発行時間差に基づいた命令ステアリング方式

服部直也[†,☆] 高田正法[†] 岡部淳[†]
入江英嗣[†,☆☆] 坂井修一[†] 田中英彦[†,☆☆☆]

マイクロプロセッサの高クロック動作と高 IPC を両立させるために，小規模な演算資源の塊を間接的に接続したクラスタアーキテクチャが提案されている．このアーキテクチャでは，命令発行に関する遅延を軽減するために，『最も発行遅延が少ないクラスタ』を特定し，そこへ命令をステアリングすることが望ましい．しかし，そのようなクラスタを厳密に求めるには複雑な計算を要するため，これまでに提案された命令ステアリング方式は，種々の近似を用いてクラスタを選択していた．本論文では，従来方式が用いている近似の精度を解析し，命令の集中/分散の判断に改善の余地があることを明らかにする．そしてこの判断を改善するために，データ生成命令と消費命令の命令間距離を用いる方式を提案する．シミュレータを用いた評価の結果，提案方式は従来方式に比べて，OOO 発行構成に対しては 9.2%，FIFO 発行構成に対しては 5.0% 性能が向上することを確認した．

Instruction Steering Algorithms Based on Issue Delay

Naoya Hattori,[†,☆] Masanori Takada,[†] Jun Okabe,[†]
Hidetsugu Irie,[†,☆☆] Shuichi Sakai[†] and Hidehiko Tanaka[†,☆☆☆]

To achieve both high clock rate and high IPC of microprocessors, clustered architecture has been proposed. In this architecture, instruction should be steered into “the cluster with minimum Issue Delay”. But to detect such a cluster causes too complex calculation. Therefore, prior instruction steering algorithms use some approximations. This paper analyzes the precision of such approximations, and finds the room for improvement at concentration/distribution heuristics. To improve it, Local Distance Steering—a more precise approximation algorithm using Instruction Distance—is proposed. The evaluation by simulation shows that Local Distance Steering improves on performance 9.2% with Out-of-Order Issue architectures, 5.0% with FIFO Issue architectures.

1. はじめに

近年プロセッサの性能は目覚ましく向上しているが，高性能化への要求はとどまるところを知らない．近年のプロセッサは主に，パイプラインを深く設計する Deeper Pipeline 技術と，半導体技術の微細化による高クロック化によって性能を向上させてきた．プロセッサの性能は動作クロックと IPC（Instructions Per Cycle）の積で求められるため，性能向上のためには両者のバランス良い改善が望ましい．しかし実際には，動作クロックを向上させると IPC は低下する傾向にある．

パイプラインを深くすると各処理に必要なサイクル数が増加し，依存関係にある命令の発行間隔が大きくなるために，IPC が低下する[5),14)]．また半導体技術の微細化に関しては，今後ゲート遅延に対して配線遅延が支配的になることが知られている．アーキテクチャ設計者は，処理遅延の増加を容認するか，演算資源を減らして遅延増加を抑えるかの選択を迫られるが，いずれにしても IPC が低下すると考えられている[1)]．

IPC の低下要因は，制御依存による発行間隔の拡大，レジスタデータ依存による発行間隔の拡大，メモリデータ依存による発行間隔の拡大，の 3 種に大別されるが，最も影響が大きいのはレジスタデータ依存である[5),14)]．そこでレジスタデータ依存に関係する機構として，演算器間のデータフォワーディング機構と命令発行機構（IQ : Issue Queue）に関する配線遅延が

† 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
☆ 現在，日立製作所中央研究所
Presently with Hitachi, Ltd., Central Research Laboratory
☆☆ 現在，科学技術振興機構
Presently with Japan Science and Technology Agency
☆☆☆ 現在，情報セキュリティ大学院大学
Presently with Institute of Information Security

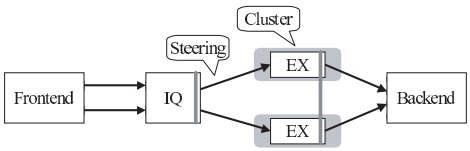

図 1 Execution-Driven アーキテクチャ
Fig. 1 Execution-Driven Steering Architecture.

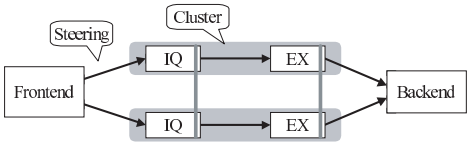

図 2 Dispatch-Driven アーキテクチャ
Fig. 2 Dispatch-Driven Steering Architecture.

重要視されており，これまでに様々な研究がなされている[4),5),8)~10),13),15)]．本研究ではその中の，クラスタアーキテクチャに着目する．

Alpha 21264[6),7)] に代表されるクラスタアーキテクチャでは，演算資源をクラスタリングすることで IPC 低下問題を改善している．IPC に大きな影響を与える，演算器間のデータフォワーディング遅延に関しては，少数の演算器の塊（クラスタ）を形成することで対処する．クラスタ内の演算器は少数であるため，フォワーディングの距離を短くすることが可能であり，遅延を抑えられる．しかし，個々のクラスタのスループットは小さい．そこで複数のクラスタを用意して，それらを間接的に接続することで，プロセッサ全体のスループットを確保する．

命令発行時には，後続命令のオペランドが揃ったこと（Wakeup）の判断のために，全 IQ エントリに発行された命令のタグを送信するが，この命令タグ転送遅延も IPC に与える影響が大きい．そこで，IQ に対してもデータフォワーディングと同様に，クラスタ化することが有用である．**図 1** のように演算器のみをクラスタ化したアーキテクチャは Execution-Driven Steering Microarchitecture[10)]，**図 2** のように命令発行機構までクラスタ化したアーキテクチャは Dispatch-Driven Steering Microarchitecture[10)] と呼ばれている．なお，図中の縦線はクラスタ間を結ぶネットワークである．

本研究では，文献 4)，8) 等，多くの文献で支持されている，Dispatch-Driven アーキテクチャに着目する．以下では特に断らない限り，Dispatch-Driven を前提とする．また，クラスタ間ネットワークに関しては最も理解しやすいクロスバトポロジを前提とする．

Dispatch-Driven では命令をクラスタに割り当てる，命令ステアリングの質がその性能を左右する．クラスタ内の高速な信号転送を利用するためには，データ依存関係にある命令を同じクラスタへ集中して割り当てることが必要である．しかしプロセッサのスループットを利用するためには，命令を適度に分散させる必要がある．したがって命令を滞りなく処理し続けるためには命令ステアリングの際に，何らかの方法で集中と分散のバランスをとらなければならない．

命令ステアリングの理想は，データ生成命令と消費命令の『発行時間差』を短縮するために,『最も早く発行可能なクラスタ』を選択することである．これまでに提案されたステアリング方式は，発行時間差を近似的に求めてクラスタを選択していたが，その近似精度に問題があった．

本論文では『発行時間差』を明確に意識した高精度な近似方式として,『命令間距離』に着目した Local Distance Steering と Global Distance Steering を提案，評価する．

本論文は以下のように構成される．2 章では命令ステアリングとクラスタアーキテクチャの性能に関して検討し，命令の発行時刻情報を利用した，理想的な命令ステアリング方式を提示する．3 章では命令ステアリングに関する先行研究を紹介し，理想命令ステアリングを近似する方法とその不備を検討する．4 章では本論文で想定するアーキテクチャと評価環境について述べる．5 章で先行ステアリング方式の性能と近似の妥当性を解析し，それを受けて 6 章ではより優れた近似方式を提案，評価する．7 章では提案手法に対して閾値，クラスタ構成を変えた場合の評価を行い，8 章では異なるクラスタアーキテクチャとの比較を行う．最後に 9 章で全体をまとめる．

2. 命令ステアリング

命令ステアリングは，処理レイテンシとスループットという 2 つの観点から，クラスタアーキテクチャの IPC に影響を与える．命令発行には Wakeup と Select の 2 段階のプロセスが存在するが，両者はこのプロセスそれぞれに対応している．

クラスタ間通信による Wakeup 遅延

本論文では，データを生成する命令（Producer）が発行されてからデータを消費する命令（Consumer）が Wakeup するまでの時間差を Wakeup 遅延（または Δ Wakeup）と呼ぶ．Producer と Consumer を異なるクラスタにステアリングする場合は，クラスタ間の命令タグフォワーディングが通信遅延の分だけ遅れることになる．その様子を **図 3** に示す．図中の W

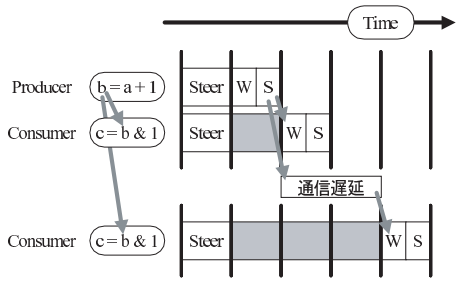

図 3 命令ステアリングと Wakeup 遅延の関係
Fig. 3 Relationship between Instruction Steering and Wakeup Delay.

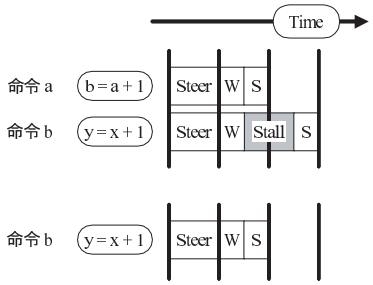

図 4 命令ステアリングと Select 遅延の関係
Fig. 4 Relationship between Instruction Steering and Select Delay.

は Wakeup，S は Select の略で，縦線はクロックを意味する．また，矢印はデータ依存関係を示している．Producer と Consumer を同じクラスタに割り当てる（上段）場合は，Consumer は次のサイクルで Wakeup するが，異なるクラスタに割り当てる（下段）とクラスタ間通信遅延と等しい Wakeup 遅延が発生する．

命令集中による Select 遅延

命令が Wakeup してから Select されるまでの時間差を，本論文では Select 遅延（または Δ Select）と呼ぶ．クラスタアーキテクチャは一般に，個々のクラスタの命令発行幅が狭い．そのため，同時に Wakeup する複数の命令を同クラスタにステアリングする場合，発行幅を超える後続命令には Select 遅延が発生する．この様子を **図 4** に示す．依存関係のない命令 a，命令 b を同じクラスタに割り当てる（上段）と，発行資源が利用可能になるまで命令 b は Stall する．これに対して，命令 b を異なるクラスタに割り当てる（下段）場合は命令 a と同時に発行可能である．

Wakeup 遅延と Select 遅延のバランス

命令を速やかに発行するためには，Producer と Consumer の『発行時間差』（または Δ Issue）を短縮することが必要である．『発行時間差』は前述の定義より，Wakeup 遅延と Select 遅延の総和で求められる．Wakeup 遅延に関してはデータ依存命令を特定クラスタへ集中させることで，Select 遅延に関しては命令を多数のクラスタへ分散させることで軽減できる．しかし，一般に実プログラム中の命令列には複雑なデータ依存関係があるため，Wakeup 遅延の改善は Select 遅延を招き，Select 遅延の改善は Wakeup 遅延を招く．したがって命令発行の遅延を改善するためには，両遅延のバランスをとることが必要である．つまり，Wakeup 遅延や Select 遅延といった個々の要素だけにとらわれず，『最も早く発行可能なクラスタ』を選ぶことが命令ステアリングの本質である．

究極的には，正確な命令発行時刻を計算してクラスタを選択することができれば，理想的な性能が得られると考えられる．すべてのクラスタの中から『最も早く発行可能な』クラスタを選択することから，本論文ではそのような方式を min(ALL) と呼ぶ．

しかし命令発行時刻の計算，特に Select 遅延の計算には，命令発行を行うのと同程度以上の複雑性と遅延を要するため，現実的ではない．そのため，これまでに近似的な手法が提案されてきた．

3. 関 連 研 究

Wakeup 遅延に関しては，オペランド数が多いほど計算が複雑になる．しかし大多数の命令は，実行時に未解決であるオペランドは 1 つ以下であることが知られている[2),9)] ため，本論文では特に断らない限り複数オペランドに関する説明は省略する．通信遅延が固定時間となるクロスバトポロジの採用を仮定すると，Consumer が Producer と同じクラスタにある場合は Wakeup 遅延が 0 になり，それ以外のクラスタにある場合は固定サイクル（C）の通信遅延が発生する．Wakeup 遅延は最大でも 2 値しかとりえないので，発行時間差（= Wakeup 遅延 + Select 遅延）が最も小さいクラスタは，以下のいずれかの条件を満たす．

- 0 入力命令の場合：
  Select 遅延が最小のクラスタ
- 1 入力命令の場合：
  Wakeup 遅延が 0 である唯一のクラスタ，または，
  Select 遅延が最小のクラスタ

Wakeup 遅延が 0 であるクラスタは比較的容易に特定可能であるが，Select 遅延が最小となるクラスタを求めることは一般に難しい．そのため多くの命令ステアリング方式では，Wakeup 遅延が 0 であるクラスタと，近似的に求めた Select 遅延が最小とおぼしきクラスタの中から，ヒューリスティックに従って一方を選択している．

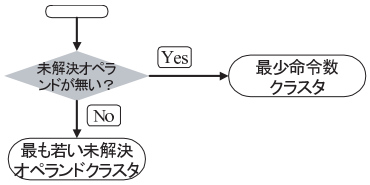

図 5 Dependence Based Steering のアルゴリズム
Fig. 5 Dependence Based Steering algorithm.

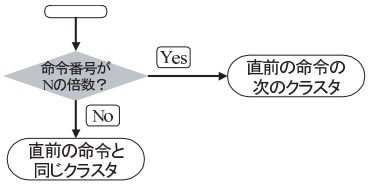

図 6 Modulo Steering のアルゴリズム
Fig. 6 Modulo Steering algorithm.

3.1 Wakeup 遅延重視の Dependence Based Steering

Wakeup 遅延の回避に特化した単純な命令ステアリングとして，**図 5** に示す Dependence Based Steering が考案されている[3]．この方式では Select 遅延が十分に小さいと仮定し，対象命令に未解決のオペランドが存在する場合はつねに未解決オペランドと同じクラスタ（以下 OP-クラスタと略記：OP は Operand Producer の略）へのステアリングを行う．未解決オペランドが存在しない場合は，最低負荷クラスタ（以下 LW-クラスタと略記：LW は Lowest Workload の略）の Select 遅延が最小であるという近似に基づいてステアリングを行う．この方式は Select 遅延が小さいことを仮定しているため，クラスタの発行幅が狭い場合のように Select 遅延の影響が大きい構成には不向きである．

3.2 Select 遅延重視の Modulo Steering

Select 遅延の回避に特化した単純な命令ステアリングとして，Modulo Steering が考案されている[3]．この方式では Wakeup 遅延が十分小さいという仮定と，累計ステアリング命令数が少ないクラスタは Select 遅延が小さいという仮定の下に，命令を Round-Robin にステアリングする．

この方式は Wakeup 遅延が無視できないアーキテクチャには不向きであるが，その点を改良した Modulo-N Steering も提案されている．Modulo-N Steering では連続する命令には依存関係があると仮定し，連続する N 命令ずつの塊を Round-Robin にステアリングすることで，Wakeup 遅延の影響を軽減する（**図 6**）．なお，Modulo Steering は $N = 1$ の場合の Modulo-N Steering と等価であるため，以下では両者を区別なく，Modulo Steering と呼ぶ．

この方式は発行時間差の計算に多数の近似を用いているため，近似精度が低下しやすいことが問題である．

3.3 Out-of-Order 発行向けの Parcerisa 方式

Wakeup 遅延と Select 遅延の双方の影響を避けるために，Parcerisa らはクラスタ間の負荷バランスに着目した方式を提案している[11),12)]（以後，本論文中では Parcerisa 方式と呼ぶ）．そのアルゴリズムを **図 7** に示す．この方式では，負荷が均衡している場合はどのクラスタも Select 遅延が同程度であると判断して OP-クラスタへステアリングする．逆に負荷集中が発生している場合は Select 遅延が最も小さいと予想される LW-クラスタへステアリングする．

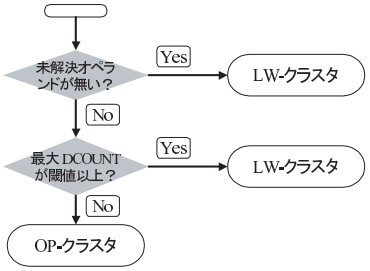

図 7 Parcerisa らのアルゴリズム
Fig. 7 Parcerisa's algorithm.

彼らは負荷の均衡状態を把握するために，DCOUNT という指標を採用している．DCOUNT は各クラスタの状態を示す符号付整数値で，命令がそのクラスタにステアリングされた場合に [クラスタ数–1] だけ増加し，他のクラスタにステアリングされた場合に 1 だけ減少する．負荷状態を判断する際には，最大の DCOUNT 値が閾値よりも大きければ負荷集中と見なし，小さければ負荷は均衡していると見なす．

この方式では Select 遅延の推定に，Consumer を Wakeup する時点での負荷ではなく，Consumer をステアリングする時点での負荷を用いている．したがってステアリングと Wakeup に時間差が大きい場合等，Wakeup 時の負荷状態とステアリング時の負荷状態が異なる命令に対しては，この方式では適切な判断ができない．

3.4 FIFO 発行向けの Palacharla 方式

Palacharla らは文献 10) で，マイクロプロセッサを高速化するうえで，命令発行の遅延が問題になっていることを指摘し，この遅延を短縮するために FIFO 発行方式に着目している．この方式では，FIFO 先頭の命令のみを発行可能とすることで，命令タグフォワーディング配線を短縮し，高速化することができる．し

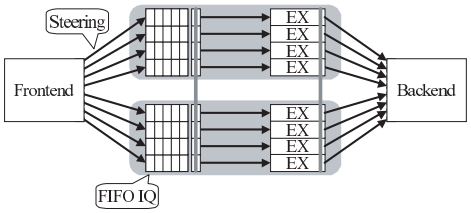

図 8 Palacharla らの FIFO 発行クラスタアーキテクチャ
Fig. 8 Palacharla's FIFO issue clustered architecture.

かし，命令発行に制限が課せられているため，各命令に対して FIFO を選択する，命令ステアリングが重要になる．彼らは独自の命令ステアリングを提案している（以後，本論文中では Palacharla 方式と呼ぶ）．文献 10) で提案されているアーキテクチャは，**図 8** に示すように，4 つの FIFO を搭載した 2-cluster から構成される．また，図中の縦線で示すように命令タグやデータをフォワーディングする，クラスタ間ネットワークが存在する．Palacharla 方式は単一クラスタ内の FIFO 選択方法を拡張された形で考案されている．

単一クラスタ内の FIFO に対しては通信遅延が存在しないため，Wakeup 遅延はつねに一定である．そのため命令ステアリングには，Select 遅延を可能な限り避けることが要求される．一般にはそのようなクラスタを特定することは難しいが，特定の条件下において簡単に求めることができる．

(1) 無負荷 FIFO（以下 NW-FIFO と略記：NW は No Workload の略）が存在する場合：
NW-FIFO へステアリングすれば，Consumer はつねに FIFO の先頭に位置する．

(2) 未解決オペランドの Producer が FIFO の末尾にある場合：
Producer と同じ FIFO（以下 OP-FIFO と略記）にステアリングすれば，Producer の発行直後に Consumer が OP-FIFO の先頭に位置する．

Palacharla 方式ではこれらの性質を受けて，条件 (2) を満たす FIFO があればその FIFO（OP-FIFO）へ，なければ条件 (1) を満たす NW-FIFO へステアリングする．NW-FIFO が不足した場合には，いずれかの FIFO が空になるまでステアリングステージ以前のパイプラインを stall させる．逆に NW-FIFO が複数存在する場合は，直前の命令と同じクラスタに属する FIFO を優先することで，クラスタ間通信の影響を軽減する．この方式のアルゴリズムフローチャートを**図 9** に示す．

この方式では，Select 遅延の有無のみに着目している．しかし，正確には Wakeup 遅延と Select 遅延の和が最小になるクラスタを選択すべきであり，そのような選択のためには，Select 遅延を数値として算出する必要がある．

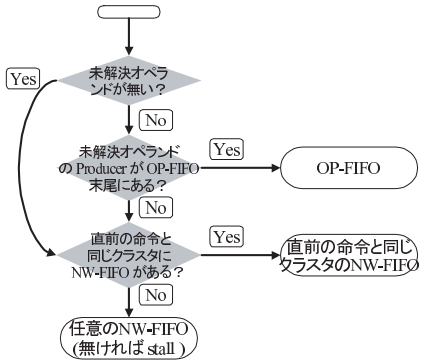

図 9 Palacharla 方式のアルゴリズム
Fig. 9 Palacharla's algorithm.

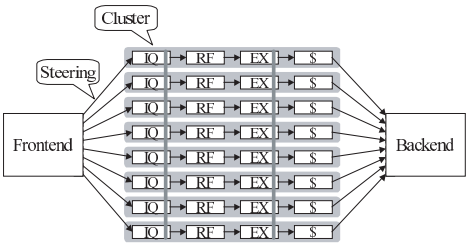

図 10 想定するアーキテクチャの構成
Fig. 10 Target architecture.

4. 本研究で想定するプロセッサ構成と評価環境

4.1 本研究で想定するプロセッサ構成

本研究で想定するアーキテクチャの構成を**図 10** に示す．想定するアーキテクチャは Frontend，クラスタ部，Backend から構成される．Frontend では命令のフェッチ，デコード，レジスタリネーミングを行い，分散した IQ への命令ステアリングを行う．IQ からキャッシュまではクラスタに含まれており，レジスタとキャッシュはそれぞれが完全な複製を保持する．Backend は主にリタイア処理を行う．なお，本論文では IQ の発行方式として，Out-of-Order（OOO）発行と FIFO 発行の 2 方式を評価の対象とする．

4.2 基本評価環境

以下で評価に用いるシミュレータの基本設定を**表 1** に示す．1 並列のクラスタ 8 つからなる構成とし，キャッシュやメモリ依存予測は理想化した．また，各クラスタの演算器は完全にパイプライン化されていると仮定

表 1 シミュレータの基本設定
Table 1 Baseline simulator configuration.

Total Pipeline Depth	20 stages
Issue-to-Wakeup	1 cycle
D1 Access	2 cycles
L2 Access	12 cycles
クラスタ間通信遅延	2 cycles
整数演算遅延	1 cycle
整数乗算遅延	15 cycles
浮動小数点演算遅延	4 cycles
分岐予測	Gshare（64 k エントリ）
メモリ依存予測	理想化
命令キャッシュ	100% hit
データキャッシュ	100% hit
各クラスタの発行幅と総数	1-issue × 8-cluster
クラスタ間ネットワークトポロジ	クロスバ
各 IQ のエントリ数	32
Active List の大きさ	256
命令発行方式	OOO or FIFO
フェッチ幅	8
リタイア幅	8
演算器の機能	全種類の命令を処理可能
命令セット	Alpha 21264
測定命令数	256 M 命令

し，それぞれ Alpha 21264 のすべての種類の演算を実行できると仮定した．分岐ミスペナルティとなる総パイプライン段数は，全部で 20 段とし，データ依存のある命令の発行間隔（Issue-to-Wakeup 遅延）は最短で 1 サイクルとした．クラスタ間の通信遅延に関しては 2 サイクルと仮定して評価を行った．

なお Sensitivity の評価のために，クラスタ間通信遅延，クラスタの並列度，キャッシュに関しては，それぞれ変化させたデータも測定する．

性能評価には SPECint95 の train 10 種をすべて使用し，これらのアプリケーションを 256 M 命令動作させ，平均 CPI（Clocks Per Instruction：IPC の逆数）を求めた．バイナリは OSF1 上で gcc 2.95.2 を用いて作成した．また測定にはトレースシミュレータを用いた．

5. 先行命令ステアリング方式の性能

Wakeup 遅延が最小となるクラスタは OP-クラスタであるが，Select 遅延が最小となるクラスタを正確に特定することは難しい．そこで，Select 遅延最小クラスタの近似として，従来方式が用いている LW-クラスタや NW-クラスタを用いる近似の精度を調べるために，min(OP, LW)，min(OP, NW) という項目も用意した．これらは min(ALL) がすべてのクラスタからΔ Delay が最小であるクラスタを選択するのに対し，OP-クラスタと LW-クラスタまたは NW-クラスタの中だけからΔ Issue が最小であるクラスタを選択する．

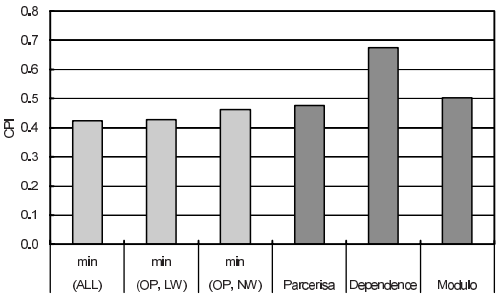

図 11 従来方式の CPI 比較（OOO 発行）
Fig. 11 Performance analysis of prior steering algorithms (OOO issue).

5.1 OOO 発行に対する先行方式の性能解析

OOO 発行の想定アーキテクチャに対して，各種ステアリング方式を用いた場合の CPI を**図 11** に示す．

まず理想的な方式に着目すると，min(OP, LW) は min(OP, NW) よりも CPI が小さく，min(ALL) との差がほとんど見受けられない．これは，Select 遅延が最小となるクラスタとして LW-クラスタを選択する近似の妥当性を示唆している．

また従来方式に関しては，Parcerisa 方式の CPI が最も小さい．これは Parcerisa 方式で用いている近似が，他方式の近似よりも精度が高いことを示している．

ここで，min(OP, LW) と Parcerisa 方式に着目すると，両者は Select 遅延が最小となるクラスタとして LW-クラスタを選択している点で共通しており，OP-クラスタと LW-クラスタの一方を選択する判断基準のみが異なる．したがって両者の CPI の差は，Parcerisa 方式のクラスタ選択の判断基準が不正確であることに起因する．つまり，Parcerisa 方式はクラスタの判断基準に改善の余地が残っていることが確認された．

5.2 FIFO 発行に対する先行方式の性能解析

同様に，FIFO 発行の想定アーキテクチャに対して，各種ステアリング方式を用いた場合の CPI を**図 12** に示す．

FIFO 発行の場合は，min(OP, NW) の方が min(OP, LW) よりも CPI が小さく，min(ALL) との差がほとんど見受けられない．これは，Select 遅延が最小となるクラスタとして NW-クラスタを選択する近似の妥当性を示唆している．

従来方式に関しては，Palacharla 方式の CPI が最も小さい．これは Palacharla 方式で用いている近似が，他方式の近似よりも精度が高いことを示している．

OOO 発行の場合と同様に，min(OP, NW) と Palacharla 方式の CPI の差は，Placharla 方式のクラ

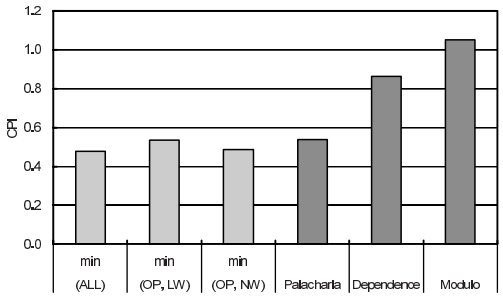

図 12 従来方式の CPI 比較（FIFO 発行）
Fig. 12 Performance analysis of prior steering algorithms (FIFO issue).

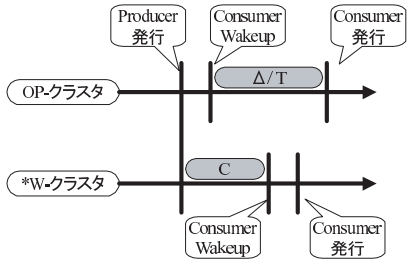

図 13 発行時間差の近似計算
Fig. 13 Issue delay approximation.

スタ選択の判断基準が不正確であることに起因する．つまり，Palacharla 方式はクラスタの判断基準に改善の余地が残っていることが確認された．

6. 依存命令の発行時間差に基づいた命令ステアリング方式の提案と評価

関連研究におけるクラスタ選択の不正確性を改善するために，本研究では,『最も早く発行可能なクラスタ』の高精度な推定に着目した．以下で説明する推定方式の概要を**図 13** に示す．この図は OP-クラスタ，*W-クラスタ（LW-クラスタと NW-クラスタの総称）に Consumer 命令をステアリングする場合の，Producer の発行時刻，Consumer の Wakeup 時刻および発行時刻を順に示している．

Producer が発行してから Consumer が発行するまでの時間差（IssueDelay）は，次の式で求められる．

$$IssueDelay = WakeupDelay + SelectDelay$$

ここで OP-クラスタ内の実際の命令発行スループットを T，OP-クラスタにおける Producer と Consumer の間の命令数を Δ とすると，OP-クラスタにおける Select 遅延は Δ / T である．また，*W-クラスタにおける Select 遅延は小さいと考えられるため，0 に近似する．よって，通信遅延を C とすると以下の近似式が成り立つ．

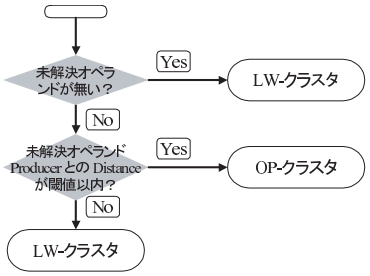

図 14 Out-of-Order 発行向け提案手法のアルゴリズム
Fig. 14 Proposal algorithm for OOO issue.

$$WakeupDelay(OP) = 0$$
$$WakeupDelay(*W) = C$$
$$SelectDelay(OP) = \Delta/T$$
$$SelectDelay(*W) = 0$$

したがって，両クラスタの発行時間差は以下の式で求められる．

$$IssueDelay(OP) = \Delta/T$$
$$IssueDelay(*W) = C$$

すると『最も早く発行可能なクラスタ』は Δ/T と C の大小比較，つまり Δ と C×T の大小比較で決定できる．

Δ を正確に算出する方法としては，ステアリングされた命令に対して，クラスタごとに通し番号を付けることが考えられる．この通し番号の差を Local Distance と呼ぶ．

しかしクラスタごとに命令の通し番号を付けるには，番号の生成や記憶等の管理ハードウェアが必要になるため，複雑性が増加する可能性がある．このような複雑性の増加を避けたい場合には，クラスタごとの通し番号を，アクティブリスト中の間の命令数で代用することが考えられる．この命令タグの差を Global Distance と呼び，その値は Δ Select に比例すると見なす．

C×T に関しては，実効スループットを求めることも考えられるが，本研究では簡単のために適当な定数を設定することにした．

したがって，先の比較計算は，Local（Global）Distance と閾値（定数）の比較に帰着された．

Local Distance や Global Distance を判断基準とする命令ステアリングは，OOO 発行向け，FIFO 発行向け命令ステアリングそれぞれの枠組みである，min(OP, LW)，min(OP, NW) に従って**図 14**，**図 15** の形になる．

6.1 評　　価

表 1 で示した基本構成における各種方式の CPI を，**図 16**，**図 17** に示す．それぞれ OOO 発行，FIFO 発

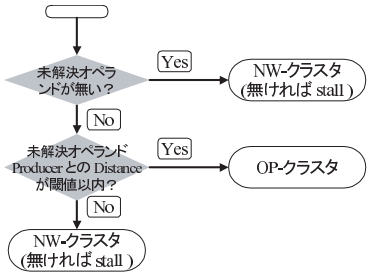

図 15 FIFO 発行向け提案手法のアルゴリズム
Fig. 15 Proposal algorithm for FIFO issue.

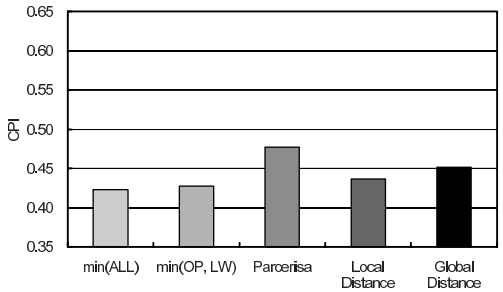

図 16 従来方式と提案方式の CPI 比較（OOO 発行）
Fig. 16 Performance comparison of prior and proposed steering algorithms (OOO issue).

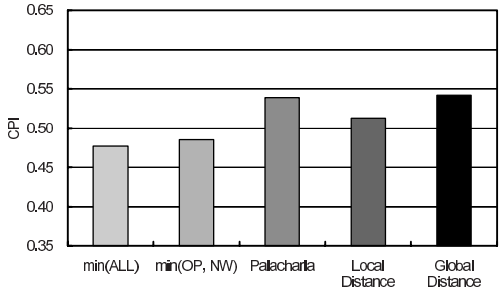

図 17 従来方式と提案方式の CPI 比較（FIFO 発行）
Fig. 17 Performance comparison of prior and proposed steering algorithms (FIFO issue).

行の場合の CPI を示している．

提案方式 Local Distance Steering は OOO 発行では 9.2%，FIFO 発行では 5.0%，従来方式よりも CPI が改善されている．これは Local Distance Steering が Select 遅延の推定に用いている近似の精度が，各従来方式が用いている近似の精度よりも高いためである．特に OOO 発行の場合は，Local Distance Steering と理想方式 min(ALL) との性能差が 3.3%ときわめて小さく，優れた近似精度であることがうかがえる．

Global Distance Steering に関しては，OOO 発行の場合は Parcerisa 方式よりも CPI が 5.7%改善されているが，FIFO 発行の場合は Palacharla 方式と比べて 0.6%とわずかながら CPI が劣化している．これは Global Distance Steering が用いている，Δ Select(OP) が Global Distance に比例するという近似の精度が，Parcerisa 方式が用いている近似の精度より高く，Palacharla 方式が用いている近似の精度より低いことを示している．

7. Sensitivity Analysis

7.1 提案手法の閾値依存性

次に，基本設定における，各アプリケーション・閾値ごとの CPI を測定した．その結果を**図 18**，**図 19** に示す．左側の 5 つの棒は Local Distance（LD）の閾値 0〜4 に対応しており，右側の 5 つの棒は Global Distance（GD）の閾値 2〜6 に対応している．

OOO 発行の場合は各アプリケーションとも，平均と同様緩やかな CPI 推移を示しており，閾値を LD02, GD04 前後に設定しておけばおおむね最適な設定になっている．しかし FIFO 発行の場合には，急激な CPI 変化を示すアプリケーションが存在する．

Local/Global Distance は閾値が無限であれば Dependence Based と同じ動作をする．その場合は，**図 20** のように，出力が複数回使用される命令を含むデータフローが特定のクラスタに集中し，Select 遅延のために性能が低下する．閾値を下げれば，そのようなデータフローが分割されやすくなり，Wakeup 遅延が増加する代わりに Select 遅延が軽減される．m88ksim や perl-p のように，閾値による急激な CPI 変化を示すアプリケーションは，図 20 のようなデータフローの実行回数が多いためであると考えられる．実際 m88ksim に関しては,『出力が複数回使用される命令』（図 20 の A）の実行頻度が，他のアプリケーションに比べて顕著に多いことを確認している．

7.2 通信遅延を変化させた場合の評価

基本設定から，通信遅延を 1〜4 の間で変化させながら測定した CPI を，**図 21**，**図 22** に示す．なお，各方式に関して，それぞれ最適な閾値を用いた場合の CPI を掲載した．

通信遅延が大きいほど Δ Wakeup が増加するため，CPI が増加している．ただし，OOO 発行の場合の min(ALL) と min(OP, LW) の差や，FIFO 発行の場合の min(ALL) と min(OP, NW) の差のように，純粋に Δ Select の推定誤差に起因する性能差は，通信遅延によらず一定である．つまり，Δ Wakeup に起因する推定誤差の分だけ CPI が劣化し，各方式の性能差が広がっている．また，提案方式 Local Distance Steering は通信遅延の大小にかかわらず，CPI を削減

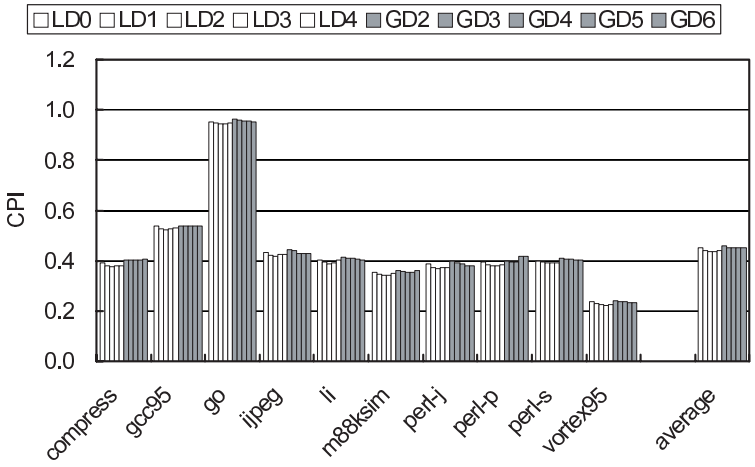

図 18 提案方式の閾値とアプリケーションごとの CPI（OOO 発行）
Fig. 18 Performance of individual applications with different threshold values (OOO issue).

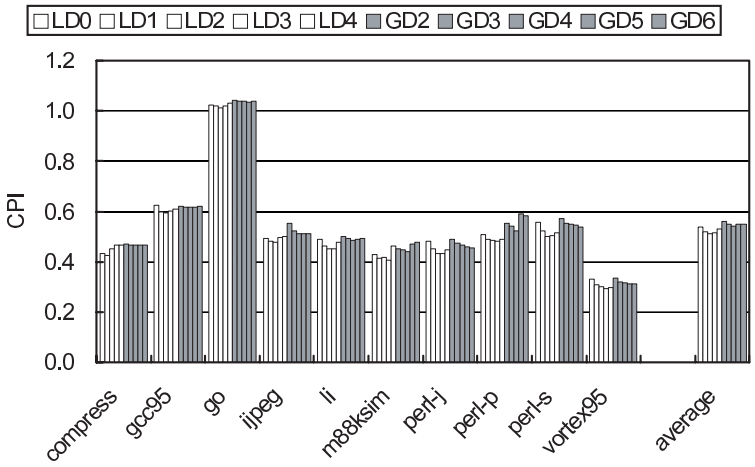

図 19 提案手法の閾値とアプリケーションごとの CPI（FIFO 発行）
Fig. 19 Performance of individual applications with different threshold values (FIFO issue).

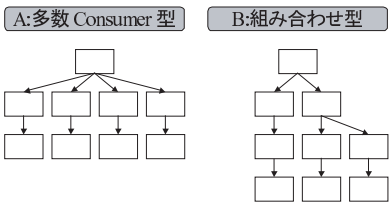

図 20 閾値が問題になりやすいデータフローグラフの例
Fig. 20 Example of threshold sensitive Data Flow Graph.

できている．

基本的に，通信遅延が変化しても各方式間の優劣は変化しないが，FIFO 発行の場合の Palacharla 方式だけは，通信遅延が増加した場合に CPI が大きく劣化し，Global Distance Steering との優劣が逆転している．これは，Palacharla 方式は Δ Wakeup =0 を前提に設計されているため，通信遅延が増加した場合に他方式よりも大きな近似誤差が発生することが原因で

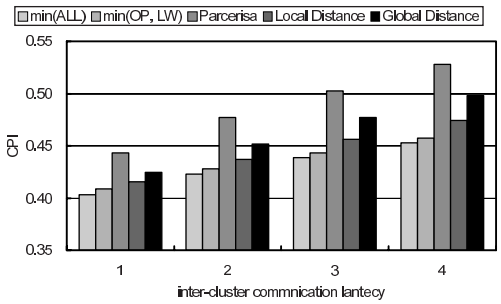

図 21 通信遅延を変化させた場合（OOO 発行）
Fig. 21 Impact of inter-cluster communication latency on CPI (OOO issue).

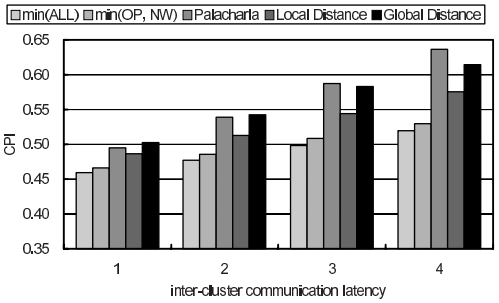

図 22 通信遅延を変化させた場合（FIFO 発行）
Fig. 22 Impact of inter-cluster communication latency on CPI (FIFO issue).

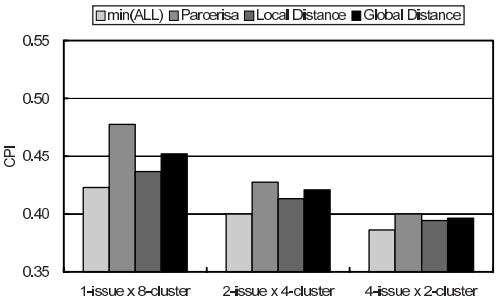

図 23 発行幅を変化させた場合（OOO 発行）
Fig. 23 Impact of issue width per cluster on CPI (OOO issue).

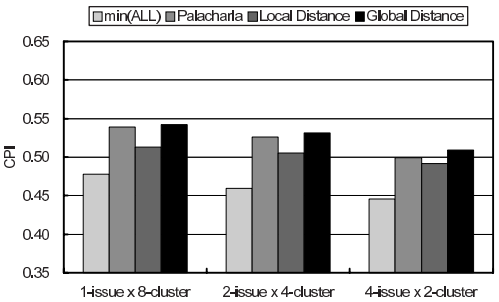

図 24 発行幅を変化させた場合（FIFO 発行）
Fig. 24 Impact of issue width per cluster on CPI (FIFO issue).

ある．

7.3 発行幅を変化させた場合の評価

同様に，発行幅を変化させた場合の CPI を**図 23**，**図 24** に示す．なお，クラスタ全体の発行幅を 8 に固定するために，発行幅を増やした分だけクラスタ数を減少させている．

OOO 発行の場合は，クラスタの並列度が上がるとステアリング方式による CPI 差が小さくなっているが，FIFO 発行の場合は，並列度が上がっても CPI 差があまり小さくならない．これは，OOO 発行の場合

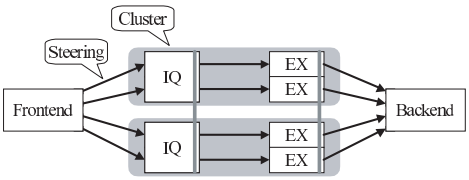

図 25 Dispatch-Driven OOO 発行アーキテクチャ（2 × 2）
Fig. 25 Dispatch-Driven OOO Issue Architecture.

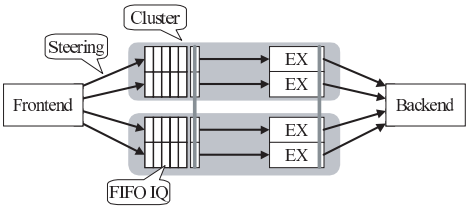

図 26 Dispatch-Driven 複数 FIFO アーキテクチャ（2 × 2）
Fig. 26 Dispatch-Driven FIFO Issue Architecture.

は並列度が上がると**図 25** のように，ステアリングの選択肢が減少するのに対し，FIFO 発行の場合は**図 26** のように並列度にかかわらず，ステアリングの選択肢が一定であることに起因する．つまり，OOO 発行の場合はステアリングの選択肢数が減少するため，運良く最適なクラスタを選択できる可能性が増加していると考えられる．

7.4 現実的なデータキャッシュを用いた場合

基本構成ではデータキャッシュを 100%ヒットとして理想化していたが，Local Distance Steering や Global Distance Steering は OP-クラスタのスループットを固定値で近似しているため，キャッシュミスの影響を受けやすい可能性がある．そこで本節では，現実的なキャッシュパラメータに対する性能を測定した．

4 章で述べたように，本研究では全クラスタに複製されたキャッシュが搭載されていることを仮定している．また，クラスタアーキテクチャの意義から考えて，個々のクラスタに 8 k bytes という小さめのキャッシュ容量を想定した．なお，内容を全クラスタに複製するため，正味のキャッシュ容量はプロセッサ全体でも 8 k bytes となる．その他のパラメータは**表 2** に示す．また，そのパラメータで測定した，各方式の CPI を**図 27**，**図 28** に示す．

理想的な性能を意味する min(ALL)，min(OP, LW)，min(OP, NW) は正確な処理レイテンシを元に算出する必要があるため，キャッシュを理想化しなければ測定できない．そのため，これらの項目は 図 27, 図 28 には掲載していない．それ以外の方式に関しては，OOO 発行，FIFO 発行のいずれも 8 k-byte キャッ

表 2 キャッシュパラメータ
Table 2 Cache parameters.

L1 Data Cache	Capacity Associativity Line Size	8 k bytes 2-way 32 bytes/line
L1 Instruction Cache	100%hit	
L2 Unified Cache	100%hit	

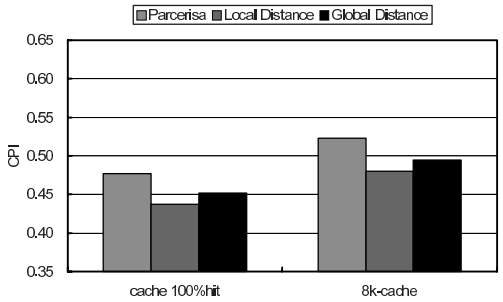

図 27 8 k のデータキャッシュを用いた場合（OOO）
Fig. 27 CPI with 8 KB data cache (OOO issue).

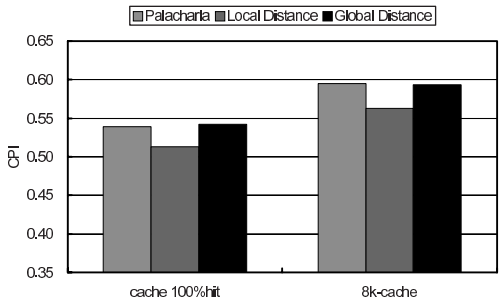

図 28 8 k のデータキャッシュを用いた場合（FIFO）
Fig. 28 CPI with 8 KB data cache (FIFO issue).

シュを用いることにともなう性能劣化はほぼ同程度であり，提案方式 Local Distance Steering が最も高性能であることが確認された．FIFO 発行であってもキャッシュミスの影響を大きく受けない理由としては，各方式とも基本的にはデータ依存に基づいてステアリングされていることが考えられる．つまり，キャッシュミスを起こしたロード命令とその後続命令が，同じクラスタに隣接してステアリングされることが多いために，キャッシュミスの影響は軽減されていると考えられる．

8. 異なるクラスタアーキテクチャとの比較

冒頭でも述べたように，クラスタアーキテクチャには Dispatch-Driven のほかに Execution-Driven という構成が存在する．本章では，Execution-Driven 構成と提案方式を用いた Dispatch-Driven 構成の比較を行う．

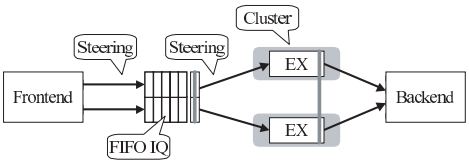

図 29 小林らのアーキテクチャ
Fig. 29 Kobayashi's architecture.

Dispatch-Driven の特徴

Dispatch-Driven では，IQ がクラスタ化されているため，Wakeup 遅延と Select 遅延の総和を小さくするステアリングが必要である．

Execution-Driven の特徴

それに対して Execution-Driven では IQ がクラスタ化されていないため，Δ Select =0 と見なしてよい．そのため Wakeup 遅延だけを小さくするステアリングを行えばよく，Dispatch-Driven よりも発行遅延を軽減することが可能である．つまり，高い IPC が期待できる．したがって，OOO 発行 Execution-Driven は，OOO 発行 Dispatch-Driven の IPC 上限と考えることができる．

しかしその反面，各クラスタに対して利用可能なオペランドを把握する必要があるため，IQ が複雑になる．

小林らのアーキテクチャの特徴

小林らは文献 16) で，Execution-Driven の IPC をさらに向上させるために，データフローグラフの最長パスの命令に着目して，発行待ち命令に優先順位を設けることを提案している．小林らは複数 FIFO からなる IQ と PIT（Path Information Table）を用いて，データフローの最長パスを計算し，優先すべき重要な命令を認識している．したがって，小林らのアーキテクチャは**図 29** のように FIFO 発行の Execution-Driven という構成になる．IQ が複数の FIFO から構成されているため，FIFO に対するステアリングも必要であるが，これには DFG 解析に応用可能な Palacharla 方式を用いている．このアーキテクチャでは，1 度目のステアリングで Select 遅延が決定し，2 度目のステアリングで Wakeup 遅延が決定する．Execution-Driven のステアリングを行う点，優先順位を設ける点，の 2 点のために複雑性が増加するが，Wakeup，Select 遅延を個別に決定できるために発行遅延を軽減しやすく，IPC 向上が期待できる．

したがって，小林らのアーキテクチャは，FIFO 発行 Dispatch-Driven の IPC 上限であると考えられる．

評　　価

各アーキテクチャの CPI 測定に関して**表 3** に示す

表 3 各種アーキテクチャ構成
Table 3 Architecture configurations.

Execution-Driven（OOO）	IQ エントリ数	256
	発行幅	8 命令/cycle
	クラスタ数	8
	クラスタの演算能力	1 命令/cycle
小林らの構成（FIFO）	IQ エントリ数	32/FIFO
	発行幅	8 命令/cycle
	クラスタ数	8
	FIFO 数	16
	クラスタの演算能力	1 命令/cycle
Dispatch-Driven（OOO/FIFO 共通）	IQ エントリ数	32/cluster
	クラスタの発行幅	1 命令/cycle
	クラスタ数	8
	クラスタの演算能力	1 命令/cycle

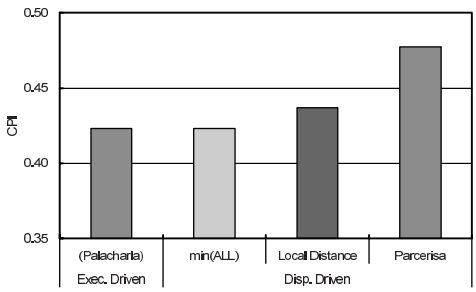

図 30 異なるアーキテクチャとの比較（OOO 発行）
Fig. 30 CPI comparison of various architectures (OOO issue).

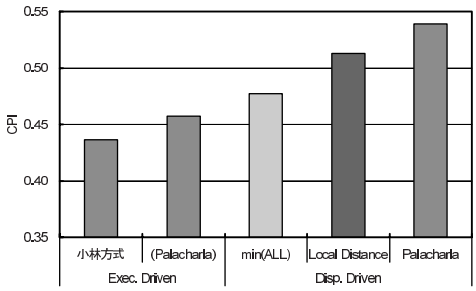

図 31 異なるアーキテクチャとの比較（FIFO 発行）
Fig. 31 CPI comparison of various architectures (FIFO issue).

パラメータを用いた．表中に記載のないパラメータは表 1 の値を用いている．なお，小林らの方式は FIFO 数が少ない構成では本来の性能を発揮できないため，評価には 16 FIFO 構成を用いた．また，小林らの提案方式に加えて，優先順位を設けない方式（FIFO 発行 Execution-Driven）も比較対象とした．各種アーキテクチャに対する CPI を**図 30**，**図 31** に示す．

OOO 発行に関しては，理想方式 min(ALL) を用いることで性能上限である Execution-Driven と遜色のない CPI が得られている．同様に，提案方式 Local Distance Steering も良好な CPI が得られており，Execution-Driven との差は 3.3%である．このことから OOO 発行の場合は，Dispatch-Driven 構成に提案方式を導入することで，複雑な Execution-Driven に近い性能が得られることが確認された．

FIFO 発行に関しては，Execution-Driven に対して小林らの方式を用いることで 4.5% CPI を改善できている．これに対し，Dispatch-Driven の CPI 下限である min(ALL) は 4.4% CPI が増加しており，提案方式 Local Distance Steering は 12% CPI が増加している．このように FIFO 発行の場合は，各方式の性能は複雑性とのトレードオフなっている．したがって提案方式は，小林らのような複雑な方式にはかなわないが，比較的単純な構成を仮定する場合には有効な方式であると位置付けられる．

9. ま と め

本論文では，命令の発行時刻情報を用いた理想的な命令ステアリング方式に対する，有効な近似方式を検討した．先行研究における近似方式では，選択するクラスタを（1）Wakeup 遅延に着目して，未解決オペランド生成クラスタ，(2）Select 遅延に着目して，命令数が少ないクラスタに限定する近似は妥当であったが，両者から 1 つを選択する部分に近似の劣化が確認された．

そこで我々は，命令の発行時刻の定義から導出した近似式を用いて，データ生成命令と消費命令の命令間距離を用いることで，先行方式よりも高精度な Select 遅延推定を行う Local Distance Steering を提案した．この方式は OOO 発行のクラスタアーキテクチャに対して，先行研究 Parcerisa 方式から 9.2%の CPI 改善を確認し，残る性能向上の余地は 3.3%しか存在しないことを確認した．また FIFO 発行のクラスタアーキテクチャに対しては，先行研究 Palacharla 方式から 5.0%の CPI 改善が確認された．

さらに我々は，Local Distance をより簡単な方法で近似する Global Distance Steering を提案した．この方式は OOO 発行のクラスタアーキテクチャに対して，Parcerisa 方式から 5.7%の CPI 改善が確認された．しかし FIFO 発行のクラスタアーキテクチャに対しては，Palacharla 方式からの改善は確認されなかった．

また，通信遅延やクラスタの発行幅を変えた場合，あるいはデータキャッシュを理想化しない場合でも，提案手法 Local Distance Steering が有効であることを

確認した．さらに，OOO 発行の場合は提案方式を用いた Dispatch-Driven なクラスタアーキテクチャが，Execution-Driven に迫る性能を発揮することを確認した．

今後の課題としては，以下の 2 点を考えている．今回の提案は，全クラスタにレジスタファイルやキャッシュの内容が複製されることを前提としていた．しかし資源の利用効率を考えると，そのような複製には無駄が多い．そこで，複製を制限するアーキテクチャに対しても提案方式を適用させたいと考えている．また，今回はクロスバ型のネットワークトポロジを前提にしていたが，リング等の他のトポロジに対しても提案方式の評価，あるいは改良を行う必要があると考えている．

謝辞 本論文の研究は，一部 21 世紀 COE「情報技術戦略コア」による．

参 考 文 献

1) Agarwal, V., Hrishikesh, M.S., Keckler, S.W. and Burger, D.: Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures, *ISCA 2000*, pp.248–259 (2000).
2) Seznec, A. and Toullec, E. and Rochecouste, O.: Register Write Specialization Register Read Specialization: A Path to Complexity-Effective Wide-Issue Superscalar Processors, *MICRO 2002*, pp.383–394 (2002).
3) Baniasadi, A. and Moshovos, A.: Instruction Distribution Heuristics for Quad-Cluster Dynamically-Scheduled, Superscalar Processors, *MICRO 2000*, pp.337–347 (2000).
4) Canal, R., Parcerisa, J.-M. and González, A.: A Cost-Effective Clustered Architecture, *PACT 1999*, pp.160–168 (1999).
5) Hrishikesh, M.S., Jouppi, N.P., Farkas, K.I., Burger, D., Keckler, S.W. and Shivakumar, P.: The Optimal Logic Depth Per Pipeline Stage is 6 to 8 FO4 Inverter Delays, *ISCA 2002*, pp.14–24 (2002).
6) Farkas, K.I., Chow, P., Jouppi, N.P. and Vranesic, Z.: The Multicluster Architecture: Reducing Cycle Time Through Partitioning, *MICRO 1997*, pp.149–159 (1997).
7) Kessler, R.E.: The Alpha 21264 Microprocessor, *IEEE Micro*, pp.25–36 (1999).
8) Kim, H.-S. and Smith, J.E.: An Instruciton Set and Microarchitecture for Instruction Level Distributed Processing, *ISCA 2002*, pp.71–86 (2002).
9) Kim, I. and Lipasti, M.H.: Half-Price Architecture, *ISCA 2003*, pp.28–38 (2003).
10) Palacharla, S., Jouppi, N.P. and Smith, J.E.: Complexity-Effective Superscalar Processors, *ISCA 1997*, pp.206–218 (1997).
11) Parcerisa, J.-M. and González, A.: Reducing Wire Delay Penalty through Value Prediction, *MICRO 2000*, pp.317–326 (2000).
12) Parcerisa, J.-M., Sahuquillo, J., González, A. and Duato, J.: Efficient Interconnects for Clustered Microarchitectures, *PACT 2002*, pp.291–300 (2002).
13) Raasch, S.E., Binkert, N.L. and Reinhardt, S.K.: A Scalable Instruction Queue Design Using Dependence Chains, *ISCA 2002*, pp.318–329 (2002).
14) Sprangle, E. and Carmean, D.: Increasing Processor Performance by Implementing Deeper Pipelines, *ISCA 2002*, pp.25–34 (2002).
15) Stark, J., Brown, M.D. and Patt, Y.N.: On Pipelining Dynamic Instruction Scheduling Logic, *MICRO 2000*, pp.57–66 (2000).
16) 小林良太郎，安藤秀樹，島田俊夫：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スカラ・プロセッサにおける命令発行機構，*JSPP 2001*, pp.31–38 (2001).

(平成 16 年 1 月 31 日受付)
(平成 16 年 5 月 21 日採録)

服部 直也

1976 年生．1999 年東京大学工学部電子情報工学科卒業．2004 年同大学院情報理工学系研究科電子情報学専攻博士課程修了．情報理工学博士．プロセッサアーキテクチャ等の研究に従事．

高田 正法（学生会員）

1979 年生．2003 年東京大学工学部電子情報工学科卒業．現在，同大学院情報理工学系研究科電子情報学専攻修士課程在学中．プロセッサアーキテクチャ等の研究に従事．

岡部 淳

1976 年生．1999 年早稲田大学理工学部電子通信学科卒業．2001 年東京大学大学院工学系研究科情報工学専攻修了．現在，同大学院情報理工学系研究科電子情報学専攻博士課程在学中．プロセッサアーキテクチャ等の研究に従事．

入江 英嗣（正会員）

1975 年生．1999 年東京大学工学部電子情報工学科卒業．2004 年同大学院情報理工学系研究科電子情報学専攻博士課程修了．情報理工学博士．プロセッサアーキテクチャ等の研究に従事．

坂井 修一（正会員）

1981 年東京大学理学部情報科学科卒業．1986 年同大学院工学系研究科情報工学専門課程修了．工学博士．同年工業技術院電子技術総合研究所入所．1991 年〜1992 年，米国マサチューセッツ工科大学招聘研究員，1993 年〜1996 年 RWC 超並列アーキテクチャ研究室室長．1996 年筑波大学電子・情報工学系助教授．1998 年東京大学大学院工学系研究科助教授，2001 年同大学院情報理工学系研究科教授．計算機システム一般，特にアーキテクチャ，並列処理，スケジューリング問題，マルチメディア応用等の研究に従事．著書『論理回路入門』，『図説コンピュータアーキテクチャ』．電子情報通信学会，人工知能学会，IEEE，ACM 各会員．

田中 英彦（フェロー）

1965 年東京大学工学部電子工学科卒業．1970 年同大学院工学系研究科博士課程修了．工学博士．同年同大学工学部講師．1971 年同助教授．1987 年同教授．2001 年より同大学院情報理工学系研究科教授・研究科長．この間 1978 年〜1979 年米国ニューヨーク市立大学客員教授．計算機アーキテクチャ，並列処理，自然言語処理，メディア処理，分散処理，CAD 等の研究に興味を持っている．著書『非ノイマンコンピュータ』,『情報通信システム』,『Parallel Inference Engine—PIE』，共著書『計算機アーキテクチャ』,『VLSI コンピュータ I，II』，『ソフトウェア指向アーキテクチャ』．本会フェロー．電子情報通信学会，人工知能学会，日本ソフトウェア科学会，IEEE，ACM 各会員．