タイミング歩留まり改善を目的とする演算器カスケーディング

渡辺, 慎吾
九州工業大学

橋本, 昌宜
大阪大学

佐藤, 寿倫
福岡大学 | 九州大学 | 独立行政法人科学技術振興機構

出版情報：第6回先進的計算基盤システムシンポジウム, SACSIS 2008, pp.115-122, 2008-06-11.
Symposium on Advanced Computing Systems and Infrastructures
バージョン：
権利関係：

タイミング歩留まり改善を目的とする演算器カスケーディング

渡辺慎吾[†1] 橋本昌宜[†2] 佐藤寿倫[†3]

半導体製造プロセスの微細化が進展するにつれ，製造ばらつきの拡大という深刻な問題が顕在化している．それによりトランジスタの特性ばらつきが増大し，タイミング歩留まりの悪化が懸念されている．我々は回路遅延の統計的性質に着目し，演算をカスケーディング実行することで演算器の遅延ばらつきを縮小することを検討している．本稿では，演算器の統計的遅延解析とプロセッサ性能の評価とから，演算カスケーディングのタイミング歩留まり改善に対する効果を調査する．その結果，ばらつき問題への対策にはマイクロアーキテクチャの大局的な検討が必要であるという知見が得られた．

ALU Cascading for Improving Timing Yield

Shingo Watanabe, Masanori Hashimoto and Toshinori Sato

As semiconductor technologies are aggressively advanced, the problem of parameter variations is emerging. Parameter variations in transistors affect circuit delay, resulting in serious yield loss. We exploit the statistical characteristics in circuit delay, and investigate a cascading technique of ALU operations for variation reduction. From the statistical timing analysis in circuit level and the performance evaluation in processor level, this paper tries to unveil how efficiently the cascading technique improves timing yield of processors. We find that innovations are required for managing parameter variations in microarchitecture level.

1. はじめに

ムーアの法則として知られるように，半導体の微細化は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性能の向上に大きく貢献してきた．しかし，一層の微細化によりトランジスタ特性のばらつきが拡大するという深刻な問題が顕在化している．ばらつきは動的なばらつきと静的なばらつきに分類される[4]．動的なばらつきは，トランジスタ周辺の温度や供給電圧など周辺環境の揺らぎを要因とするばらつきである．静的なばらつきは製造ばらつきと呼ばれ，製造時にトランジスタのゲート形状や不純物濃度などに生じる微細な揺らぎを要因とする．製造時の揺らぎは本質的に避けがたく，トランジスタサイズの縮小に伴い拡大している[3]．製造ばらつきはチップ間ばらつきとチップ内ばらつきに分類される．チップ間ばらつきは各チップの平均値が変動するばらつきであり，チップ内ばらつきはチップ内において各トランジスタの特性などが変動するばらつきである．微細化の進展にともないチップ内ばらつきが深刻化している[15]．

トランジスタの特性ばらつきにより回路遅延が変動する[3]と，製造されたチップの中にはタイミング制約を満たせないものが現れる．今後製造ばらつきが拡大すると，タイミング制約を満たせないチップの割合が増加し，タイミング歩留まりが悪化することが予想される．本研究では製造ばらつきに起因するタイミング歩留まりを改善することを目標としている．

回路遅延の統計的性質に関し，以下のことが示されている[5)7]．回路中のクリティカルパス数が多くなると遅延は増加するが，ばらつきは相対的に縮小する．論理段数が大きくなると遅延は増加するが，ばらつきは相対的に縮小する．これらの性質をマイクロアーキテクチャにおいて利用する方法として，演算をカスケーディング実行することを提案する．直列接続された演算器上で，依存関係にある複数命令を同一サイクルで連続的に実行する方法である[10)13]．カスケーディング演算するための演算器は論理段数が大きくなるため，遅延ばらつきの縮小が期待できる．また，カスケーディング演算を行う演算器の遅延は増加するが，同一サイクルで複数命令を実行できるためサイクル当たりの実行命令数を改善可能である．本稿では，カスケーディング演算を行う演算器の統計的静的遅延解析を行い，それを採用するプロセッサの性能を考慮して，そのタイミング歩留まりの改善度合いを評価する．

2. 関連研究

特性ばらつきを意識して歩留まりを改善しようという試みが現れ始めているが，回路あるいはマイクロアーキテクチャのレベルでばらつきそのものを縮小しよう

†1 九州工業大学
†2 大阪大学
†3 福岡大学, 九州大学, JST CREST

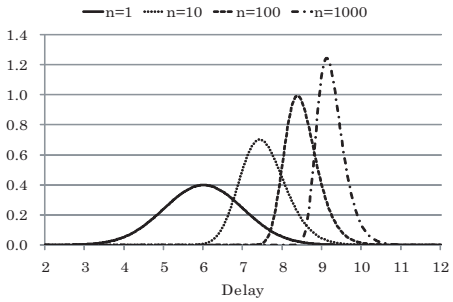

図 1 クリティカルパス数の影響

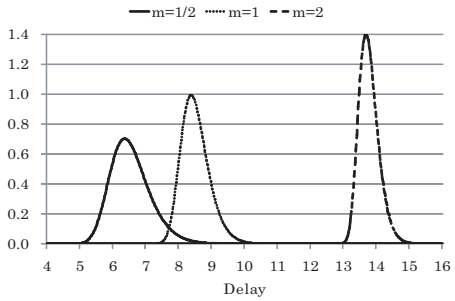

図 2 論理段数の影響

とする試みはほとんどなされていない．可変レイテンシ加算器[9]，可変レイテンシ FPU[8]，可変レイテンシレジスタファイル[8]，そして ReCycle[12] 等は，いずれも製造後に顕在化した遅延増加に対してレイテンシの増加やタイムボローイングを採用するだけで，対処療法に過ぎない．Razor[6] や X-Pipe[14] はばらつき耐性を持つフリップフロップであるが，タイミング違反を検出することしか出来ず，抜本的な対策ではない．対照的に，本稿は回路遅延の統計的性質に着目し，マイクロアーキテクチャのレベルで遅延ばらつきそのものを縮小することを目指している．

複数演算のカスケード実行については多くの先行研究が存在する．中でも，13) は命令レベル並列性を向上させる目的でカスケード実行を検討している先駆的な研究であり，10) は GALS プロセッサで利用している点で特徴的である．われわれは，カスケード演算の新規性を主張するつもりは無い．これまで性能改善や消費電力削減の目的で利用されてきた演算カスケーディングを，遅延ばらつきの縮小という新たな目的のために利用することを提案している．

3. 回路遅延の統計的性質

回路遅延の統計的性質について説明する．図 1 はクリティカルパスの総数が回路遅延に与える影響を示している[5)7)]．横軸は遅延を，縦軸は遅延の出現度を示す．各パスは互いに無相関で，平均 (μ) が 6，標準偏差 (σ) が 1 である正規分布 $N\left(6, 1^2\right)$ に従うものと仮定している．**図 1** の n はクリティカルパスの本数であり，$n = 1$ の場合が $N\left(6, 1^2\right)$ の正規分布である．クリティカルパスの本数が増加するにしたがって，遅延の平均は増大する方向へ移動している．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もつ回路ほど，ばらつきによる回路遅延増大の影響を受けやすいことがわかる．一方で分布の広がり，標準偏差はクリティカルパス数が増加するにしたがって小さくなっている．

図 2 はクリティカルパスの論理段数が回路遅延に与える影響を示してる．図 1 と同様に横軸は遅延を，縦軸は遅延の出現度を示す．図 2 の m は相対的な論理段数を示す．$m = 1$ が図 1 に示すクリティカルパス数が 100 本のときの遅延分布である．$m = 1/2$ は論理段数が 1/2 倍になった場合を，$m = 2$ は論理段数が 2 倍になった場合を示している．論理段数が大きくなるとゲート遅延の変動が平均化されるため，遅延のばらつきは縮小する．論理段数 m が大きくなるに従って標準偏差は $1/\sqrt{m}$ で相対的に小さくなる．このことは論理段数が大きくなると遅延は増加するがばらつきの度合いは縮小することを示している．

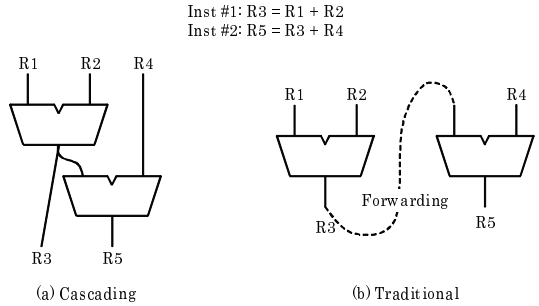

図 3 演算器カスケーディング

4. 演算器カスケーディング

本節で，遅延ばらつきそのものを縮小するマイクロアーキテクチャを提案する．実行ステージに着目し，その遅延ばらつきを改善する．つまり本稿では，実行ステージがプロセッサの動作周波数を決定すると仮定して，以降の議論を進める．実行ステージにおける工夫が他のマイクロアーキテクチャに与える影響を，出来るだけ小さく抑えるという方針で検討する．

回路遅延の統計的性質を活用し遅延ばらつきを縮小することを目的として，演算のカスケーディング実行を提案する．演算カスケーディングでは，**図 3**(a) のように 2 つの演算器を用いて一方の演算器の出力を他方の演算器の入力に接続する．直列接続された演算器を用いて依存関係のある複数命令を 1 サイクル内で連続実行する[10)13)]．カスケーディング演算するための演算器は論理段数が大きくなるため，回路遅延のばらつきを縮小することが可能である．

表 1 合成条件

演算器	32 ビット桁上げ選択加算器
セルライブラリ	日立 0.18μm
動作条件	TYPICAL 出力ポートの付加容量： 30 出力ポートのファンアウト数: 4 入力ポートのドライブ抵抗： 1.5
制約条件	遅延： 最小 動的電力： 最小

従来のスーパスカラプロセッサは，図 3(b) のように依存関係にある命令を並列に処理できない．図の先行命令 1 と後続命令 2 には依存関係があり，命令 1 が演算を終了しないと命令 2 を実行できない．そのため少なくとも 2 サイクルを要する．2 演算をカスケード実行すると，それら 2 命令を 1 サイクルで実行できる．サイクル当たりの実行命令数を増やすことができ，IPC(Instructions per Cycle) を改善できる．

カスケード実行可能な演算器は回路遅延が増加する．遅延増加には利点と欠点がある．利点は，実行ステージ以外のステージにおけるタイミングマージンを大きくすることである．動作周波数が低下することから，実行ステージ以外のステージはタイミング制約が緩和されるわけである．欠点は，その動作周波数低下によるプロセッサ性能低下の可能性である．IPC を改善できたとしても，周波数が大きく低下すればプロセッサ性能は低下する．本稿ではこれらを考慮し評価する．

5. 演算器の統計的遅延解析

5.1 対 象 回 路

対象は 32 ビット桁上げ選択加算器（Carry Select Adder: CSLA）である．まず 2 入力 1 出力の CSLA（2 入力 CSLA と呼ぶ）を Verilog HDL で設計する．その 2 入力 CSLA モジュールをふたつインスタンス宣言し，それらを HDL レベルで直列接続し，3 入力 2 出力の CSLA（3 入力 CSLA と呼ぶ）を作成する．桁上げ保存加算器等を利用する 3 入力 1 出力の複合演算器ではないことに注意されたい．4 節で説明したように，マイクロアーキテクチャの変更を極力抑えるという方針のため，2 演算の結果は両方ともにレジスタファイルに書き込まれる必要があるためである．Synopsys 社の DesignCompiler により，日立 0.18um セルライブラリを用いてゲートレベル・ネットリストを合成する．**表 1** に合成時の条件をまとめる．3 入力 CSLA の合成では遅延の増加が小さくなることを期待して階層化しないでフラットに合成する．

5.2 統計的評価方法

図 4 に示す回路の統計的遅延解析方法を説明する．図中, DesignCompiler のみが市販ツールであり, SDF w/ variations Generator と Statistical Processing Tool は自作ツールである．

① 対象回路を合成しネットリストと Standard Delay Format（SDF）ファイルを得る．

② ① の SDF ファイルを基に，各ゲート遅延がばらついた状態の SDF ファイルを作成する．

③ ② の工程を繰り返し，ばらつきの異なる多数の SDF ファイルを作成する．

④ ① で合成したネットリストと ③ で作成した SDF ファイルを 1 つ用いて，DesignCompiler で対象回路の静的遅延解析を行う．これを全ての SDF ファイルで実施する．

⑤ 全ての遅延解析結果を統計処理する．

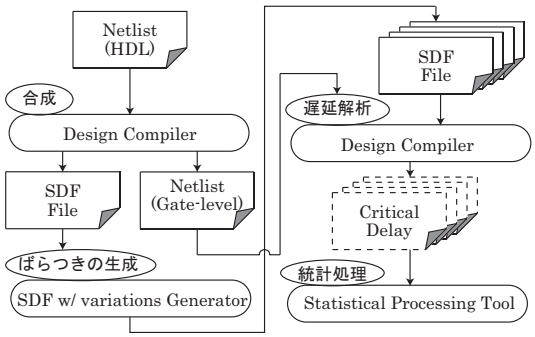

図 4 統計的評価の概要

② でゲート遅延をばらつかせる方法を説明する．SDF ファイルには，論理ゲートの各パスの立ち上り遅延と立ち下り遅延が記述されている．これらの遅延をモンテカルロシミュレーションと同様の要領で変動させる．各ゲート遅延は互いに無相関とし，それぞれに正規分布に従った変動量を与える．

本稿では以下のように評価する．文献 3) によると 65nm テクノロジでのゲート遅延のばらつきは σ/μ で 0.064 である．それを基にばらつきがより深刻になる場合と軽減される場合を想定し，ゲート遅延の変動量が 0.04，0.064，0.08，0.1 の場合でそれぞれ作成する．また，作成する SDF ファイルのサンプル数はそれぞれの解析で 10,000 とする．

5.3 解 析 結 果

5.3.1 ゲート遅延変動量が回路遅延に与える影響

2 入力 CSLA と 3 入力 CSLA の遅延分布を**図 5** に示す．横軸は遅延を，縦軸は度数を示す．図中の左側の一群が 2 入力 CSLA の，右側の一群が 3 入力 CSLA の分布である．図から分かるようにゲート遅延の変動量が増加するに従って，演算器の遅延は増加し，ばらつきは拡大している．

表 2 に遅延の μ，σ，及び σ/μ をまとめる．表 2 の 5 列目は 2 入力 CSLA の σ/μ に対する 3 入力 CSLA の σ/μ の改善率を示している．2 入力 CSLA と比較して 3 入力 CSLA は遅延が大きいため，当然 μ と σ は大きくなる．しかし，σ/μ は小さくなっており，遅延ばらつきが小さくなっていることが分かる．

以上から，図 2 で示した統計的性質を定量的評価で

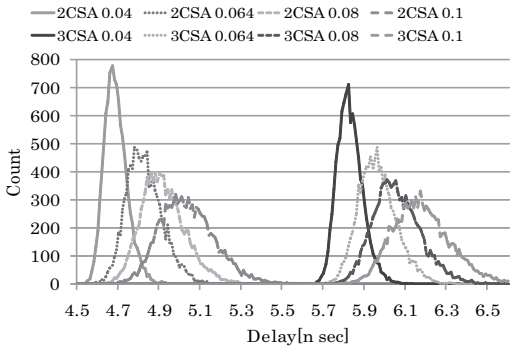

図 5 ゲート遅延変動量に対する回路遅延分布の変化

表 2 ゲート遅延変動量に対する回路遅延の平均と標準偏差の変化

	Gate σ/μ	μ	σ	σ/μ	Imp.(%)
2CSLA	0.04	4.68	0.056	0.0120	–
	0.064	4.82	0.088	0.0183	–
	0.08	4.91	0.108	0.0219	–
	0.1	5.03	0.134	0.0267	–
3CSLA	0.04	5.82	0.062	0.0107	10.8
	0.064	5.96	0.093	0.0155	15.1
	0.08	6.05	0.117	0.0192	12.5
	0.1	6.18	0.143	0.0231	13.4

確認できた．つまり，演算カスケーディングを行う演算器を利用することで回路の論理段数を大きく出来，その結果遅延ばらつきを縮小出来ることが確認された．

5.3.2 演算器数が回路遅延に与える影響

3 節で説明したクリティカルパスが回路遅延に与える影響を考慮すると，演算器数がプロセッサ全体の遅延に影響を与えることが分かる．スーパスカラプロセッサは演算器を複数持つが，演算器数が増加すると実行ステージのクリティカルパス数が増加する．クリティカルパス数は回路遅延に影響するので，その影響を調査する．5.3.1 節で得られた遅延分布を用いて統計的 MAX 演算[2] で求める．なお，本節より 5.3.1 節で得られた遅延分布を正規分布に近似して扱う．

図 6 に演算器数を変えたときの遅延分布を示す．ゲート遅延変動量は 0.064 を用いている．横軸は遅延を，縦軸は出現度を表す．図中の左側の一群が 2 入力 CSLA の分布を，右側の一群が 3 入力 CSLA の分布である．2 入力 CSLA は演算器数が 1，4，6，8 のときの分布を，3 入力 CSLA では演算器数が 1，2，3，4 のときの分布をそれぞれ示す．このとき，基本構成要素である 2 入力 CSLA の数では両者で同じになっている．図から演算器数が増加すると，遅延の平均は増加し，ばらつきは縮小していることが確認できる．

表 3 に遅延の μ，σ，σ/μ をまとめる．基本構成要素の 2 入力 CSLA が 1 から 8 へ増加するとき，2 入力 CSLA では μ が 2.6%増加し σ/μ が 40%縮小している．一方，3 入力 CSLA では μ が 1.6%増加し σ/μ が 31%縮小している．しかし，基本構成要素の 2 入力

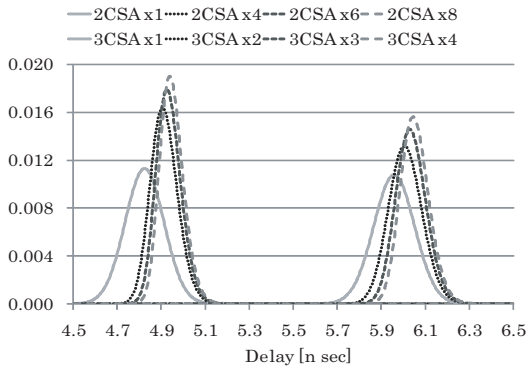

図 6 演算器数に対する回路遅延分布の変化

表 3 演算器数に対する回路遅延の平均値と標準偏差の変化

		μ	σ	σ/μ
2CSLA	x1	4.82	0.088	0.0183
	x4	4.91	0.062	0.0126
	x6	4.93	0.057	0.0115
	x8	4.95	0.054	0.0109
3CSLA	x1	5.96	0.093	0.0155
	x2	6.01	0.076	0.0127
	x3	6.04	0.069	0.0115
	x4	6.05	0.065	0.0107

CLSA の数が同じときには両者に有意差は無く，演算器数の回路全体の遅延に与える影響は同程度である．

6. プロセッサの IPC の評価

本節では演算カスケーディングがプロセッサの IPC に与える影響を評価する．これ以降，それを利用するプロセッサをカスケード・プロセッサと呼ぶ．

6.1 評価環境

SimpleScalar ツールセット[1] を用いて評価環境を構築した．命令セットは Alpha 命令セットを用いる．ベンチマークプログラムには 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．カスケーディング演算を行わないプロセッサ（基準プロセッサと呼ぶ）の構成を**表 4** にまとめる．これが比較基準となる．基準プロセッサは 4 命令，6 命令，8 命令発行のプロセッサを用意する．これらのプロセッサは，命令発行幅と整数 ALU の数以外の構成で共通である．

6.2 カスケード・プロセッサの構成

カスケード・プロセッサの構成について説明する．カスケード実行される演算は整数 ALU 命令のみを対象とする．本稿では 5 節で求めた遅延分布を整数 ALU の遅延分布とし評価を行う．カスケード実行を行う演算器の回路遅延のため，カスケード・プロセッサの実行ステージの遅延は基準プロセッサに比べ増加する．典型値での遅延が 25%増加するので，基準プロセッサに対して 80%の動作周波数となる．4 節で説明したように，本稿では実行ステージがプロセッサの動作周波数

表 4 基準プロセッサの構成

Fetch width	8 instructions
L1 I-cache	32KB, 2way, 1 cycle
Branch predictor	gshare + bimodal gshare:4K entries, 12 histories bimodal:4K entries
RUU size	128 entries
Issue width	4/6/8 instructions
Integer ALUs	4/6/8 units, 1 cycle
Integer multipliers	1 units MULT 3 cycles, DIV 20 cycles
Floating ALUs	4 units, 2 cycles
Floating multipliers	1 unit MULT 4 cycles, DVI 12 cycles, SQRT 24 cycles
L1 D-cache port	2 ports
L1 D-cache	32KB, 4way, 2 cycles
Unified L2 cache	4MB, 8way, 8cycles
Memory	150 cycles
Commit width	8 instructions

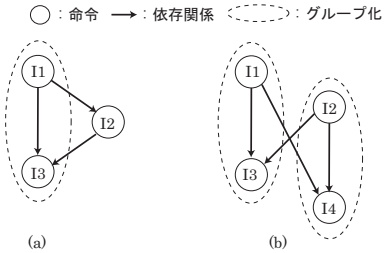

図 7 デッドロック状態のグループ化

を決定すると仮定して議論しており，以上はこの仮定に従っている．同様に 4 節で述べたように，評価した遅延分布は実行ステージについてのみであるが，動作周波数が低下することから，実行ステージ以外のステージはタイミング制約が緩和される．

3 入力 ALU での演算カスケーディングには以下の制約を設ける．以降，カスケーディング実行される演算を特定することをグループ化と呼び，カスケーディングされない命令をグループ化不可能命令と呼ぶ．

① グループ化の対象は整数 ALU 命令のみとする．ただしロード／ストア命令はアドレス計算とメモリアクセスに分離されており，アドレス計算部はグループ化の対象とする．

② 2 命令のグループ化のみを許す．先行命令を生産者，後続命令を消費者と呼ぶ．また，どの命令も複数のグループには属さない．

③ 命令ウィンドウへのディスパッチ時にグループ化を施す．グループ化の探索範囲はウィンドウ内に限られる．生産者を見つけられない場合には，グループ化不可能命令となる．

④ ディスパッチ時に利用不可能なオペランドを 2 つ持つ命令はグループ化の消費者にはなり得ない．これは**図 7** に示すようなグループ化を防ぐためである．グループ化された命令は同時に発行されなければらないので，図のグループ化は命令発行できないデッドロックを起こす．

表 5 カスケード・プロセッサの構成

	Symmetry	Asymmetry
4 issue :2in ALU	–	2 units, 1 cycle
:3in ALU	2 units, 1 cycle	1 unit, 1 cycle
6 issue :2in ALU	–	4 units, 1 cycle
:3in ALU	3 units, 1 cycle	1 unit, 1 cycle
8 issue :2in ALU	–	4 units, 1 cycle
:3in ALU	4 units, 1 cycle	2 units, 1 cycle
Common :L2 cache	4MB, 8 way, **7 cycles**	
:Memory	**120 cycles**	

カスケード・プロセッサの演算器構成について述べる．2 種類の演算器構成を検討する．1 つは 3 入力 ALU のみを備える構成であり，これを均質構成と呼ぶ．もう 1 つは 2 入力 ALU と 3 入力 ALU を混在させる構成であり，これを不均質構成と呼ぶ．**表 5** にカスケード・プロセッサの構成をまとめる．例えば 6 命令発行プロセッサでは，均質構成は 3 入力 ALU のみを 3 つ備え，不均質構成は 2 入力 ALU を 4 つと 3 入力 ALU をひとつ備える．これらの ALU の数を決定する際には，レジスタポート数に注意した．基準プロセッサの備えるレジスタファイルのポート数を増やさないという制約をおいている．6 命令発行の場合，基準プロセッサのレジスタポート数は 12 read 6 write であるが，均質構成では 9 read 6 write，不均質構成では 11 read 6 write である．加えて，同じ命令発行幅では 2 入力 ALU 換算で総演算器数が各構成間で同一になることにも注意を払っている．これは，演算器数の増加やオペランドバイパスネットワークの複雑度増加によりチップ面積が増加する影響を極力排除するためである．面積増加の歩留まりへの影響を無視できるように配慮している．また，総演算器数と発行幅とは等しい．つまり，3 入力 ALU で 2 命令をカスケード演算するためには，2 命令の発行幅を必要とする．一方不均質構成での二種類の ALU の比率は，予備評価を行いグループ化可能な命令の割合を調査した上で決定している．

不均質構成では，グループ化可能命令を 3 入力 ALU に，グループ化不可能命令を 2 入力 ALU に発行する．一方均質構成では，グループ化不可能命令も 3 入力 ALU で実行しなければならない．その結果を得るために，余る演算器では 0 加算などの無意味な演算を実行する．この無意味な演算の結果はレジスタには書き込まない．上述したようにグループ化可能か否かはディスパッチ時に判明するので，命令ウインドウ中にグループ化可能/不可能を表すフラグを用意し，それに基づいて命令を発行する．そのフラグに基づいて，不均質構成では発行先の演算器を決定し，均質構成では無意味な演算を表す制御信号を生成する．

以上の演算器構成と命令発行ポリシーは IPC に影響を与えると予想される．均質構成では，グループ化不可能な命令が多い場合に，演算器の総数が基準プロセッサよりも少ないために，演算器資源が不足し IPC が低

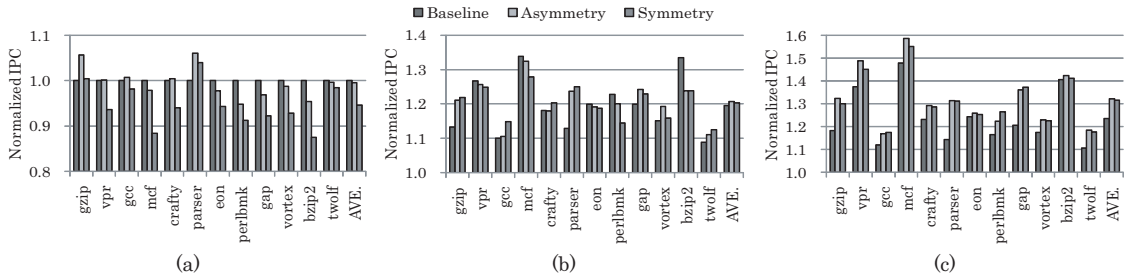

図 8 IPC の評価結果：(a)4 命令発行構成，(b)6 命令発行構成，(c)8 命令発行構成

下する可能性がある．一方，不均質構成では，グループ化可能な命令が多い場合には 3 入力 ALU が不足して，命令発行を妨げる状況を生じ IPC が低下する可能性がある．これらの IPC に与える影響も評価する．

表 5 には，メモリと 2 次キャッシュのアクセスレイテンシも記載されている．動作周波数が低下するために，基準プロセッサと比較するとこれらのレイテンシはサイクル数では小さくなっている．

6.3 評価結果

図 8 に 4 命令，6 命令，8 命令発行のプロセッサ構成での IPC をそれぞれ示す．全ての値は 4 命令発行基準プロセッサの IPC で正規化されている．横軸はプログラムを示す．図中の Baseline は基準プロセッサを，Asymmetry は不均質構成の，Symmetry は均質構成のカスケード・プロセッサをそれぞれ示す．

はじめに，演算カスケーディングが IPC に与える影響について考える．4 命令発行ではカスケード・プロセッサの IPC は多くのプログラムで基準プロセッサよりも低い．平均では不均質構成で 0.5%，均質構成で 5.4%，それぞれ IPC が低下している．6 命令発行では若干の改善が得られている．不均質構成で平均 1.0%，均質構成で平均 0.6%向上している．8 命令発行では改善度が拡大している．不均質構成と均質構成での IPC の改善は，平均でそれぞれ 6.9%と 6.4% である．いずれの場合も，期待したほどの IPC 改善はできていない．

IPC を改善出来ない理由のひとつは，グループ化可能命令が少ないことである．**図 9** に 4 命令不均質構成プロセッサでのプログラム実行時の命令の内訳を示す．縦軸は全実行命令数に占める割合を示している．Prod と Cons がグループ化された命令を示し，それぞれ生産者と消費者の割合を示す．Ungrouped はグループ化不可能命令を，Others はグループ化対象外の命令を示す．グループ化命令は平均で 29%，グループ化不可能命令は平均で 41%存在する．その他のプロセッサ構成でも同様の傾向のため，それらのグラフは省略する．グループ化可能命令はそれほど多くないことがわかる．グループ化可能命令が少ない理由は，グループ化相手命令の探索が命令ウインドウという非常に狭い範囲に限定されることである．

第二の理由は，ベースとなるプロセッサ性能が低下

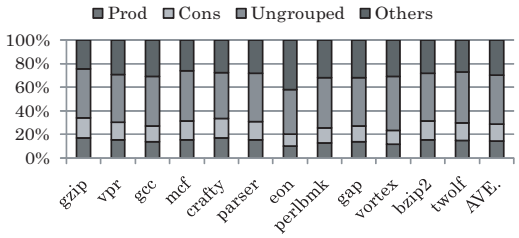

図 9 プログラム実行時の命令の内訳

していることである．カスケード・プロセッサは基準プロセッサに演算カスケーディングの機能を追加しているわけではない．上述したように，現実的な実装になることに配慮して，2 入力 ALU 換算での演算器総数を増やしていない．つまり，利用可能な命令レベル並列度を犠牲にして演算カスケーディングを実現し，カスケード・プロセッサとしている．言い換えると，演算カスケーディングが効果を発揮できるのは，基準プロセッサで命令レベル並列度が小さい場合に限られる．

次に，IPC の改善（または悪化）の度合いと発行幅との関係を考察する．4 命令発行で IPC が低下するのは，4 命令発行では基準プロセッサが利用可能な命令レベル並列性を十分に引き出せるためである．基準プロセッサはサイクル当たり 4 命令までの命令レベル並列性を利用できる．対してカスケード・プロセッサでは，2 入力 ALU 換算の 4 つの演算器を全て利用できるケースは非常に稀である．均質構成では，グループ化不可能命令が存在すると無駄な演算を実行しなければならず，演算器を全て利用することができない．不均質構成では，2 命令よりも多くの命令レベル並列性を利用出来ない上，同時に実行可能な演算カスケーディングは一組に限られる．グループ化不可能命令はサイクル当たり 2〜3 命令しか実行できないため，実行スループットが低下している．

6〜8 命令発行で IPC の改善度合いが向上するのは，命令発行幅が増加するに従って基準プロセッサが命令レベル並列性を抽出することが困難になるためである．基準プロセッサで演算器の利用効率が低下する一方で，カスケード・プロセッサでは依存関係にある命令をグループ化することで余っている演算器の利用効率が上がる．これらのことから，演算カスケーディングはよ

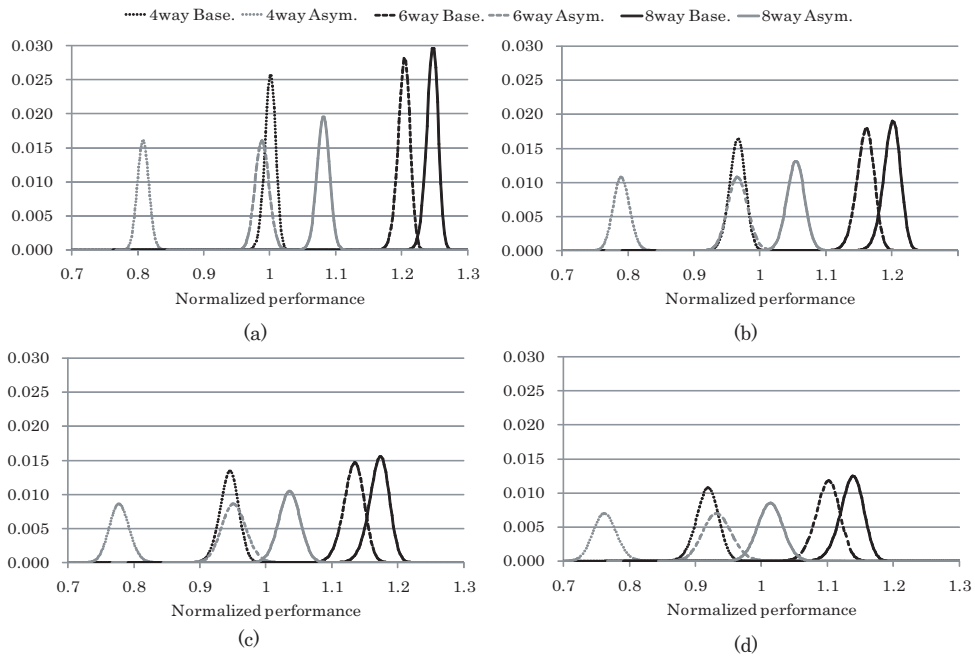

図 10 プロセッサ性能の分布：ゲート遅延変動量= (a) 0.04，(b) 0.064，(c) 0.08，(d) 0.1

り発行幅の広いプロセッサに適用することで IPC を改善することが可能であることが分かる．

続いて，演算器構成による IPC の変化について考える．全ての発行幅で，均質構成よりも不均質構成の方が大きな IPC を示している．しかし，4 命令発行に比べ，6 命令発行と 8 命令発行では IPC の差は小さい．不均質構成の方が IPC をより向上できている．均質構成では演算器の利用効率が悪いためである．図 9 で見たように，グループ化命令が不可能命令に比べて多くないためである．このため，グループ化不可能命令による演算器の利用効率悪化の影響の小さな不均質構成の方がより高い IPC を得ることができている．グループ化の探索範囲を拡大すればグループ化可能命令の増加が期待できるので，オペランド表[11] などの利用で探索範囲を拡大できれば，均質構成での IPC を改善できる可能性があると考えられる．

7. タイミング歩留まり

5 節で求めた遅延分布と 6 節で評価した IPC とからタイミング歩留まりについて考察する．動作周波数と IPC との積からプロセッサ性能を求め，その分布を用いて，性能を考慮したタイミング歩留まりについて考察する．その分布上で性能の下限を与えると，それを満たすものと満たしていないものに分けることができ，歩留まりを比較できる．

図 10 にゲート遅延の変動量が 0.04，0.064，0.08，0.1 のときのそれぞれの性能分布を示す．横軸は性能を，縦軸は出現度を示す．変動量 0.04 のときの 4 命令発行基準プロセッサの結果で正規化している．カスケード・プロセッサについては IPC の大きい不均質構成の結果のみを示す．不均質構成では二種類の演算器が混在するが，クリティカルパスとなるのは 3 入力演算器であるので，3 入力演算器数のみを考慮する．

はじめに，基準プロセッサとカスケード・プロセッサをそれぞれの命令発行幅で比較する．ゲート遅延の変動量に関わらず，平均性能は前者が高い．IPC の評価では 6 命令と 8 命令発行のときに，基準プロセッサに対してカスケード・プロセッサに改善が見られている．それにも関わらず性能が低下しているのは，動作周波数の低下による性能悪化が大きいためである．一方，それぞれの命令発行幅で，ばらつきを比較すると，カスケード・プロセッサの方で σ/μ が僅かに大きい．これは演算器構成が要因である．不均質構成では 3 入力演算器数が少ないため，クリティカルパス数の増加によるばらつき縮小の効果が少ないためである．

次に，ゲート遅延の変動量による影響を考える．ゲート遅延変動量が増加すると動作周波数が低下するために性能も低下する．基準プロセッサとカスケード・プロセッサでは影響の受け方が異なる．5.3.1 節の評価結果から，遅延変動量増加の影響をより大きく受けるのは 2 入力 CSLA である．つまり，ゲート遅延が増加した際により動作周波数が低下するのは基準プロセッサである．その影響を最も観察できるのは 4 命令発行基準プロセッサと 6 命令発行カスケード・プロセッサで

の性能分布の変化である．ゲート遅延変動量が 0.04 のときはカスケード・プロセッサの平均性能の方が低いが，変動量が増加するに従って基準プロセッサの平均性能が低下し，変動量が 0.1 のときにはカスケード・プロセッサの性能を下回っている．

演算カスケーディングを実行できる演算器を採用すると動作周波数が低下するため，同一の命令発行幅のプロセッサ構成では性能は悪化する．6 命令と 8 命令発行のプロセッサ構成では演算カスケーディングで IPC を改善しているが，周波数の低下を打ち消すほどではない．また，不均質の演算器構成では演算器数が少ないため，周波数ばらつきの改善は得られてない．一方，ゲート遅延変動量増加の影響は基準プロセッサの方がより大きく受けることが確認される．

以上の考察から以下の知見が得られる．演算器単体ではばらつきを改善できたが，演算器の遅延増の影響，演算器数の影響，IPC の改善度合いから総合的に判断すると，性能を考慮したタイミング歩留まりを改善できていないと結論できる．したがって，タイミング歩留まりの改善にはプロセッサ性能の維持が必須の要件であり，性能を維持するためには動作周波数の低下を抑えることと IPC を改善することとが必要である．以下では，これらの性能維持の方策について考察する．

まず，動作周波数の低下を抑えるためには，回路遅延の小さな演算器の利用が考えられる．例えば，桁上げ保存加算器を用いる 3 入力 1 出力の複合演算器である．桁上げ保存加算器は遅延が非常に小さいので，動作周波数の低下を抑えられると期待できる．しかし以下の 2 点を慎重に検討する必要がある．第一に，3 入力 1 出力の複合演算器を利用するためには，マイクロアーキテクチャの変更が必要になる．途中の演算結果が失われてしまうので，正確な割込みを実現するための何らかの工夫が必要である．そのための回路が複雑であれば，逆に動作周波数を低下しかねないし，タイミング歩留まりへの影響も懸念される．第二に，幸運にも動作周波数の低下を抑えることが出来たとすると，実行ステージ以外のステージにおけるタイミングマージンが小さくなる．一方で，複合演算器の利用によりレジスタ利用効率が改善される効果が期待されるため，IPC が向上する可能性がある．IPC の改善には，グループ化対象命令の探索範囲を拡大することが効果的だと思われる．しかし，そのための機構を追加する必要があり，その回路が複雑であれば，動作周波数を低下しかねないし，タイミング歩留まりへの影響も懸念される．

以上を考慮すると，もはや実行ステージが動作周波数を決定するという仮定は満足できず，実行ステージだけを考慮するのでは不十分である．つまり，ばらつきの問題は局所的な改善を検討するだけでは不十分で，プロセッサシステム全体を大局的に検討する必要があることを示唆している．性能低下を引き起こす工夫は受け入れがたく，性能低下を抑制できるマイクロアーキテクチャを構築するためには，プロセッサの一部のみに着目するのではなく全体に配慮する必要がある．ひとつには，演算器単体のばらつきよりもプロセッサ性能の方がタイミング歩留まりに大きく影響するからである．第二には，演算器単体のばらつきを改善できても，演算器数という並列度が変わることで容易にばらつきの大小関係が入れ替わるからである．この知見は今後の研究において大きな意味があると言えよう．

8. ま と め

マイクロアーキテクチャレベルでタイミング歩留まりを改善するために，演算カスケーディングの利用を提案した．評価の結果，回路レベルではばらつきを縮小できるものの，プロセッサレベルでばらつきを縮小することはできなかった．プロセッサレベルでタイミング歩留まりを改善するには，小手先の工夫では不十分で，抜本的なマイクロアーキテクチャの見直しが必要であるという知見が得られた．

謝辞 本研究は一部，科研費・基盤 A(19200004)，基盤 B(20300019)，および，JST CREST プログラムの支援による．なお，東京大学 VDEC を通じて提供された日立 0.18μm ライブラリを使用している．

参 考 文 献

1) T. Austin et al., “SimpleScalar: an infrastructure for computer system modeling”, Computer, 35(2), 2002.
2) M. Berkelaar, “Statistical delay calculation, a linear time method”, Int. Workshop on Logic Synthesis, 1997.
3) K. Bernstein et al., “High-performance CMOS variability in the 65-nm regime and beyond”, IBM Jour. Res. & Dev., 50(4/5), 2006.
4) S. Borkar et al., “Parameter variations and impact on circuits and microarchitecture”, 40th DAC, 2003.
5) K. A. Bowman et al., “Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration”, Jour. Solid-State Circuits, 37(2), 2002.
6) D. Ernst et al., “Razor: a low-power pipeline based on circuit-level timing speculation”, MICRO-36, 2003.
7) M. Hashimoto et al., “Increase in delay uncertainty by performance optimization”, ISCAS, 2001.
8) X. Liang et al., “Mitigating the impact of process variations on processor register files and execution units”, MICRO-39, 2006.
9) D. Mohapatra et al., “Low-power process-variation tolerant arithmetic units using input-based elastic clocking”, ISLPED, 2007.
10) M. Ozawa, “Cascade ALU Architecture: Preserving Performance Scalability with Power Consumption Suppressed”, COOL Chips V, 2002.
11) P. G. Sassone et al., “Dynamic strands: collapsing speculative dependence chains for reducing pipeline communication”, MICRO-37, 2004.
12) A. Tiwari et al., “ReCycle: pipeline adaptation to tolerate process variation”, ISCA-34, 2007.
13) S. Vassiliadis et al., “Interlock collapsing ALU’s”, Trans. Comput., 42(7), 1993.
14) X. Vera et al., “X-Pipe: an adaptive resilient microarchitecture for parameter variations”, ASGI, 2006.
15) 小野寺, “ばらつきを克服する設計技術”, 軽井沢 WS, 2006.