命令カスケーディングを利用する耐ばらつきアーキテクチャにおける命令の重要度

渡辺, 慎吾
九州工業大学大学院情報工学研究科情報科学専攻

Sato, Toshinori
System LSI Research Center, Kyushu University

出版情報：情報処理学会九州支部若手の会セミナー講演論文集, pp.21-26, 2007-09. 情報処理学会九州支部
バージョン：

命令カスケーディングを利用する 耐ばらつきアーキテクチャにおける命令の重要度

渡辺　慎吾 †　　佐藤　寿倫 ‡

† 九州工業大学大学院 情報工学研究科 情報科学専攻
‡ 九州大学 システムＬＳＩ研究センター

近年，LSI の微細化が進み製造ばらつきが深刻になっている．特にチップ内で発生するトランジスタ特性のランダムなばらつきが問題である．トランジスタ特性のばらつきは回路遅延にばらつきを生じさせる．本研究では，回路遅延の統計的性質に着目し，命令カスケーディングを利用することで回路遅延ばらつきを縮小することを検討している．しかし，ばらつきに配慮するために，プロセッサの性能を大きく低下させることは許容できない．本稿では命令カスケーディングおける命令の重要度に着目し，性能維持の可能性についての調査を行う．

Exploiting Instruction Criticality on Variation Resilient Microarchitecture Utilizing Instruction Cascading

SHINGO WATANABE†　　TOSHINORI SATO‡

† Department of Artificial Intelligence, Kyushu Institute of Technology
‡ System LSI Research Center, Kyushu University

Due to the aggressively advanced semiconductor technologies, a problem of parameter variations is emerging. Especially, with-in-die variations in transistor performance are very serious. Parameter variations in transistors affect circuit delay. In this paper, we investigate statistical characteristics of circuit delay, and propose to utilize instruction cascading for variation resiliency. An important thing to consider is that performance degradation cannot be accepted even for reduction in variations. In this paper, we exploit instruction criticality to maintain performance of the processor that utilizes instruction cascading.

1. はじめに

LSI の微細化はムーアの法則として知られるように利用可能なトランジスタ数を飛躍的に増加させ，これまでプロセッサの性能向上に大きく貢献してきた．しかし，一層の微細化により，性能ばらつきの増加という深刻な問題が顕在化している．LSI におけるばらつきは，チップ間ばらつき (die-to-die:D2D) とチップ内ばらつき (within-die:WID) に分類される．近年問題視されているのは後者であり，特にランダムなばらつきが深刻である．WID ばらつきの原因は以下の 4 点である．①不純物密度の不均質な分布，②リソグラフィーにおけるレイアウトパタンの再現性低下，③電力消費の不均一さに起因する電源のゆらぎ，④温度分布の不均一さ，である．これらはトランジスタの速度とリーク電流に影響を及ぼす．①と②は LSI の製造工程上避けがたいばらつきであり，その結果，閾値電圧つまりトランジスタ特性にランダムなばらつきが生じる．トランジスタ特性のばらつきは回路遅延にもランダムなばらつきを与える．これまでは回路遅延の最悪値を考慮して，ばらつきが発生しても正しく動作するよう設計していた．しかし，最悪値を用いたのでは，動作周波数や消費電力など求められる仕様を満たすことは困難になっている．本研究では，生じてしまうトランジスタ特性のばらつきを考慮し，根本的にばらつきを縮小できるマイクロアーキテクチャを検討する．

回路遅延の統計的性質に関して，以下のことが示されている[3]．最長パス数が多くなるとばらつきは縮小し，遅延は増大する．論理段数が大きくなるとばらつきは縮小し，かつ遅延も小さくなる．これらの性質をプロセッサにおいて利用する方法として，パイプライン段数を少なくし各ステージの論理段数を大きくすることでばらつきを縮小することが考えられる[8]．しかしながら，パイプライン段数の削減はプロセッサの動作周波数を低下させ，性能を悪化させる．性能を維持するにはクロックサイクル当たりの命令実行数を増加させる必要がある．文献 8) では，プロセッサのバックエンドに対して命令カスケーディング[4)6)9)] を利用することが提案されている．命令カスケーディングでは依存関係のある複数の命令をグループ化し，同一サイクルで実行する．そのため，クロックサイクル当たりの命令実行数を改善可能であるが，文献 8) ではグ

ループ化できる命令の割合は 7 割程度であり命令カスケーディングだけではスループットの維持は困難であると結ばれている．

しかし，グループ化できる命令とできない命令の重要度に着目すると，グループ化できない命令はプロセッサの性能に影響を与えないと考えられる．グループ化できる命令は互いに依存関係があり，それらを結ぶとクリティカルパスになると考えられる．クリティカルパスとはプログラムの実行時間を決定する命令列である[5]．一方，グループ化できない命令はプロセッサ内の他の命令と依存関係が存在しない．そのためクリティカルパス上の命令ではなく，プログラムの実行時間に影響を与えない重要度の低い命令である．本稿では命令の重要度に着目して，命令カスケーディングのスループット維持可能性について調査する．

本稿の構成は以下のとおりである．次節で回路における遅延の統計的性質について説明する．3 節では検討しているばらつきを考慮したマイクロアーキテクチャを紹介する．4 節で命令カスケーディングとプロセッサの性能維持の可能性について述べる．5 節でプロセッサの性能維持可能性の調査を行う．6 節で考察し，今後の課題について述べる．7 節でまとめとする．

2. 回路遅延の統計的性質

本節では回路遅延の統計的性質を説明する．

図 1 は最長遅延パスの総数が回路遅延に及ぼす影響を示している[3]．横軸は遅延を，縦軸はその遅延を取りうる確率を示している．各パス遅延は互いに無相関で，平均が 6 で分散が 1 である正規分布 $N\left(6,\ 1^2\right)$ に従うものと仮定されている．図中の n は最長遅延パスの本数であり，$n\ =\ 1$ の場合が $N\left(6,\ 1^2\right)$ の正規分布である．最長遅延パスの本数が増加するにしたがって，分布は回路遅延が増大する方向へ移動していることがわかる．これは回路中の全てのパスの中で最も遅延の大きなパスが，その回路の遅延時間を決定するためである．この性質から，遅延の揃ったパスを多く持つような回路ほど，ばらつきによる回路遅延増の影響を受けやすいことがわかる．一方で遅延のばらつきの度合いは，最長パス数が増加するにしたがって小さくなっている．このことは，遅延自体は増加する傾向にあるものの，ばらつきは小さくなることを示している．

図 2 は最長パスの論理段数が回路遅延に及ぼす影響を示している[3]．図 1 と同様に横軸は遅延を，縦軸はその遅延を取りうる確率を示している．パス遅延が互いに無相関だと仮定しているので，論理段数 m が大きくなるにしたがって標準偏差 σ は $1/\sqrt{m}$ で小さくなる．そこで，論理段数の変化を標準偏差で表したものが同図である．パス数の総数 n は 100 本である．標準偏差 σ が大きいほど論理段数 m が小さいことに相当する．容易に想像されることではあるが，論理段数が小さくなるにしたがって，分布は回路遅延が大きくなる方向に移動し，同時にばらつきも大きくなっている．これは論理段数が浅い高速な回路ほど，ばらつきの影響を大きく被ることを示している．

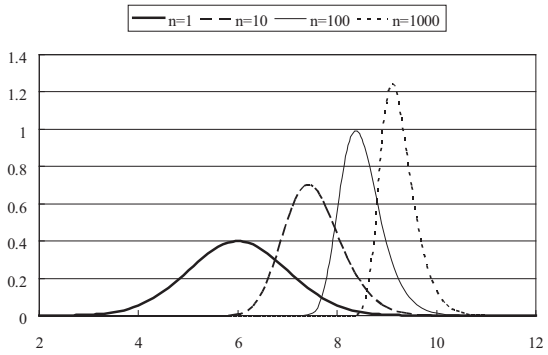

図 1 最長遅延パス数の影響

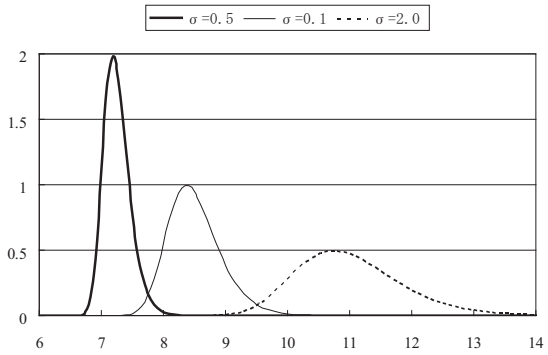

図 2 論理段数の影響

3. 耐ばらつきアーキテクチャ

3.1 パイプラインステージの統合と削減

前節の回路遅延の統計的性質，特に論理段数の影響に着目すると，これまでのプロセッサのように深いパイプライン構成をとると各パイプラインステージの論理段数が小さくなり，回路遅延ばらつきを増大させる．本研究では，逆にパイプラインステージを統合し，各ステージの論理段数を大きくすることを検討している．2 ステージ分を 1 ステージに統合することを想定している．パイプライン段数の削減はプロセッサの動作周波数を低下させる．理想的にはパイプライン段数と動作周波数は比例の関係にある．パイプライン段数が 1/2 になれば動作周波数も 1/2 になる．動作周波数の低下はプロセッサのスループットを悪化させる．プロセッサのスループットはクロックサイクル当たりの命令実行数と動作周波数の積で表わされる．スループットを維持するにはクロックサイクル当たりの命令実行数を 2 倍に改善する必要がある．スループットを維持する方法も検討する必要がある．

パイプラインステージを統合した場合の遅延への影響は次のようになる．それぞれのステージを隣のステー

ジと接続するため，各ステージの論理段数は 2 倍になる．そのため，平均は 2 倍，標準偏差は $\sqrt{2}$ 倍となるので，分布は $N\left(6*2,\ \sqrt{2}^2\right)$ となる．遅延の平均は 2 倍になるがスループットを維持できれば命令当たりの遅延は半分とみなすことができる．さらに，回路のパス数は 1/2 となる．これは互いのステージのパスを接続することで，1 本のパスになるためである．パス数の減少は遅延を小さくするが，ばらつきを大きくする．図 3 に回路のパス数が 100 のとき，パイプライン段数が 1/2 になった場合の命令当たりの遅延変化を示す．実線が統合前である．点線が統合後であるが，1 ステージ中の命令数が 2 倍になることを考慮して，1 命令当たりの遅延を示している．同図からわかるように遅延のばらつきは縮小している．パス数の減少によるばらつきの増大より，論理段数の増加によるばらつきの縮小の効果が大きいことがわかる．パイプラインステージの統合は有効な方法である．興味深いのは，統合後の遅延の平均の方が統合前より小さくなっている点である．2 倍になった遅延を半分にしているので，統合前と統合後の遅延は一致するはずである．この理由は上述のように最長パス数の減少のため，回路遅延そのものが縮小しているからである．

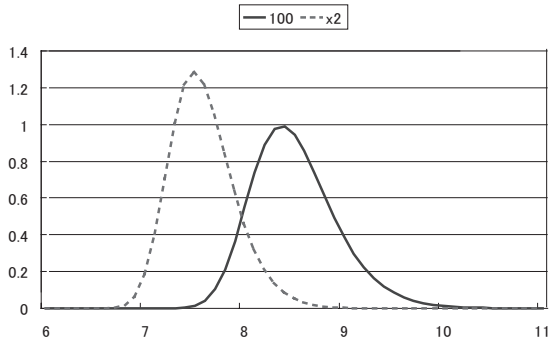

図 3 パイプライン段数削減の遅延への影響

3.2 命令カスケーディング

本稿ではプロセッサのバックエンド，演算ステージの論理段数を増加させる手法について検討する．上述のようにスループットを維持しつつ，論理段数を増加させる必要がある．本研究では命令カスケーディング[4)6)9)]を利用することを検討している．命令カスケーディングとは，互いに依存関係のある命令をグループ化し，カスケード接続した演算器を用いて同一サイクル内で連続実行する手法である．命令間の依存とは図 4 のように先行命令 I1 の結果を後続命令 I2 が使用する場合に発生する．図 4(a) のように従来のプロセッサでは，依存関係のある命令を同一サイクルで実行することはできない．I1 が実行を完了しなければ I2 は実行できない．命令カスケーディングでは図 4(b) のように，一方の演算器の出力を他方の演算器の入力に接続し，依存関係のある命令を 1 つにグループ化し連続的に実行する．なお，本稿では命令カスケーディングのための命令のグループ化を単にグループ化と呼ぶ．

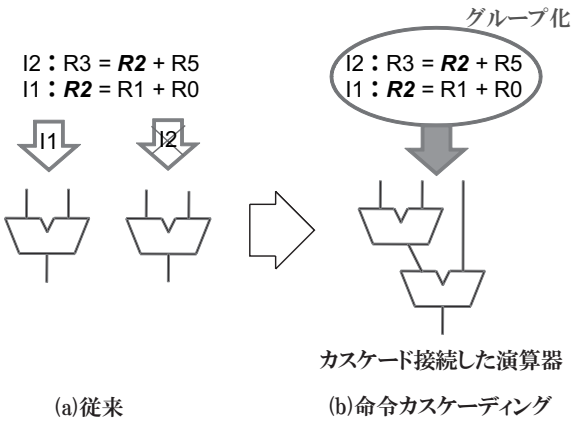

図 4 命令カスケーディング

命令カスケーディングでは，図 4(b) のようなカスケード接続した演算器を用いる．カスケード接続することによって演算器の論理段数が増加し，その結果，演算ステージの回路遅延ばらつきを縮小することができる．回路遅延は増加するが，周波数を 1/2 にすれば 1 クロックサイクル内で終了する．前節で述べたようにパイプラインステージの統合で動作周波数は低下しているので問題にならない．また，依存関係のある命令を同一サイクルで実行できるのでクロックサイクル当たりの命令実行数は増加する．動作周波数低下によるスループットの減少を，命令カスケーディングによるクロックサイクル当たりの命令実行数の増加で相殺することができれば，演算ステージ全体のスループットは維持できる．

命令カスケーディングでは動作周波数を 1/2 にするので，クロックサイクル当たりの命令実行数を 2 倍にしなければスループットを維持できない．つまり，全ての命令をグループ化することが出来ればスループットを維持できる．しかし，先行調査の結果，動的な命令カスケーディングでは約 7 割の命令しかグループ化できないことが判明した[8)]．残り 3 割の命令はグループ化できないことになる．グループ化できれば 1 クロックサイクルで 2 命令実行できるが，グループ化できなければ 1 命令しか実行できない．そのため，上述のようにクロックサイクル当たりの命令実行数を 2 倍にすることはできない．しかしながら，グループ化できない命令の重要度を考慮すると，スループットを維持できる可能性がある．次節で詳しく述べる．

4. 命令の重要度

プログラムの実行時間を決定する命令列をクリティカルパスと呼ぶ[5)]．クリティカルパスは命令間の依存関係を結んだ鎖のうち最長のものを結んだ実行パスで

ある．図 5 は命令間の依存関係を表すデータフローグラフの例である．各ノードは命令を，各枝は命令間の依存関係を表している．また，数字の小さい命令ほど古いことを示している．同図において最も長いパスである命令 I1->I2->I4->I5->I7->I9 を結んだパスがクリティカルパスとなる．クリティカルパス上の命令は依存関係があるので逐次的にしか実行できない．クリティカルパスの長さがプログラムの実行時間を決定する．つまり，クリティカルパス上の命令はプログラムの実行時間に影響を与える重要な命令である．逆に，クリティカルパス上ではない命令は重要度の低い命令である．

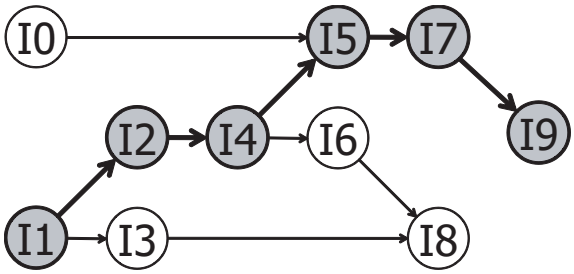

図 5 クリティカルパス

図 6 を用いて命令カスケーディング時の命令の振舞を説明する．まず，time #0 のとき，命令ウィンドウ内には命令 I0〜I3 までの命令が存在する．命令 I1 と I2，I3 間には依存関係が存在するので，グループ化可能である．命令の出現順にグループ化することに決めると，命令 I1 と I2 がグループ化される．命令 I0 と命令グループ (I1，I2) は実行可能であるので演算器に発行される．次のサイクル time #1 では，命令 I0，I1，I2 が命令ウィンドウからなくなり，命令 I4，I5，I6 が入ってくる．このとき命令 I4 と I5 がグループ化される．命令 I3 と命令グループ (I4，I5) は実行可能なので発行される．同様に次のサイクル time #2 では命令 I6 と I8，命令 I7 と I9 がグループ化される．

この振舞いで着目する点は，クリティカルパス上の命令はすべてグループ化され，グループ化されない命令はクリティカルパス上にはないことである．仮に，time #0 で命令 I1 と I3 がグループ化されても，次のサイクルで命令 I2 は I4 とグループ化される．クリティカルパス上の命令は互いに依存関係があり，しかも近い距離にあるのでグループ化されることになる．また，クリティカルパス上にない命令がグループ化されても，グループ化される命令が増え性能向上に寄与するので，むしろ好ましい．

以上のことから，グループ化できない命令はクリティカルパス上の命令ではなく，プログラムの実行時間に影響を与えないと言える．つまり，グループ化ができない命令が存在してもスループットを維持できる可能性がある．

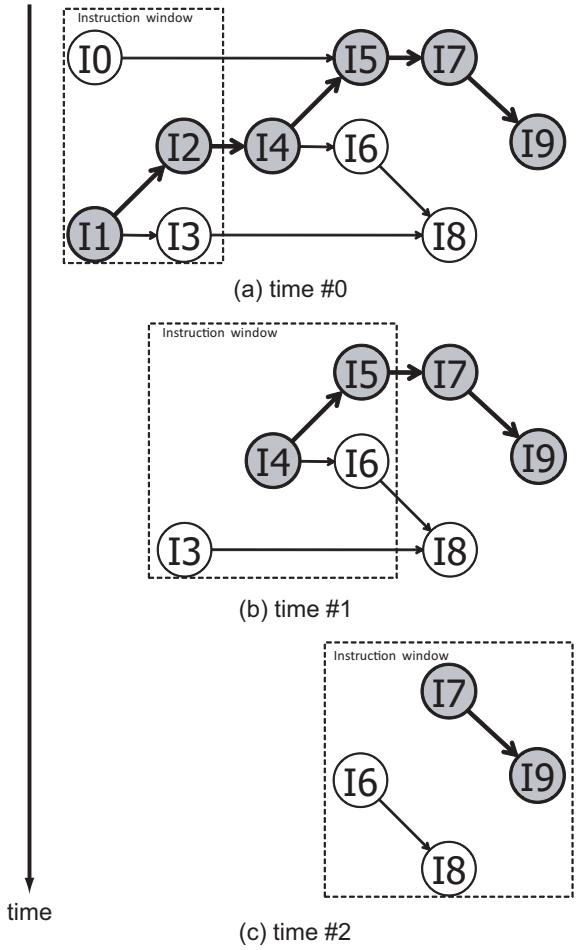

図 6 命令カスケーディング時の命令の振舞い

5. 予備調査

上述した，グループ化されない命令は性能に影響しないという予想に関して，本節で予備調査を行う．調査にはプロセッサシミュレータを用いる．本シミュレータにはカスケーディングを行う機能は実装されていない．以下に説明するように命令の実行レイテンシを変えることで，命令の実行時間への影響を調査する．

命令カスケーディング時の命令実行の振舞に着目すると，グループ化された命令は 1 クロックサイクルで 2 命令実行される．グループ化されない命令は 1 クロックサイクルで 1 命令実行される．そのため，グループ化された命令に対してグループ化されない命令は実行レイテンシが伸びたように見える．この点を利用して，グループ化できる命令は 1 サイクルで実行する．一方，グループ化できない命令は実行サイクル数を増加させる．このようにすることで，命令カスケーディングを施したときの実行時間への影響を見積もることができる．

グループ化できる命令の条件は以下のとおりである．

- 対象は整数 ALU で実行される命令とする．乗除算，ロード／ストアの各命令は対象外とする．だた

し，ロード／ストア命令のアドレス演算はグループ化の対象とする．

- 命令の発行時にスケジューリングウィンドウ内を探索し，依存関係のある整数命令をグループ化する．
- 複数の命令とはグループ化しない．3 命令以上の依存関係は 2 命令ずつに分けてグループ化する．また，1 命令に対して複数依存する命令があっても，その中の 1 命令だけを選んでグループ化する．

なお，実装については考慮しておらず，今後の検討を要する．

5.1 調査環境

調査にはプロセッサシミュレータである SimpleScalar ツールセット[1] を用いる．命令セットは Alpha を用いる．シミュレータに，命令をグループ化可能か否か判定する機能と，グループ化できない命令の実行サイクル数を増加させる機能を追加する．グループ化対象外の命令の実行サイクル数は変化させない．その他のプロセッサの構成を表 1 に示す．

表 1 プロセッサの構成

Fetch width	4 instructions
L1 I cache	64 KB
Branch prediction	bimodal and gshare
Dispatch width	4 instructions
RUU size	128 entries
Issue width	4 instructions
Integer ALUs	4 units, 1 cycle
Integer MULT/DIV	1 unit, MULT : 3 cycles DIV : 20 cycles
Floating ALUs	4 units, 2 cycles
Floating MULT/DIV	1 unit, MULT : 4 cycles DIV : 12 cycles SQRT : 24 cycles
L1 D cache	64 KB
Unified L2 cache	2048 KB
Commit width	4 instructions

ベンチマークプログラムは SPEC2000 CINT を用い，初めの 5 億命令をスキップ後, 1 億命令をシミュレーションする．

5.2 調査結果

図 7 に調査結果を示す．縦軸は実行時間を表し，基準となるプロセッサモデルの値を 1 として正規化を施している．base が評価基準となるプロセッサで，整数命令の実行サイクル数は全て 1 である．+1 はグループ化できない命令の実行サイクル数を 1 増加させたときの実行時間である．同様に 2 サイクル，3 サイクル増加させた場合が+2，+3 である．3 サイクル増加した場合でも，実行時間は平均で 3%の増加であった．最も増加した gzip でも 14%であった．

本研究で検討しているのは 2 命令のグループ化である．2 命令のグループ化ではグループ化されない命令の実行サイクル数は 1 サイクル増加する．つまり，本調査では+1 である．+1 では，プログラムの実行時間は平均で 1%増加している．最も悪化した gzip でも 6%の増加である．これらの結果から，グループ化されない命令が実行時間に与える影響はごく僅かであることが言える．

図 8 に実行された命令の内訳を示す．cascadable がグループ化できる命令の割合を表す．uncascadable は整数命令であるがグループ化できない命令を示す．others はグループ化対象外の命令である．それぞれの命令の割合は，実行サイクル数を増加させても僅しか変化しなかった．同図では実行サイクル数を 1 増加させた場合のみを示している．内訳は，グループ化できる命令は平均で 27%，グループ化できない命令は平均 43%であった．整数命令の内約 4 割がグループ化できる．最もグループ化できる割合が少なかった eon では 14%の命令しかグループ化できていない．しかし，eon は他のプログラムと比較して，実行時間の増加の割合が少ないプログラムである．また，実行時間が最も増加した gzip は，グループ化できる命令の割合が多いプログラムの一つである．このことから，グループ化できる命令の割合とプログラムの実行時間は直接関係しないことがわかる．

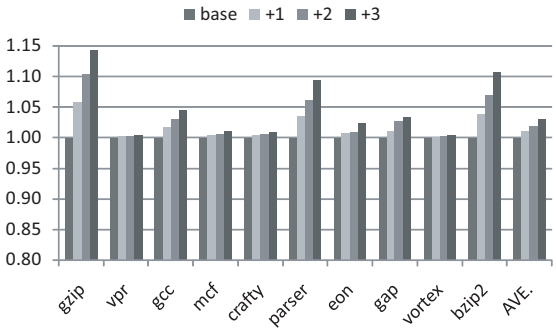

図 7 プログラムの実行時間への影響

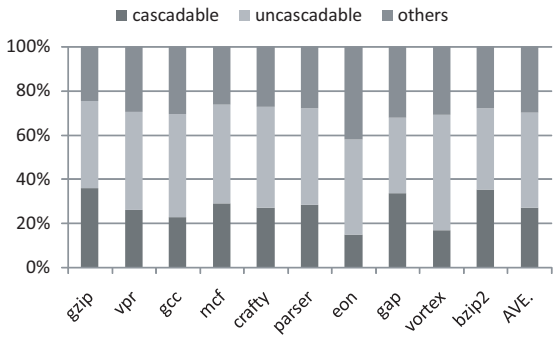

図 8 実行命令の内訳

6. 考察，および今後の課題

調査の結果，命令カスケーディングにおいてグループ化できない命令はプログラムの実行時間に影響を与えないことがわかった．グループ化できる命令とできな

い命令では，命令の重要度に違いがあり命令カスケーディングはこれを適切に利用できているためである．また，グループ化できる命令は全実行命令の27%であり，全整数命令の 4 割程度しかグループ化できていないことも分かった．このことから，グループ化できない命令とできる命令が同程度の割合で実行されることが適切である．どちらか一方の命令実行が滞るとそれがボトルネックとなり，全体のスループットに悪影響を与える．これらを考慮すると図 9 のような整数演算器の構成が適切であると考えられる．このとき，1 段の演算器では，論理段数が小さいままなので遅延ばらつきは大きい．しかし，カスケード接続した演算器と同じ周波数で動作すれば良いので，ばらつきに対して十分なマージンを取ることができる．遅延ばらつきが大きなままでも問題ない．

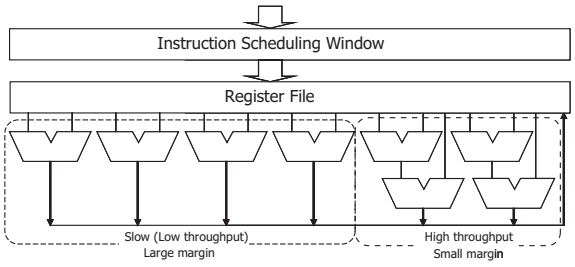

図 9 演算器の構成

検討手法では，動作周波数が低下するためクロックサイクル当たりの命令実行数を改善する必要がある．つまり，演算器数を 2 倍にし，1 クロックサイクルで同時に実行できる命令数を増やす必要がある．しかし，同時実行数の増加は，命令スケジューリング機構，レジスタファイル，および演算結果のフォワーディング機構を複雑にする．これらの機構では既に消費電力や動作速度，回路面積などの問題が発生している[2)7)10)]．そのため同時実行数を安易に増加することはできない．今後はこれらの問題の対策を合わせて検討する必要がある．

7. ま と め

トランジスタの特性ばらつきから生じる回路遅延のばらつきを縮小する方法を検討している．回路遅延の統計的性質に着目し，命令カスケーディングを利用することで演算器のばらつきを縮小できる．本稿では，命令の重要度の違いに着目し，命令カスケーディングにおける性能維持の可能性について調査を行った．調査の結果，グループ化されない命令が存在しても，性能に影響を与えないことが確認できた．今後は命令のグループ化を行う機構とともに，命令発行幅増加の対策を考案する必要がある．

謝　　辞

本研究の一部は，科学研究費補助金・基盤 A(No. 19200004)，および科学技術振興機構・CREST プロジェクトの支援によるものである．

参 考 文 献

1) T. Austin, E. Larson, and D. Ernst, SimpleScalar: an infrastructure for computer system modeling, IEEE Computer, Vol.35, No.2 (2002)
2) 内田, 三谷, M. H. Jurgen, 小出, 弘中, マルチバンク構造による小面積多ポートレジスタファイル, 信学技報, Vol.102, No.479(2002)
3) 小野寺, ばらつきを克服する設計技術, 回路とシステム軽井沢ワークショップ (2006)
4) 小沢, 中村, 南谷, Cascade ALU を用いた命令実行手法の提案と評価, 情処研報, 99-ARC-13(1999)
5) 小林, 安藤, 島田, データフロー・グラフの最長パスに着目したクラスタ化スーパースカラ・プロセッサにおける命令発行機構, 並列処理シンポジウム (2001)
6) 佐々木, 近藤, 中村, GALS 型プロセッサにおける動的命令カスケーディング, 情処研報, 2005-ARC-164(2005)
7) 佐々木, 近藤, 中村, 依存情報を用いた命令グループ化による動的命令スケジューリング機構の電力削減手法, 情処研報, 2006-ARC-168(2006)
8) 佐藤, 命令レベル逐次プロセッサ, 情処研報, 2006-ARC-169(2006)
9) 佐藤, 千代延, 命令カスケーディングにおける典型的パス遅延の効用, 信学技報, CPSY2005-36(2005)
10) 三輪, 一林, 入江, 五島, 富田, 小容量 RAM を用いたオペランド・バイパスの複雑さの低減手法, 第 5 回先進的計算基盤システムシンポジウム (2007)