

命令レベル逐次プロセッサ

佐藤 寿倫

九州大学 システム LSI 研究センター
〒814-0001 福岡市早良区百道浜 3-8-33-3F
E-mail: toshinori.sato@computer.org

LSI の製造プロセスにおける微細化が進展するにつれ、従来は考慮する必要のなかった様々な問題が顕在化している。それらは、消費電力の増大、ソフトエラー耐性の低下、そしてばらつきの拡大である。なかでもランダムなプロセスばらつきに起因する性能ばらつきの問題は、設計手法に対して本質的な転換を求めるほどに深刻である。われわれは、性能ばらつきを縮小できるマイクロアーキテクチャの研究を行っている。回路における遅延の統計的性質に着目し、論理段数を大きく出来る命令カスケードの活用を検討している。本稿ではその予備評価結果を紹介する。ばらつきを考慮できたとしても、性能を大幅に悪化させたのでは問題を解決できているとは言えない。命令カスケードによるスループット維持の可能性について考察する。

Hybridizing Instruction-Level Parallelism and Serialism

Toshinori Sato

System LSI Research Center, Kyushu University
3-8-33-3F, Momochihama, Sawara-ku, Fukuoka, 814-0001 Japan
E-mail: toshinori.sato@computer.org

Continuous advance in semiconductor technologies unveils several problems, which have not always been seriously considered. They are power consumption, soft errors, and variations. Especially, performance variations due to random process variations are severe as they require fundamental change in design methodologies. We are currently investigating microarchitectural techniques than attain the reduction in variations. By exploiting the statistical characteristics in circuit delay, we are considering the use of instruction cascading, which effectively increases logic depth. This paper presents preliminary evaluation results. Even though variations can be considered, techniques diminishing performance will not be a solution. We discuss how throughput can be maintained by the instruction cascading.

1. はじめに

LSI の微細化の進展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大させ、これまでマイクロ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、消費電力の爆発、ソフトエラー耐性の悪化、そして性能ばらつきという深刻な問題が顕在化している。本稿では性能ばらつきに注目し、それを考慮できるマイクロアーキテクチャについて考察する。

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。①不純物密度の不均一な分布[10]、②リソグラフィ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、そして④温度分布の不均一さが、WID ばらつきの要因と考えられる[2, 7]。①と②による閾値電圧のゆらぎと③はトランジスタ速度に影響し、同様に閾値電圧のゆらぎと④はリーク電流に影響する。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。最悪値を用いたのではランダムなばらつきを十分には解析できず、統計的な取り扱いが必要である。つまりマイクロプロセッサの設計においても最悪値を考慮した設計は不可能であり、新しい設計手法への転換が必要である。

わたしたちはこれまでに、典型値指向設計手法（あるいは Better-than-worst-case 設計）のひとつとして、建設的タイミング違反方式を検討してきた[12]。マイクロアーキテクチャと回路とを協調させる方法であり、

冗長性を利用する点で近年のディベンダブルコンピューティングの流れとも相性の良い方式だと思うが、残念ながら必ずしも十分な支持を得てはいない。本稿では、生じてしまった性能ばらつきに対処できるようにするのではなく、元から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。

次節で回路における遅延の統計的性質を説明し、マイクロアーキテクチャへの要請をまとめる。本稿ではトランジスタ速度のばらつきに注目する。リーク電流におけるばらつきの配慮は将来の課題である¹。3節で性能ばらつきに配慮できる命令レベル逐次プロセッサを提案し、4節でその予備評価を行う。5節では関連研究を紹介し、6節でまとめとする。

2. 遅延の統計的性質

楢本ら[6]やBowmanら[3]は、以下に紹介する回路遅延の統計的性質を見出している。彼らの知見に基づいて、マイクロアーキテクチャへの要請を検討する。

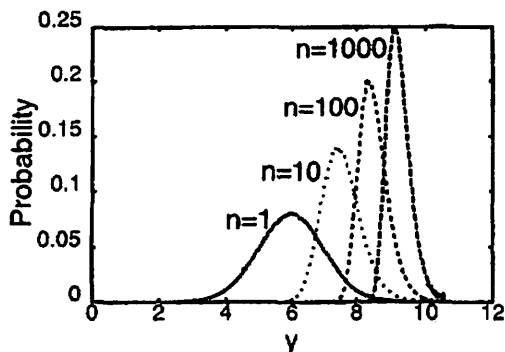


図1：最長遅延パス数の影響[6] (©2001 IEEE)

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[6]。各パス遅延は無相関で、平均が6で分散が1である正規分布 $N(6, 1^2)$ にしたが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 $N(6, 1^2)$ の正規分布である。容易に分かるように、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

¹ リーク電流におけるばらつきを検討対象から外す理由は以下のとおりである。リーク電流のメモリに対する影響と比較して、ロジック部における深刻さがわれわれにはまだ理解できていない。現状では、チップ全体でのリーク電流を配慮する以上の必要性をわれわれはまだ認識できていない。

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。例えばメモリはそのような回路である。

一方で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。したがって統計的な性的遅延解析手法 (statistical static timing analysis, SSTA) が確立されれば、ばらつきを減少させる方法としてこの性質を利用できる。

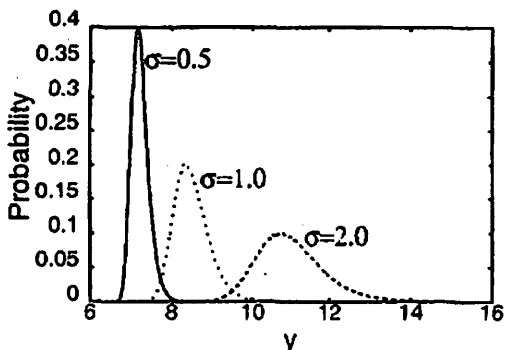


図2：論理段数の影響[6] (© 2001 IEEE)

図2は最長遅延パスの論理段数が回路遅延に及ぼす影響を示している[6]。パスが互いに無相関だと仮定しているので、論理段数 m が深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の変化で表したものが図2である。パスの総数 n は100本である。標準偏差 σ が大きいほど論理段数 m が浅いことに相当する。容易に想像されることではあるが、論理段数が浅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。例えば近年の深い命令パイプラインはそのような回路である。

以上をまとめると、ばらつきに配慮したマイクロアーキテクチャへの要請は以下のとおりである。

1. 最長遅延パスの数が多いこと
2. パイプラインの段数が小さいこと

ただし、SSTAの確立を前提としている。

3. 命令レベル逐次プロセッサ

前節で紹介した遅延の統計的性質を考慮し、ばらつきに配慮できるマイクロアーキテクチャを検討する。

3.1. 命令カスケディング

まず上記の要請 2 に着目する。パイプライン段数が小さいことが必要であるが、これはちょうど近年の高クロック周波数志向のマイクロアーキテクチャから逆行することになる。したがって、従来の深いパイプラインを浅くするために、複数のパイプラインステージを統合することが望ましい。そこで、命令レベル逐次性の利用について検討する。図 3 に命令レベル逐次性の利用のイメージを示す。

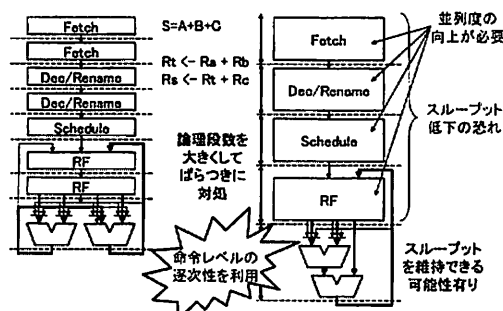


図 3：命令レベルの逐次性を利用

フロントエンドでは深いパイプラインを実現するために、デコードなどの処理をより小さな処理に分割している。したがってステージの統合は容易である。もちろん、安易な統合は命令スループットを低下させるため、命令レベル並列度の向上が必要になる。ただし、従来と比較して並列度を向上させる機構の導入が容易である。例えば、1 サイクルで実行する必要のある命令スケジューリングのステージには時間的な余裕が生まれることになるので、複雑なスケジューリングが可能になり大きな性能改善が期待できる。

一方バックエンド、特に演算実行ステージは、1 ステージ内に処理を終えるように配慮した設計がこれまで行われてきたため、ステージの論理段数を増加させることは容易ではない。そこで命令カスケディング [12] を利用する。互いに依存関係にある命令を同時には実行できないので、これらをグループ化して発行することで、演算ステージにおける論理段数を大きくする。言い換えれば、命令レベルの逐次性を利用しているわけである。充分な数の依存命令の組を用意できれば、従来と同等の命令スループットを維持できる可能性があると思われる。この結果、性能を維持しつつ、ばらつきを縮小することが期待できる。

すなわち、演算実行ステージでの命令レベル逐次性の利用と、残りのステージでの命令レベル並列性の利用とを併用することで、従来のプロセッサ性能を維持

しつつ、ばらつきを縮小できる。

3.2. 履歴の利用

メモリ構造は上記の要請 1 とのマッチングが高い。実行履歴を利用した動的な適応方式がこれまでに多く提案されているが、それらのほとんどは専用メモリに履歴あるいは適応結果を保持して利用している。近年の高クロック周波数志向のマイクロアーキテクチャでは、メモリアクセス速度が足枷になるために、これらの方式は好まれない。しかし、細かくパイプラインを切って動作周波数を向上し、プロセッサの性能を改善するという方法は、明らかに要請 2 に反しており性能ばらつきに対して耐性が無い。寧ろ要請 1 とのマッチングが高いという意味から、専用メモリを利用して命令レベルの並列度を向上させる方式の方が、ばらつきへの配慮という観点から将来の有望なマイクロアーキテクチャであると言える。

更には、演算や制御もメモリに置き換えることが望ましいかも知れない。

3.3. クラスタ型コア

長配線は突出した遅延パスになる可能性が高く、要請 1 の観点から好ましくない。クラスタ型プロセッサは配線遅延に配慮したアーキテクチャであり、ばらつきに配慮するマイクロアーキテクチャとして有望である。クラスタ化することで長配線を取り除くことが出来る。その結果遅延の増った最長遅延パスを増加することができる。最長遅延を削減しつつその数を増加できるので、回路遅延を大幅に増大させることなくそのばらつきを縮小できる。クラスタ型コアを用いるマルチコアプロセッサ [9] はソフトウェアなどに対する信頼性に配慮されており、ばらつきに起因する動作異常にも対処できると思われる。

クラスタ型コアの利得については、近い将来に稿を改めて紹介するつもりである。

4. 予備調査

本節では、命令レベル逐次プロセッサで命令カスケディングを実施することにより、命令スループットを維持できる可能性について調査する。

4.1. 調査環境

SimpleScalar ツールセット [1] を用いたシミュレーションで調査する。命令セットには Alpha ISA を選択する。プロセッサ構成は表 1 の通りである。ベンチマークには SPEC2000 の整数系プログラム全てを用いる。入力は ref データセットである。なお、最初の 2 億命令をスキップし、続く 200 万命令を評価対象とする。

表 1: プロセッサ構成

Fetch width	8 instructions
L1 instruction cache	16K, 2 way, 1 cycle
Branch predictor	gshare + bimodal
Gshare predictor	4K entries, 12 histories
Bimodal predictor	4K entries
Branch target buffer	1K sets, 4 way
Dispatch width	4 instructions
Instruction window size	64, 128, 256, 512 entries
Issue width	4 instructions/strands
Integer ALUs	4 units
Integer multipliers	2 units
Floating ALUs	1 unit
Floating multipliers	1 unit
L1 data cache ports	2 ports
L1 data cache	16K, 4 way, 2 cycle
Unified L2 cache	8M, 8 way, 10 cycles
Memory	Infinite, 100 cycles
Commit width	8 instructions

4.2. 調査項目

パイプラインステージ当たりの論理段数を増加させても、すなわち動作周波数を低下させても、命令カスケディングにより命令スループットを維持できるかどうかを調査する。具体的には、まず全命令に対して、それに依存する命令の数を調査する。すなわち、各生産者に対する消費者数の分布を調査する。命令カスケディングを実施するためには、多くの生産者・消費者をグループ化できることが必要である。統いて、生産者と消費者の距離を調査する。プログラムの実行時に上記のグループを検出するためには、両者の距離が小さいことが必要である。距離が大きいと多くの命令を保持する機構が必要なうえ、保持されている命令から生産者・消費者のグループを検出する機構が複雑になる。

二通りの場合を調査する。いずれもプログラムを静的に解析するのではなく、実行される命令を対象にした動的な解析を行う。さらに、プログラム実行として意味のあるコミットされた命令についてのみ調査するのではなく、投機失敗時に実行される命令も含めた調査を行う。まず全ての生産者に対して、その後続の N 命令 (N=64, 128, 256, 512) 中に現れる消費者について調査する。この場合には、以下の二点を除いて、プログラムの振る舞いに影響を受けない。まず、投機失敗時に実行される命令についても調査するために、表 1 に示した分岐予測機構を用いて命令トレースを生成する。全ての命令はインオーダーに調査されるので、各分岐命令には最初に予測された分岐先の命令が続く。もう一点はシステムコール命令の扱いである。システムコール命令で命令ウィンドウが輻になるようなスケジューリングを実施しているため、システムコール命令が出現すると探索範囲 N が限定されることになる。

統いて、プログラム実行時の生産者・消費者について調査する。この場合には常に命令ウィンドウが一杯になるわけではない。

同様の調査がすでになされている[4, 5, 13]が、レジスタではなく命令に着目している点、生産者・消費者間の距離を調査している点、コミットされる命令だけでなく投機失敗時に破棄される命令が対象に含まれている点、実行中か否かで探索範囲を変えて調査している点に違いがある。

4.3. 結果

図 4 に、各生産者の後続 N 命令中に存在する消費者数の分布を示す。各プログラムに対して結果を表すグラフが 4 本あり、それぞれ N が 64, 128, 256, そして 512 命令の場合である。さらに各グラフは消費者数の分布を表しており、下から順に消費者が 0 個, 1 個, 2 個, 3 個, そして 4 個以上の場合を示している。N が大きくなっても結果に大きな違いは見られず、消費者が 0 個の割合が減る一方で消費者が 4 個以上の割合が増えるプログラムが若干見られるのみである。概ね、2 割の命令が消費者を持たず、他の 2 割の命令が 2 個以上の消費者を持つ。先行研究[4, 5]と比較すると消費者を持たない命令の割合が大きいが、消費者の探索範囲を 512 命令までに限定していることと、分岐予測機構がひ弱なために間違って実行される分岐先命令が多いことが理由だろうと予想される。

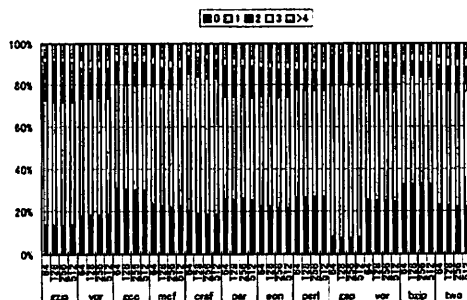


図 4: 依存命令数の分布

図 5 は、生産者とその全消費者間の平均距離 (distance) と、生産者とその最初に現れた消費者との距離の平均 (nearest) を表している。図中、隣り合うプログラム間の結線には意味は無いので、無視された。全距離の平均がプログラムの違いや探索命令数 N の違いに大きく影響されるのに対し、最近接距離の平均はそれらの影響が小さい。全プログラムの平均で、N が 64 命令と 512 命令の時にそれぞれ 5.3 命令と 6.2

命令である。

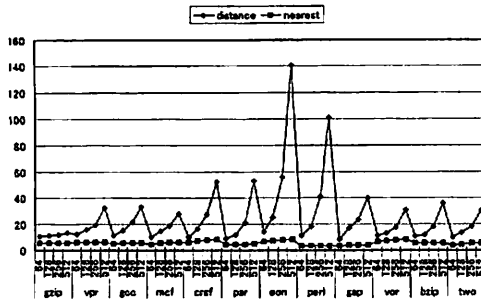


図 5：全距離の平均と最近接距離の平均

図 6 には、最近接距離の分布を示す。図 4 と同様の構成をしており、各グラフにおける分布は、下から順にそれぞれ 8 命令、16 命令、32 命令、64 命令以内に最初の消費者が現れる割合、そして 64 命令以降にそれが現れる割合である。こちらもプログラムや探索命令数 N の違いの影響は小さい。概ね、消費者を持つ生産者の 6 割で 8 命令以内に消費者が現れる。一方で、約 2 割の生産者には 64 命令以内に消費者が現れない。

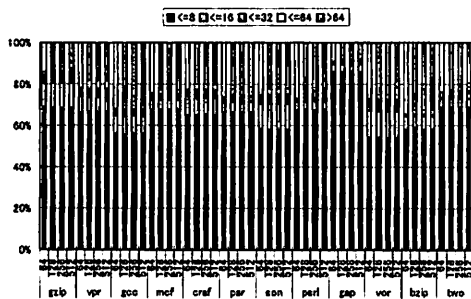


図 6：最近接距離の分布

以上の結果から次のようにまとめることができる。2 命令のカスケーディングであれば、ほぼ全ての命令がグループ化される命令を見つけることが出来、後続命令へのフォワーディングが遅れる影響を無視できれば、命令スループットを維持できる可能性がある。しかし一方で、64 命令以内にグループ化の対象命令を見つけることの出来ない命令が、全体の 2 割に達する。このことは、デコード・ディスパッチ時あるいはイシュー待機時にグループ化を行うことが困難であることを意味している。将来同じ命令が出現する時に備えてコミット後にグループ化を実施する、などの検討が必要である。その際には履歴の利用が可能である。

続いて、プログラム実行時の調査結果を紹介する。図 7 に、命令ウィンドウの大きさに対して探索範囲 N がどの程度の値となるかを示す。各命令に対して、命令ウィンドウ中に後方に位置する命令がいくつあるかを数えあげ、その平均を求めている。プログラムの違いによる影響が大きく、gcc や perl では大きな命令ウィンドウが有意義に利用されていないことがわかる。

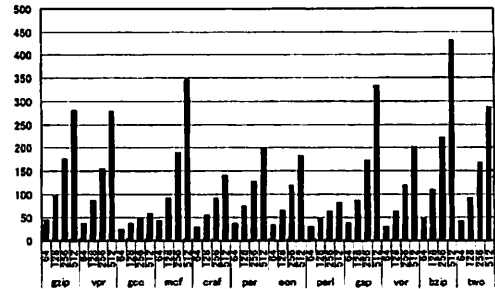


図 7：後続命令数の平均

図 8 は図 4 に相当し、各生産者の後続 N 命令中に存在する消費者数の分布を示している。消費者の探索範囲が限定されるため当然の結果であるが、消費者を見つけることの出来ない生産者の割合が増加している。約 3 割の生産者には消費者を見つけることができない。これはグループ化できる命令数の減少を意味しており、命令スループットの低下が危惧される。

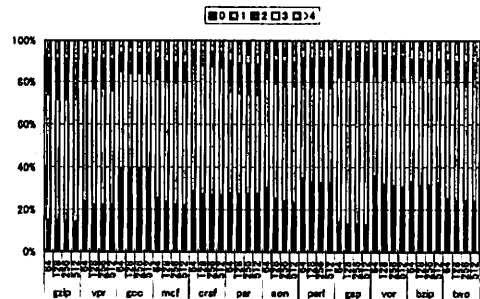
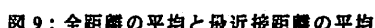


図 8：依存命令数の分布

図 9 は図 5 に相当する。依然として全距離の平均はプログラムの違いや命令ウィンドウサイズに大きく影響されるが、その絶対値は著しく小さくなっている。大きく離れた消費者を見つけるのでできない生産者が増加していることに起因すると思われる。一方で最近接距離の平均には、それらの影響に加えて探索命令数が縮小する影響が小さい。全プログラムの平均で、

た。今回の調査では命令スループットの維持が困難であることが明らかとなり、命令レベル並列性を向上する何らかのサポートが必要であることが判明した。



謝辭

本研究の一部は、科学研究費補助金 (No.16300019) の援助によるものです。

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] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, Parameter variations and impact on circuits and microarchitecture, 40th Design Automation Conference (2003)
- [3] K. A. Bowman, S. G. Duvall, and J. M. Meindl, Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration, *IEEE Journal of Solid-State Circuits*, Vol.37, No.2 (2002)
- [4] J. A. Butts and G. S. Sohi, Characterizing and predicting value degree of use, 35th International Symposium on Microarchitecture (2002)
- [5] M. Franklin and G. S. Sohi, Register traffic analysis for streamlining inter-operation communication in fine-grain parallel processors, 25th International Symposium on Microarchitecture (1992)
- [6] M. Hashimoto and H. Onodera, Increase in delay uncertainty by performance optimization, International Symposium on Circuits and Systems (2001)
- [7] T. Karnik, S. Borkar, and V. De, Sub-90nm technologies: challenges and opportunities for CAD, International Conference on Computer Aided Design (2002)
- [8] I. Kim and M. H. Lipasti, Macro-op scheduling: relaxing scheduling loop constraints, 36th International Symposium on Microarchitecture (2003)
- [9] T. Sato and A. Chiyonobu, Multiple clustered core processors, 13th Workshop on Synthesis and System Integration of Mixed Information Technologies (2006)
- [10] X. Tang, V. K. De, and J. D. Meindl, Intrinsic MOSFET Parameter Fluctuations Due to Random Dopant Placement, *IEEE Transactions on VLSI SYSTEMS*, VOL.5, NO.4 (1997)
- [11] 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, 情報処理2006-ARC-168 (2006)
- [12] 佐藤, 千代延, 命令カスケードイングにおける典型的バズ延の効用, 信学技報 CPSY2005-36 (2005)
- [13] 渡辺, 千代延, 佐藤, スーパースカラプロセッサにおける命令間依存関係の特徴調査, 第13回電子情報通信学会九州支部学生会講演会 (2005)

6. まとめ

マイクロアーキテクチャのレベルでばらつきに配慮出来る命令レベル逐次プロセッサの可能性を検討し