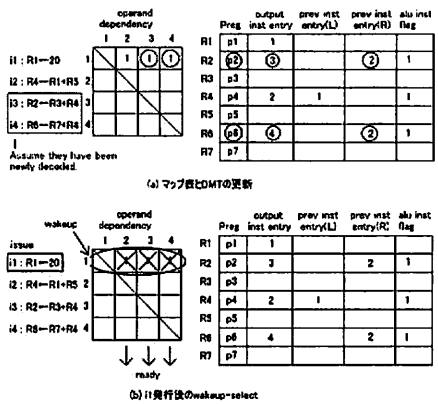

図 5　ALU Cascading のアルゴリズム

でに命令番号 1,2 として DMT に登録されており，DMT の 1 行 2 列目にフラグが立っている．また，マップ表において i2 に対応する prev inst entry(L) には 1 が記憶されている．i1 は即値ロード命令なので ALU inst flag は 0 となり，i2 は ALU 演算命令なので ALU inst flag は 1 となっている．

(1) 命令デコードとマップ表読み出し

図 5(a) のように，新たに命令 i3，i4 がデコードされたとする．まず，2.1 節 (1) と同様に，i3,i4 のソース論理レジスタ番号でマップ表を検索し，物理レジスタ番号を得る．まだ値が生成されていない論理レジスタについては，その物理レジスタに値を出力する命令の命令番号を得る．図 5(a) の例では，i3,i4 のソースの R4 で検索し，ヒットしたエントリの output inst entry から i2 の命令番号 2 を得る．さらに同じエントリの prev inst entry(L) から i1 の命令番号 1 を得る．さらに，ALU inst flag より，この論理レジスタに結果を出力する命令は ALU 演算命令であることを知る．

(2) マップ表更新

2.1 節 (2) と同様に，i3,i4 に DMT の命令番号 3,4，および，物理レジスタ p2,p6 が割り当てられたとする．マップ表の R2,R6 のエントリに割り当てられた物理レジスタと，その物理レジスタに結果を出力する i3,i4 の命令番号を書き込む．さらに，i3，i4 ともに右ソース・オペランドが i2 に依存しているため，それらのエントリの prev inst entry(R) に本節 (1) で得られた i2 の命令番号 2 を書き込む．

(3)DMT 更新

(1) のマップ表読み出しで，i3，i4 が依存している i2 は i1 に依存していることが分かった．また，ALU inst flag より，i2 は ALU 演算命令であり，ALU cascading に組み込めることも分かった．そこで，通常の DMT では output inst entry から得られた命令番号 2 を用いて 2 行 3 列と 2 行 4 列に 1 を立てるところを，ここでは prev inst entry から得られた命令番号 1 を使って 1 行 3 列と 1 行 4 列に 1 を立てる．

このような手順で，2 つ前のデータ依存先の命令に対する依存を DMT に記述する．なお，図の例では 2 つ前のデータ依存先の命令は 1 つしかないが，最大で 4 個の命令に依存することが考えられる．このように，複数のデータ依存先がある場合でも，DMT は 1 列中の複数のエントリに 1 を立てることによって複数の依存を記述できるため，問題はない．

(4)i1 の発行後の wakeup-select

i1 が発行されると，図 5(b) のように DMT の 1 行目のエントリ全てに 0 が書き込まれる．2，3 および 4 列目は全て 0 になるので，i2，i3 および i4 は同時に ready となる．このように，(3) で DMT の更新に prev inst entry の情報を用いたことにより，依存関係にある演算命令を一度に ready にすることができる．これら ready となった演算命令は，別々の空いている ALU に発行され，実行ステージのクロック・サイクル時間の前半で i2 を実行中に，ALU 間の接続情報を作成し，ALU Cascading 実行のためのデータ・パスを構築する．この部分の動作は，通常のプロセッサにおける結果フォワーディングの論理と同様である．なお，ALU に空きがなく，全ての演算命令を一度に発行できない場合は，命令ウィンドウ内において古い命令から順に発行できる分だけ発行される．古い命令から発行するため，i3 や i4 のように先行命令に依存している命令だけが発行されることはない．なお，図 5 に示した例では，i2→i3 と i2→i4 のように，ALU Cascading を適用できる組が 2 つあるが，提案構成では両方の組に対して同時に ALU Cascading を適用できる．以下，これを，1 対多の ALU Cascading と呼ぶ．文献 [9] の方式では，このような 1 対多の ALU Cascading はできず，提案構成が文献 [9] よりも高い性能を得られる理由の 1 つとなる．

3.3　複数段の ALU Cascading

図 1 では 2 つのデータ依存関係にある命令列の実行について説明したが，クロック・サイクル時間に余裕がある場合，さらに多くのデータ依存関係にある命令を 1 クロック・サイクルで実行することもできる．例えば，"命令 i1 →命令 i2 →命令 i3 →命令 i4"というデータ依存関係にある命令列がある場合，4 つの演算器を準備しその入出力を適切に接続すれば，この命令列を 1 クロック・サイクルで実行することができる．以下，このように n 個の命令を ALU Cascading で実行することを，"n 段の

ALU cascading"と記述する.

提案手法では，マップ表の拡張のみでn段のALU cascadingを実現可能である．具体的には，図4の2段のALU cascadingの例ではマップ表は2つ前のデータ依存先の命令の命令番号を保持していたが，これを，n個前のデータ依存先の命令の命令番号まで持てるように拡張すれば良い．この場合の表の更新は，3.2節の表の更新アルゴリズムの中で，マップ表から読み出したのk-1(k=2,...,n-1)個前のデータ依存先の命令番号を，マップ表への書き込み時にk個前のデータ依存先として書き込むだけでよい.

上記のようにn段のALU Cascadingをサポートする場合，マップ表のサイズは指数関数的に増加する．一方，DMTの方は一列中の1を立てるエントリが増えるだけであり，サイズは増加しない.

3.4 CAMを用いた命令ウィンドウによるALU Cascading

CAMを用いた命令ウィンドウでも，以下のようにすればi2とi3，もしくは，i2とi4を同時にreadyにすることができる.

（1） マップ表のprev iwin entryの代わりに，その物理レジスタに値を書き込む命令のソース・タグを書き込んでおく

（2） 通常，命令ウィンドウに命令を登録する時，マップ表から読み出されたソース物理レジスタ番号をソース・タグとしてCAMに登録するが，ALU Cascadingを行う時には1)でマップ表に新たに追加したソースを生成する命令のソース・タグを書き込む

上記の方法では，従来のCAMを用いた命令ウィンドウではCAMは2つ必要になるが，ALU Cascadingを行う場合は4つ必要となる．この構成でもクロック・サイクル時間に悪影響を与えずにALU Cascadingは行えるが，命令ウィンドウ用のCAMの追加は，回路面積や消費電力の点から望ましくない．なお，この構成では，ALU Cascadingの段数を増やすごとに，CAMの数が指数関数的に増加してゆく点も問題である.

DMTでは，従来の命令ウィンドウの2つのCAMを1つの行列にまとめれるのと同様に，上記の4つのCAMを1つの行列表にまとめることができるため．この問題によるコストの増加はない.

4. 評　　価

4.1 評価環境

ALU Cascadingの評価には，SimpleScalar Tool Set [4]を用い，その中に含まれるout-of-order実行シミュレータにALU Cascadingを実装した．シミュレーションで仮定するプロセッサの仕様を表1に示す.

表1　プロセッサの仕様

命令発行幅	8命令
命令ウィンドウ	128エントリ
LSQ	64エントリ
int ALU	8
fp ALU	4
int mul/div	8
fp mul/div	4
メモリ・ポート	8
分岐予測機構	8K-entry g-share, 6-bit history 2K-entry BTB, 16-entry RAS
1次命令キャッシュ	64KB / 32B-line / 2-way
1次データ・キャッシュ	64KB / 32B-line / 2-way
2次キャッシュ	命令データ混在 / 2MB / 64B-line / 4-way

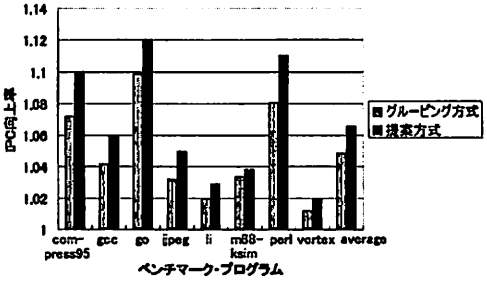

図6　ALU CascadingによるIPC向上率

プロセッサのパイプラインは10段であると仮定し，2段のALU Cascadingを適用したときのIPCの値を測定した．今回のシミュレーションでは，ALU Cascadingの対象を整数加減算，シフト，論理演算といった，実行にかかる時間の短い単純なALU演算命令に限定した．また，先行するALU演算命令に依存するALU演算命令が複数ある場合は，1対多のALU Cascadingをサポートしている．さらに，比較のため，ALU Cascadingを無効にしたときのIPCの値や，同時にデコードされたALU演算命令に対して1対1のALU Cascadingを行う命令グルーピング方式[3]のIPCの値も測定し，提案する命令スケジューラによるALU Cascadingの効果を評価した．ベンチマーク・プログラムは，SPECint95の8本を用いた.

4.2 評価結果

図6に，ALU Cascadingしない場合を1としたときの，ALU CascadingによるIPC向上率を示す．グラフ横軸はベンチマーク・プログラム，縦軸はIPC向上率を表す．各ベンチマークに対する棒グラフは左から，命令グルーピング方式でALU Cascadingを適用した場合，提案方式でALU Cascadingを適用した場合となっている．いずれのベンチマークでも，ALU Cascadingを適用することによりIPCの向上が見られた．特にgoにおいてIPC向上が顕著であり，命令グルーピング方式で9.9%，提案方式で12.0%IPCが向上した．一方，vortexのようにIPCがほとんど向上しない場合もあった．これは，演算命令間のデータ依存がほどんどなかったためと考えられる．平

均IPC向上率は命令グルーピング方式で4.9%，提案方式で6.6%となった．クロック・サイクル時間の後半に実行されたALU演算命令の全ALU演算命令に対する割合は，命令グルーピング方式で平均18.9%，提案方式で平均34.1%であった．

5. 関連文献

あるALUの出力を別のALUの出力に接続し，1クロック・サイクル中に複数の演算を行うことを提案している論文は多数ある．古くは，ベクトル・プロセッサの性能向上のために提案されている[6]．また，時代の流れに伴って，当該手法の適用対象も変化し，マルチメディア処理への適用[7]，GALSプロセッサへの適用[3]など提案されている．

しかし，このような演算を行わせる際に，out-of-order実行を行うスーパスカラ・プロセッサの動的命令スケジューリングを考慮したものは少ない．参考文献[8]では，スーパスカラ・プロセッサにおいて，ALU Cascadingと同様に，1サイクル中に複数のデータ依存関係にある命令を実行するCHAINという手法を提案している．しかし，そのアルゴリズムは命令列に対する複数回のスキャンやオペランドの出現回数の計数が含まれており，クロック・サイクル時間に影響を与えずにハードウェア化をすることが難しいと考えられる．それに対し，我々はクロック・サイクル時間に影響を与えないように配慮しつつ，ハードウェアの構成まで検討を行った．また，参考文献[9]では，同時にデコードされた命令に対してALU Cascadingで実行できる組を探し，それを1組の命令として，命令ウィンドウの1エントリに登録する手法が提案されている．この方法には，命令ウィンドウのサイズを増加させることができるという利点もある．しかし，ALU Cascading可能な命令が同時にデコードされた命令に限定される点や，組となった命令を同時に発行しなくてはならない制限がある．これに対し，我々は上記の制限はなく，1対多のALU Cascading可能という利点もある．

6. まとめ

本論文では，ALU Cascadingをスーパスカラ・プロセッサに実装する時に必要となる，任意の命令の組に対するALU Cascadingを行える命令スケジューラを提案した．提案する命令スケジューラをSPECint95を用いて評価した結果，2段のALU Cascadingを行うと，IPCが平均で6.6%向上するという結果になった．

今後の研究においては，命令スケジューラの変更による負の面について厳密な評価を行い，スケジューラの改良を行う予定である．得に，近年におけるALU Cascadingの研究は，消費電力を削減する研究に付随して使用されるものが多いため，まず，消費電力の増加について厳密な評価を行う予定である．

謝辞　研究の一部は日本学術振興会科学研究費補助金基盤研究S(課題番号16100001)による．

文　献

[1] 嶋田創, 安藤秀樹, 島田俊夫, "パイプラインステージ統合によるプロセッサの消費エネルギーの削減," 情報処理学会論文誌, コンピューティングシステム, Vol. 45, No. SIG 1 (ACS 4), pp.18-30, 2004年1月.

[2] 嶋田創, 安藤秀樹, 島田俊夫, "パイプラインステージ統合とダイナミック・ボルテージ・スケーリングを併用したハイブリッド消費電力削減機構," SACSIS 2004, pp.11-18, 2004年5月.

[3] 佐々木広, 近藤正章, 中村宏, "GALS型プロセッサにおける動的命令カスケーディング," IPSJ, 2005-ARC-164, pp.67-72. 2005年8月.

[4] D. Burger and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0," Technical Report CS-TR-97-1342, University of Wisconsin-Madison Computer Sciences Dept., July 1997.

[5] 五島正裕, 西野賢悟, 小西将人, 中島康彦, 森眞一郎, 北村俊明, 富田眞治, "行列に基づくOut-of-Orderスケジューリング方式の評価," 情報処理学会論文誌, ハイパフォーマンスコンピューティングシステム, Vol.43, No.SIG 6(HPS5), pp.13-23 2002年6月.

[6] 長島重夫, 稲上泰弘, 阿部仁, 河辺峻, "動的チェイニングによるベクトルプロセッサの実効性能の向上," 電子情報通信学会論文誌 D, Vol.J74-D1, No.12, pp. 836-845, 1991年12月.

[7] 山崎信行, 伊藤務, 内山真郷, 安西祐一郎, "柔軟なマルチメディア処理機構を有したリアルタイムプロセッサアーキテクチャ," 日本機械学会ロボティクスメカトロニクス講演会'01, pp. 1-2, 2001年6月.

[8] 孟林, 小柳滋, "スーパースカラプロセッサにおける動的RENAME手法とCHAIN手法," 平成18年度情報処理学会関西支部支部大会講演論文集, pp. 207-210, 2006年10月.

[9] 佐々木宏, 近藤正章, 中村宏, "命令グルーピングによる効率的な命令実行方式," IPSJ, 2006-ARC-170, pp. 73-78, 2006年11月.

[10] 尾形幸亮, 嶋田創, 中島康彦, 森眞一郎, 富田眞治, "パイプラインステージ統合におけるALU Inlining," 平成18年度情報処理学会関西支部支部大会講演論文集, pp. 203-206, 2006年10月.