Vol.2021-ARC-245 No.1
2021/7/20

STRAIGHTアーキテクチャにおける スケーラブルなスケジューラの提案と評価

新 浩太朗[1]　小泉 透[1]　杉田 脩[1]　光野 聡志[1]　門本 淳一郎[1]　塩谷 亮太[1]　入江 英嗣[1]　坂井 修一[1]

概要：プロセッサのシングルスレッド実行性能は向上し続けており，Out-of-Order スーパースカラプロセッサの規模は大型化している．距離表現を用いた命令セットを持つ STRAIGHT アーキテクチャでは，レジスタリネーミングがなくなることによってフロントエンド幅やリオーダバッファを拡張しやすくなる．一方で，命令レベル並列性の抽出にはスケジューラのエントリ数も重要である．本論文では，STRAIGHT アーキテクチャの特徴を利用した，面積当たりのエントリ数効率の良いスケジューラを提案する．また，それによって低下してしまうエントリ利用効率を，リプレイを導入することで改善する改良手法も提案する．複数の最大参照距離における提案手法の評価を行い，従来手法と比べて同一規模でより優れた性能を出しうることを確認した．

Proposal and Evaluation of Scalable Scheduler for STRAIGHT Architecture

1. はじめに

プロセッサのシングルスレッド性能の向上は，並列化しづらい逐次処理部分の高速化に効果的である．これは並列化の進んだタスクを含むあらゆるプログラムの高速化に寄与する．

Our-of-Order スーパースカラプロセッサ [1] [2] は，プログラムから命令レベル並列性（ILP）を抽出して複数の命令を同時に実行するプロセッサであり，シングルスレッド性能を向上させるために広く使われている．このアプローチでは，ムーアの法則に従って増え続けたトランジスタ数を活用し，より多くの命令レベル並列性を抽出することで，性能を向上させてきた．これは，より多くの命令を同時に実行することや，高度な予測による投機的な実行を行うことによって達成されてきた．

しかし，このアプローチの大きな課題として電力効率の悪さが挙げられる．これは，積極的に ILP を抽出するために搭載した機構には本質的に演算そのものとは無関係のものがあるためである．また，さらなる性能向上のために同時に処理可能な命令数を増やし続けることには回路面積及び電力の爆発的増加という問題がある．

これらの問題を解決するためのアプローチとして，STRAIGHT アーキテクチャが提案されている [3]．STRAIGHT の命令セットアーキテクチャ（ISA）では，ソースレジスタの指定に命令間の距離を用いる．ディスティネーションレジスタは命令中で指定されず，上書きが起こらないように割り当てられるので，命令間の偽の依存が存在しない．そのため，従来の Out-of-Order スーパースカラプロセッサにおいて面積や消費電力に大きな影響を及ぼしているレジスタリネーミング [4] は不要となる．フロントエンドからリネームがなくなることにより，フロントエンド幅の拡大が容易になる．また，レジスタマップテーブルの復帰が必要なくなるため，予測ミス時のリカバリが高速になり，リオーダバッファの拡大も容易になる．

しかし，同時に実行できる命令を増やすためには，リオーダバッファの拡大だけでなく，スケジューラエントリ数の拡大が必要である．これは，スケジューラのエントリ数を拡大することにより，次に実行する命令の候補を増やすことができるからである．一方でスケジューラのロジックは，パイプライン化することが困難であり，クリティカルパスとなりやすい [5]．これは，このロジックを複数サイクルで行ってしまうと性能に影響するからである．スケ

[1] 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

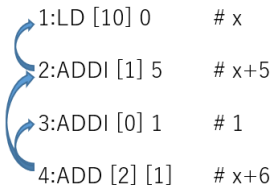

図 1 STRAIGHT アセンブリコードの例
Fig. 1 An example of STRAIGHT assembly code

ジューラを拡大することは，このクリティカルパスをさらに伸ばすことにつながるため，工夫なしに行うと動作周波数の低下を招く．同様にスケジューラの拡大には，回路面積の増加に伴う消費電力増加を引き起こす．したがって，スケジューラエントリ数の拡大は簡単ではない．

本論文では，STRAIGHT の特徴を活かしてスケジューラの回路を圧縮する方法を提案し，その特性を評価する．このスケジューラは，面積や遅延を一定とした場合，既存手法よりも大きなエントリ数を実現することが可能である．また，この手法によりエントリ利用効率が低下してしまうため，リプレイ [6][7][8] を導入した改良手法も提案し，その特性を評価する．

以降，第 2 章では STRAIGHT アーキテクチャについて述べる．第 3 章では，スケジューラの構造と拡大におけるコストについて説明し，第 4 章では本論文で提案するスケジューラについて述べる．最後に第 5 章では提案手法の評価を行う．

2. STRAIGHT アーキテクチャ

2.1 STRAIGHT ISA

STRAIGHT ISA では，ソースオペランドは何命令前の結果を参照するという形で指定され，ディスティネーションレジスタ番号は命令語中では指定されない．代わりにディスティネーションレジスタは，プログラム順に割り当てられる．図 1 に STRAIGHT のアセンブリコード例を示す．角括弧内に示されている値は何命令前の結果をソースオペランドとするかを示し，角括弧のない値は即値である．例えば，「[2]」は 2 命令前の結果をソースオペランドとすることを示し，「[0]」はゼロレジスタを指す．また，ソースオペランドとして指定する命令間の距離の最大値は，アーキテクチャで定められる最大参照距離によって制限される．

ディスティネーションレジスタはプログラム順に 1 つずつ割り当てられていくため，いつかはレジスタが一巡し，以前他の命令に割り当てたレジスタを再使用することになる．しかし，最大参照距離により，新しい命令にレジスタが割り当てられた時点には，そのレジスタの値は既に最後の参照を終えていることが保証されるため，偽の依存を生じさせるようなレジスタの上書きは発生しない．このことにより，偽の依存が ISA レベルで存在しないことが保証されるので，レジスタリネーミングを行わずに済む．

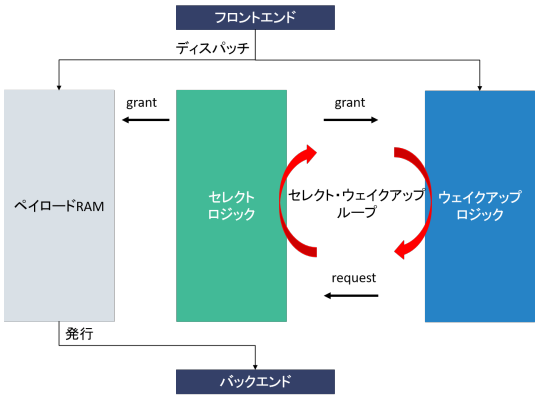

図 2 スケジューラの構成
Fig. 2 Structure of schduler

2.2 STRAIGHT アーキテクチャにおけるレジスタ割り当て

STRAIGHT アーキテクチャでは，レジスタリネーミングの代わりに，レジスタの割り当てとソースレジスタの決定のみを以下のように行う．

各命令は，レジスタポインタ（RP）という特殊レジスタの値を自身のディスティネーションレジスタ番号として使う．RP の値は基本的に一つの命令のディスティネーションレジスタとして割り当てるごとにインクリメントされていき，物理レジスタ数を超えた場合には 0 に戻る．また，ソースレジスタ番号は RP の値からオペランドで指定された命令間距離を引くことで算出する．

結果として，STRAIGHT においてレジスタ割り当てに必要な機構は RP と簡単な回路で実現できる [9]．そのため，従来の Out-of-Order スーパースカラプロセッサではレジスタリネーミング機構によって妨げられていたフロントエンド幅や，物理レジスタ数の拡大が STRAIGHT においては容易になる．このことは従来アーキテクチャと STRAIGHT との大きな違いとなっている．

3. スケジューラ

3.1 概要

スケジューラは，in-order にフロントエンドパイプラインを通過してきた命令を，依存するデータの完了状態や使用する演算器の利用状況に応じて，Out-of-Order にバックエンドパイプラインに送る役割を持つ．スケジューラは図 2 のように，ペイロード RAM，セレクトロジック，ウェイクアップロジックに分けられる [10]．

フロントエンドパイプラインを抜けた命令は，スケジューラエントリのフリーリストから空いているスケジューラエントリを割り当てられ，in-order に格納される（ディスパッチ）．ここで，格納すべきエントリ番号の割り当ては，

フリーリストから一つ取り出すことで行われる．フリーリストは，どのエントリが使われていないかを保持するハードウェアである．

ウェイクアップロジックは，ディスパッチされた命令のそれぞれについて，全てのオペランドが揃い実行可能であるかどうかを管理する．これは，オペランドごとに，依存する命令の結果が揃っているかを示す ready ビットを用いることで行う．全てのオペランドが ready な命令は，セレクトロジックに対して，request 信号を送出する．

セレクトロジックでは，request 信号を送った命令の中から，演算器の状況などを考慮して，バックエンドパイプラインに発行する命令を選択する．この際ウェイクアップロジックでは，選択された命令に依存する命令の ready ビットを更新する必要がある．そのためセレクトロジックは，grant 信号をウェイクアップロジックの全命令に向けて送出（ブロードキャスト）する必要がある．それと同時に，選択された命令は，ペイロード RAM からバックエンドパイプラインに発行されていく．

このように，ウェイクアップロジックとセレクトロジックは，フィードバックループ（セレクト・ウェイクアップループ）を形成している．このループは，パイプライン化して複数サイクルに分けてしまうと，依存先の命令を依存元の命令の実行の直後に実行できないことが知られている [5]．そのため，このループのロジックは 1 サイクルのうちに行うことが望ましい．

プロセッサがプログラムから抽出できる ILP は，スケジューラがどれだけの命令を選択の対象とできるかによって決まる．そのためスケジューラのエントリ数をより大きくすることで，実行パイプラインの使用率を向上させることができ，結果的に全体の性能向上につながる．

3.2 マトリクス方式ウェイクアップロジック [11]

マトリクス方式は，連想検索を用いる従来の方法よりも面積効率のよいウェイクアップロジックを構成できる．そのため，この論文ではこの方式のウェイクアップロジックをベースとする．

マトリクス方式ウェイクアップロジックは，図 3 のように，命令とエントリの対応関係を管理する Wake up Allocation Table（WAT）と命令同士の依存関係を表すプロデューサーマトリクスに分けることができる．

WAT は，スケジューラに命令をディスパッチする際に，依存している命令がどのエントリに割り当てられているかを取得するのに用いられる．

プロデューサーマトリクスは，エントリ数を n とした場合，一辺が n の正方行列状にセルが敷き詰められたような構造となっている．図 4 に，スケジューラのエントリ数を 4 とした場合のプロデューサーマトリクスを示す．正方行列の各セルは，命令同士の依存関係を表現しており，i 行 j

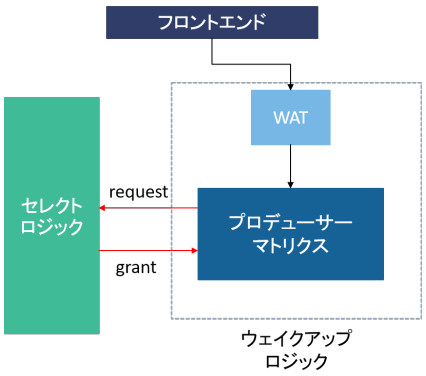

図 3 マトリクス方式ウェイクアップロジックの構成（赤矢印がセレクト・ウェイクアップループを表す）

Fig. 3 Structure of wake up logic of matrix scheduler (red arrows represent select-wakeup loop.)

0	0	0	0	1:ready
1	0	0	0	2:命令1に依存
0	1	0	1	3:命令2,4に依存
0	0	0	0	4:ready

図 4 プロデューサーマトリクスの例

Fig. 4 An example of producer matrix

列のビットが 1 であった場合，エントリ i の命令はエントリ j の命令に依存していることを示している．つまり，i 行目の全ての列のビットが 0 になると，エントリ i の命令は実行可能であることになる．この状態になった時，ウェイクアップロジックは request 信号を送出する．

3.3 スケジューラの拡大のコスト

スケジューラエントリ数を拡大してより多くの ILP を抽出することは性能向上につながる一方，回路面積やクリティカルパス遅延の増大といった問題が生じる．

まず，回路面積の増大について述べる．3.2 節からわかるように，マトリクス方式のスケジューラにおいて，プロデューサーマトリクスはスケジューラのエントリ数の 2 乗に比例する回路面積を要する．このため，性能向上のためにエントリ数を増やしても，それに見合わない回路面積・消費電力の増加を引き起こす恐れがある．

次に，クリティカルパス遅延の増大について述べる．エントリ数を増大させることによってウェイクアップロジックの遅延が大きくなり，セレクト・ウェイクアップループ全体の遅延が大きくなる．性能のためにはセレクト・ウェイクアップループを 1 サイクルで行わないといけないことを考えると，この遅延はパイプライン化によって緩和でき

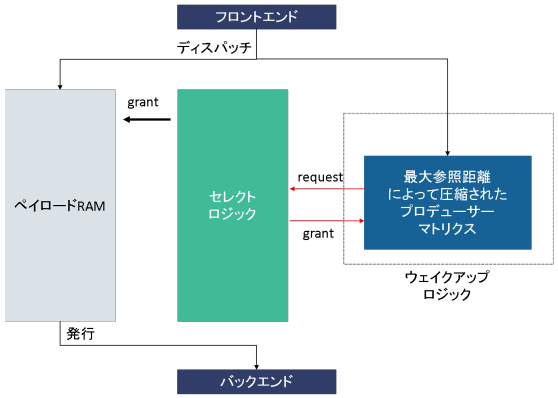

図 5　提案するスケジューラの構成（赤矢印はセレクト・ウェイクアップループ）

Fig. 5　Structure of scheduler in proposal method(Red arrows represent select-wakeup loop)

ず，動作周波数を制約するおそれがある．

4. 提案手法

4.1 概要

この章では，STRAIGHT アーキテクチャの特徴を利用することで，回路面積やクリティカルパス遅延の不必要な増大を防ぎつつもエントリ数を大きくすることを可能にするスケジューラを提案する．また，このことにより生じてしまうスケジューラエントリ利用効率の低下を緩和する改良手法について 4.4.2 項で述べる．

提案手法では，スケジューラのエントリをディスティネーションレジスタと同様に RP を用いて割り当てる．このようにすることで，以下の 2 つの利点が生まれる．1 つ目は，プロデューサーマトリクスの一部は必ず使われないことが保証され，省略することができるということである．2 つ目は，エントリの割り当てを行うために必要だった WAT やフリーリストが要らなくなるということである．結果的に，提案するスケジューラの構成は図 5 のようになる．

以下に 2 つの利点について詳しく述べる．

4.2 最大参照距離によるプロデューサーマトリクスの圧縮

提案手法においては図 6 のように，各命令に割り当てられたエントリ間の距離と，各命令に割り当てられた RP の差が等しくなるように割り当てる．スケジューラのエントリをこのように割り当てることによって，あるエントリに対応する命令がセレクトロジックによって選択されたとき，最大参照距離以上離れたエントリにブロードキャストする必要がなくなる．エントリ数 5，最大参照距離 3 の際の例を図 6 に示す．最大参照距離は 3 であるので，RP が 3 である命令 A の結果は，最大でも RP が 6 の命令にしか参照されない．各命令に割り当てられた RP の差とエントリ間の距離は等しいため，エントリ 1 に格納された命令 A の結

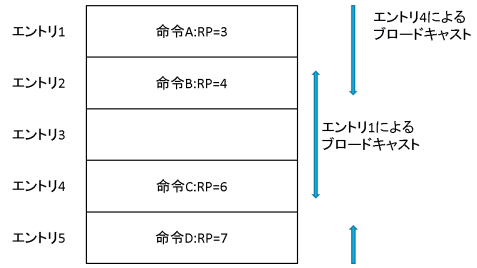

図 6　提案手法におけるエントリ割り当てとブロードキャスト

Fig. 6　Entry allocation and broadcast in proposal method

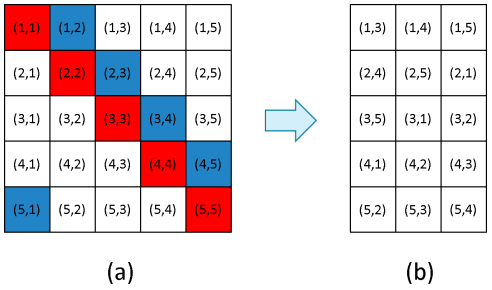

図 7　プロデューサーマトリクスの圧縮（左:圧縮前，右:圧縮後）

Fig. 7　Compression of producer matrix. (a) before compression. (b) after compression.

果は，エントリ 2,3,4 に格納された命令にブロードキャストするだけでよい．エントリ間の距離は循環して計算されるため，命令 C についても同様に，エントリ 5,1,2 に格納された命令にブロードキャストするだけでよい．

以上を踏まえると，提案手法において，エントリ数 5，最大参照距離 3 のプロデューサーマトリクスは図 7 のように圧縮できる．圧縮前の図左のプロデューサーマトリクスにおいて，青く塗られたセルは最大参照距離を超えた距離の依存を表すセルであるため，不要となる．また赤く塗られたセルは命令自身への依存を表すセルであり，これも不要である．

圧縮の結果，エントリ数 n，最大参照距離 M のスケジューラにおけるウェイクアップマトリクスのセル数は n^2 から，$n \times \min\{n, M\}$ まで減らすことができる．例として，マトリクス方式と，最大参照距離 31, 96, 127 それぞれにおける提案手法のセル数を図 8 に示す．エントリ数が最大参照距離と一致するまでは同一のセル数だが，それ以降はマトリクス方式がエントリ数の二乗に比例するのに対して，提案手法ではエントリ数に比例して増加していく．そのため，エントリ数が大きくなればなるほど圧縮の効果も大きくなり，同一規模のウェイクアップロジックを考えたときに，マトリクス方式に比べて大きなエントリ数のスケジューラを構成することが可能になる．

最大参照距離を小さくしても圧縮の効果は大きくなるが，最大参照距離を小さくすることで，値を中継するための RMOV 命令などが追加で挿入されることがあるので，命令数増加とのトレードオフとなる．

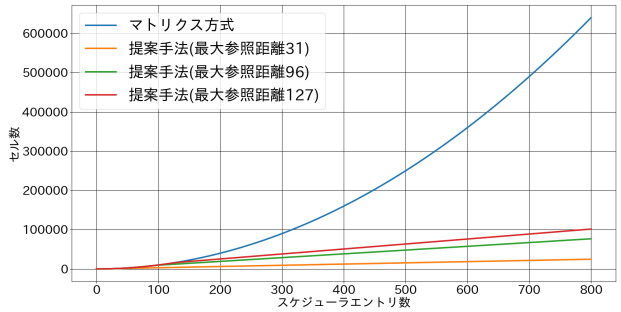

図 8　マトリクス方式と提案手法のウェイクアップロジックセル数
Fig. 8　Cell number in matrix method and proposal method

4.3　WAT

提案手法では，各命令に割り当てられたエントリ間の距離と RP の差が等しくなるため，ある命令がどのエントリの命令に依存するかは，自身に割り当てられたエントリ番号とソースオペランドから算出できる．そのため，提案するスケジューラにおいては，WAT は必要なくなる．

このことによりディスパッチの際に，スケジューラエントリのフリーリストから空いているエントリを探す必要がなくなるので，提案手法においてはディスパッチ幅を拡大することが容易になる．

4.4　リプレイを利用したエントリ利用効率低下の緩和

4.4.1　提案手法でのエントリ利用効率の低下

提案手法におけるエントリ割り当てによって，以下の 2 つの要因によるエントリ利用効率の低下が生じる．

1 つ目は，不連続な RP によるエントリの歯抜けを要因とするものである．STRAIGHT では，連続した命令には基本的には連続した RP が振られるが，距離調整のための NOP 命令などにより，不連続となる場合がある．この場合，そのような命令間のエントリは実際には使われていないにも関わらず新たに割り当てることはできない．例えば図 6 では，エントリ 2 の命令 B とエントリ 4 の命令 C は連続した命令であるにも関わらず，RP は不連続となっている．そのため，間のエントリ 3 は歯抜けになってしまっている．

2 つ目は，スケジューラからの脱退が遅れている命令を要因とするものである．近年のプロセッサでは，投機的スケジューリング [12] によって，依存する命令の結果が正しく計算されたことを確かめずに発行することで性能を向上させることが普通となっている．例えばロード命令に依存する命令は，ロード命令がキャッシュヒットすると仮定して投機的に発行することで，キャッシュミス/ヒットが判明してから発行するよりも早く実行を行うことができる．しかし，キャッシュミスなどにより，投機的スケジューリングミスが判明した場合には，投機的に発行された命令を再スケジュールすることが必要である．再スケジュールの際には，初回のスケジューリングと同様の情報が必要となるため，実行終了時に投機的スケジューリングが正しく行われたことを確認するまではスケジューラのエントリを解放してはいけない [13]．そのため，ロード命令がキャッシュミスを起こした場合には，数十サイクルから数百サイクルの間スケジューラから脱退することはない．このような命令がスケジューラに残っているうちは，いくら後続の命令が実行を終了してスケジューラを脱退しても，新たにエントリを割り当てられない．例えば図 6 において，命令 A はキャッシュミスを引き起こし，後続の命令 B〜D は正常に終了して脱退したとする．このときに新たに RP が 9 である命令 E がディスパッチされようとしていても，命令 E が格納されるエントリには命令 A がまだ脱退せずに残っているため，割り当てることができない．

エントリ利用効率の低下によってエントリを割り当てられなくなると，フロントエンドパイプラインをストールする必要がある．この 2 つの要因のうち，脱退が遅れた命令を要因とするものは特に，たった一つの命令によって長期間エントリ利用効率の低下を引き起こすことになり，性能への影響は著しいと予想できる．そこで，この性能低下を緩和するためにリプレイという手法を導入する．

4.4.2　リプレイ

リプレイ [6][7][8] は，投機的スケジューリングが失敗した時にスケジューラの代わりにリプレイキューから再発行を行う手法である．これによって，投機的スケジューリングの成否に関係なく命令の発行時にスケジューラのエントリを解放することができる．投機的スケジューリングが正しく行われなかったと判明したら，再発行に必要な情報をリプレイキューに入れた後，一定のサイクルだけ待ってから再発行する．

リプレイキューは，スケジューラのエントリと比べてポート数は少なく，セレクト・ウェイクアップループに含まれないため，クリティカルパスとはならない．そのため，スケジューラと比べて，エントリ数の拡大は容易である．

リプレイを採用した場合には，ロード命令がキャッシュミスを起こしてもスケジューラに長くとどまり続けることがなくなる．したがって，そのような命令によるフロントエンドストールが生じるサイクルが減る．

5.　評価

5.1　評価環境

投機的実行を含めたパイプライン動作の正確にシミュレーションするために，サイクル精度でプロセッサのシミュレーションを行うことのできる鬼斬弐 [14] を使用した．また，サイクリック方式のスケジューラやリプレイを実装した．

また，ベンチマークプログラムとして，CoreMark[15]，SPEC による CPU2017[16] のうち 605.mcf_s，619.lbm_s，

657.xz_s を使用した．605.mcf_s，619.lbm_s，657.xz_s については，シミュレーションの高速化のため，主要な部分を切り出してシミュレーションを行った．計測区間は，605.mcf_s，657.xz_s は 2 つずつ，619.lbm_s は 1 つ選定した．この計測区間は，最大参照距離の異なるバイナリ同士での比較を行えるようにする必要がある．そのため，小泉 [17] と同様に，実行回数の多い関数を含む特定の区間という形で選定した．これらのプログラムを最大参照距離を 127 及び 96 に設定してコンパイルした．

性能の指標としては，4 つのベンチマークプログラムのそれぞれを実行するのにかかったサイクル数を用いた．605.mcf_s，657.xz_s については，それぞれ計測区間が 2 つずつあるため，それぞれの計測区間が全体に占める割合によって実行サイクル数に重み付けをしたものをそのベンチマークの性能の指標とした．619.lbm_s については，計測区間の実行サイクル数をそのまま性能の指標とした．また，CoreMark は，10 周分全体を実行するのにかかったサイクル数を用いた．

評価には，従来手法のウェイクアップロジックを採用した Base，提案手法を採用した Cyclic，リプレイを含んだ改良手法を採用した CyclicReplay という 3 つのモデルを使用した．この 3 つに共通するプロセッサパラメータを表 1 に示す．モデル間でスケジューラの違いが性能にどう影響するかを確かめるため，スケジューラ以外の要素がボトルネックとならないように設定した．

それぞれのモデルにおけるスケジューラのエントリ数を表 2 に示す．評価の目的は，マトリクス方式と提案手法で同一規模のスケジューラを構成したとき，性能がどのように違うか確かめることことである．そのため，提案手法におけるエントリ数は，マトリクス方式と同等のセル数を持つよう最大参照距離に応じて決定した．

なお，ウェイクアップロジックの遅延に関しても，ブロードキャストの信号線がいくつのセルをまたぐかという基準 [18] で考えると，表に示したエントリ数においてマトリクス方式と提案手法で大きく変わることはない．

以降では，モデル，スケジューラエントリ数，最大参照距離の組によって結果を表すこととする．例えば，CyclicReplay-144-M96 というのは，モデル CyclicReplay において，エントリ数 144 のマトリクス方式と同等の規模を持つ，最大参照距離 96 で構成したスケジューラを用いたことを表す．

5.2 同一規模のスケジューラでの比較

図 9 に，Base-96-M127 の性能を 1 とした実行性能比の，4 つのベンチマークプログラムにおける幾何平均を示す．また図 10 に，それぞれのベンチマークにおける，Base-96-M127 の性能を 1 としたときの実行性能比を示す．

a) 最大参照距離による性能の違い

図 9，10 において，提案手法を用いた CyclicReplay-144,160,180-M127 と-M96 とを比較する．すると，最大参照距離を 127 から 96 にすることによって，性能が上がっていることが分かる．これは最大参照距離を 127 から 96 にすることによって，4.2 節で述べたように圧縮の効果が高まり同一の規模でもスケジューラエントリをより大きくすることができているためである．ただし，スケジューラエントリ数を増やすことにあまり意味の無いベンチマーク 657.xz_s では，性能はほぼ変わらない．

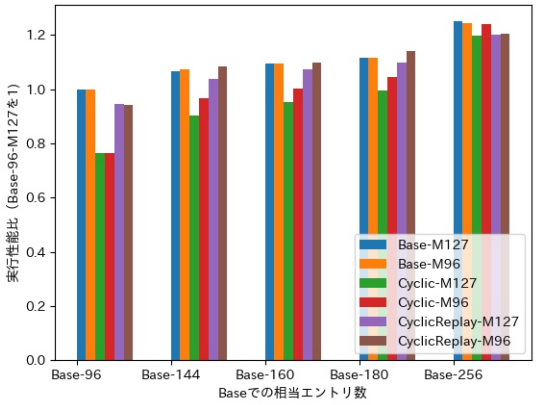

図 9 Base-96-M127 の性能を 1 としたときの実行性能比の幾何平均
Fig. 9 Geometric mean of the execution performance ratio. The value is normalized by the performance of Base-96-M127.

b) マトリクス方式と提案手法との性能の違い

図 9 より，CyclicReplay-144,160,180-M96 と，それぞれ同等の規模のマトリクス方式とを比較すると，提案手法の方が最大で 2%高い性能を出していることが分かる．これは提案手法が，同一規模のマトリクス方式に比べてエントリ数を大きくすることができ，プログラムからより多くの ILP を抽出することができるからである．また図 10 より，特に最大参照距離 96 における計測区間 lbm では，提案手法を用いた CyclicReplay-180-M96 が，Base-180-M127 に対して 17%の性能向上を達成していることが分かる．

c) リプレイによる影響

リプレイを採用した CyclicReplay と，採用していない Cyclic とを比較すると，エントリ数が多い場合を除いてリプレイを採用することで性能が向上していることや，その向上幅はより小さいエントリ数で顕著であることが分かる．これは 4.4 節で述べたように，提案手法のスケジューラをそのまま使うと，スケジューラからの脱退が遅れた命令によってエントリの利用効率が大幅に下がってしまう一方，リプレイを採用することでエントリの利用効率を向上させることができるからである．

一方で，CyclicReplay-256-M127,96 では，マトリクス方式の 2 倍以上のエントリを実現できるにも関わらず，どのベンチマークでも Base,Cyclic の両方よりも性能が低下し

表 1　評価に用いるプロセッサパラメータ
Table 1　Parameters for evaluation

フロントエンド/発行/コミット幅	8/18/16
スケジューラエントリ数	表 2 に示す
ロード/ストアキュー	256/144 エントリ
リオーダバッファ	1024 エントリ
L1I,D キャッシュ	32 KiB, 8 way, 64 B line, アクセスレイテンシ 4 サイクル
L2 キャッシュ	256 KiB, 8 way, 64 B line, アクセスレイテンシ 8 サイクル
L3 キャッシュ	2 MiB, 16 way, 64 B line, アクセスレイテンシ 30 サイクル
プリフェッチャ	Stream prefetcher (L3) distance 8, degree 2, 16 エントリ

表 2　同一規模のスケジューラのエントリ数（括弧内はセル数）
Table 2　Capacity for schedulers of the same scale(the number of cells is shown in parentheses)

マトリクス方式（Base）	96（9216）	144（20736）	160（25600）	180（32400）	256（65536）
最大参照距離 127 での提案手法（Cyclic, CyclicReplay）	96（9216）	164（20828）	201（25527）	256（32512）	512（65024）
最大参照距離 96 での提案手法（Cyclic, CyclicReplay）	96（9216）	216（20736）	267（25632）	338（32448）	682（65472）

てしまっている．この要因を，5.3 節で考察する．

5.3　リプレイによる性能低下の考察

パイプライン可視化ツール konata[19] を用いて性能が低下した計測区間の実行の流れを可視化することで要因について調査した．その結果，以下のような条件が揃うことでリプレイの導入が大きな性能低下の要因となっていることが判明した．

- ある命令が投機的スケジューリングミスを起こす．
- その命令に依存する命令も投機的スケジューリングミスを起こす．
- リオーダバッファが埋まるなどしてフロントエンドパイプラインがストールする．

図 11 に性能低下の要因の具体例を示す．命令 A は，投機的スケジューリングミスによってリプレイされ，その後さらにキャッシュミスを起こすため再リプレイを行う必要がある．命令 B は，命令 A に依存するため，一回目のリプレイの後，再びリプレイキューに格納される．命令 C は，命令 B に依存するため，同様に，リプレイの後再びリプレイキューに格納される．ただし，命令 C は，フロントエンドストールなどによってスケジューラに格納されるのが遅れたため，初回の発行が遅れている．

リプレイキューでの待機サイクルは，直前にリプレイキューに入った発行グループとの発行タイミングの差が同一になるように決定される．これによってリプレイにおいては，初回の発行を再現して繰り返すことになる．２回目以降のリプレイでの待機サイクルを決定するときにも同じことが言え，初回のスケジューリングの際の発行タイミングの差がそのまま使われる．そのため，命令 C のように発行のタイミングがフロントエンドストールなどによって遅れた場合には，依存する命令 B の結果が揃っているにもかかわらず不必要にリプレイキューの中で待たされることとなり，このことは命令 C のレイテンシのみならず，命令 C に依存する命令のレイテンシにも影響を与える．

この現象は，一度発生するとスケジューラからの発行が止まらない限り連鎖的に続いていくという特徴がある．規模の大きいプロセッサでは資源不足による発行の停止が起こりにくいため，影響がより大きくなる．

このため，この現象による性能低下を防ぐための方向性は，発行された命令の多くがリプレイキューに入り，再リプレイの引き金となっていることを検知し，一時的に発行を停止したりフロントエンドパイプラインをストールさせるなど意図的に命令の処理数を少なくするというものである．この際，あまり大幅に処理数を少なくすると全体の性能を下げることにつながるので，処理数の削減は必要最低限とする必要がある．

6.　おわりに

本論文では，STRAIGHT アーキテクチャの特徴を利用することで，同じウェイクアップロジックセル数で従来より多くのエントリ数を実現することのできるスケジューラを提案し，それによって低下するエントリ利用効率を高めるリプレイという手法について述べ，複数の最大参照距離における提案手法のシミュレーションでの評価を行った．

その結果，最大参照距離によっては，提案手法によって従来手法よりも大きなエントリ数を実現することができ，全体では 2%，個々のベンチマークでは最大 17%の性能向上につながることが明らかになった．一方で，リプレイの採用が条件によっては性能低下に繋がってしまうことも明らかになった．

今後の課題としては，大規模な命令ウィンドウを持つプロセッサにおいて，リプレイによる性能を抑える方法の探

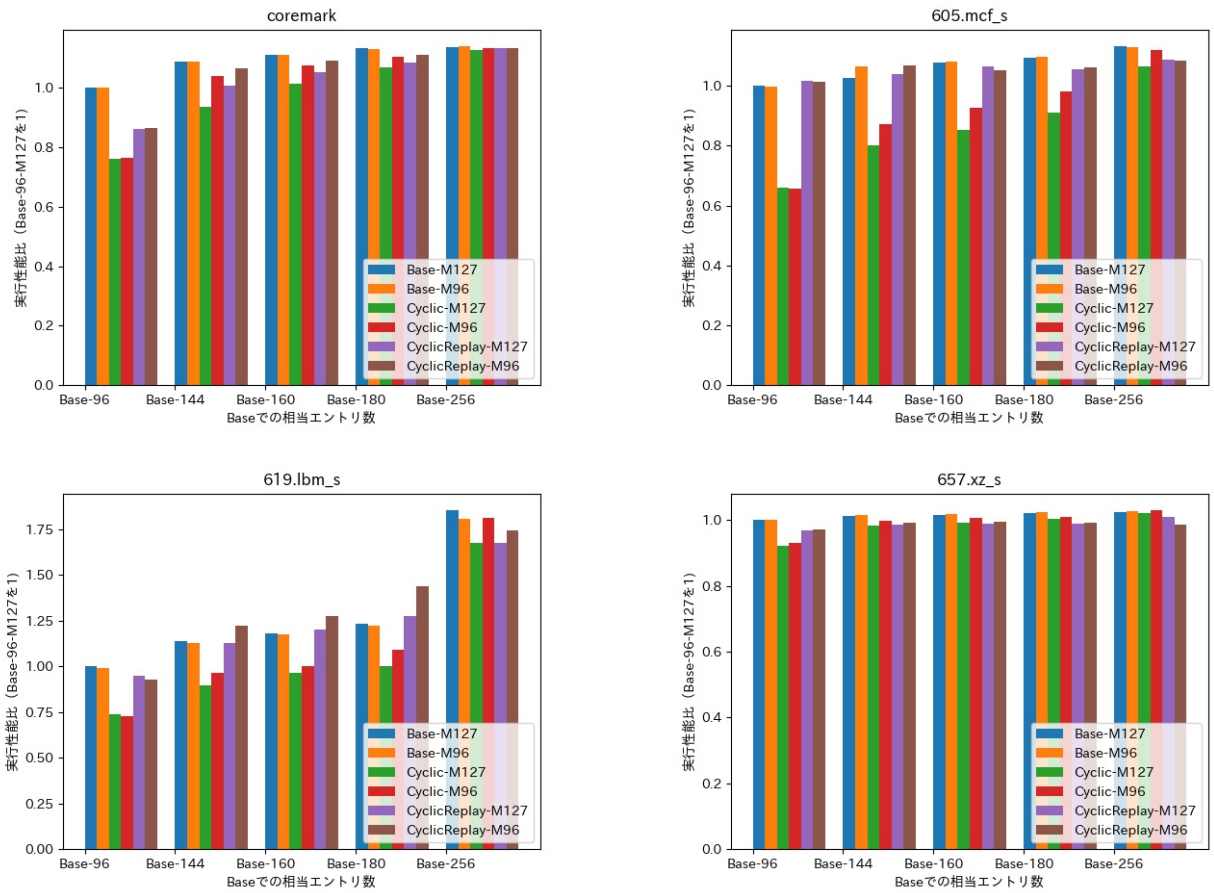

図 10　Base-96-M127 の性能を 1 としたときの各ベンチマークにおける実行性能比

Fig. 10　The execution performance ratio of each benchmark. The value is normalized by the performance of Base-96-M127.

図 11　リプレイの導入による性能低下の要因の例

Fig. 11　An example of performance degradation caused by introducing replay

求が挙げられる．また，提案スケジューラが実際の回路上実現可能か，またその回路面積や遅延はどの程度かについて，正確な評価を行うことも挙げられる．これは，最大参照距離についてのトレードオフを明らかにし，命令セットとの co-design を行うことを可能にする．

謝辞　本研究の一部は JSPS 科研費 JP19H04077，JP20H04153，JP20J22752 による．

参考文献

[1] Kessler, R. E.: The alpha 21264 microprocessor, *IEEE Micro*, Vol. 19, No. 2, pp. 24–36 (online), DOI: 10.1109/40.755465 (1999).

[2] Hinton, G., Sager, D., Upton, M., Boggs, D., Carmean, D., Kyker, A. and Roussel, P.: The Microarchitecture of the Pentium 4 Processor, Technical report.

[3] Irie, H., Koizumi, T., Fukuda, A., Akaki, S., Nakae, S., Bessho, Y., Shioya, R., Notsu, T., Yoda, K., Ishihara, T. and Sakai, S.: STRAIGHT: Hazardless processor architecture without register renaming, *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, Vol. 2018-Octob, pp. 121–133 (online), DOI: 10.1109/MICRO.2018.00019 (2018).

[4] Shioya, R., Horio, K., Goshima, M. and Sakai, S.: Register cache system not for latency reduction purpose, *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, pp. 301–312 (online), DOI: 10.1109/MICRO.2010.43 (2010).

[5] Palacharla, S., Jouppi, N. P. and Smith, J. E.: Complexity-effective superscalar processors, *Conference Proceedings - Annual International Symposium on Computer Architecture, ISCA*, pp. 206–218 (online), DOI: 10.1145/384286.264201 (1997).

[6] Amit A. Merchant and David J. Sagger: Computer processor having a checker, US Patent #6212626 (1998).

[7] Amit A. Merchant, Darrell D. Boggs and David J. Sager: Processor with a replay system that includes a replay queue for improved throughput, US Patent #7200727 (2007).

[8] Mashimo, S., Inoue, K., Shioya, R., Fujita, A., Matsuo, R., Akaki, S., Fukuda, A., Koizumi, T., Kadomoto, J., Irie, H. and Goshima, M.: An open source FPGA-optimized out-of-order RISC-V soft

processor, *Proceedings - 2019 International Conference on Field-Programmable Technology, ICFPT 2019*, Vol. 2019-Decem, pp. 63–71 (online), DOI: 10.1109/ICFPT47387.2019.00016 (2019).
[9] 光野聡志：Out-of-Order STRAIGHT ソフトプロセッサの実装とマイクロアーキテクチャ最適化 (2020).
[10] 赤木晟也：STRAIGHT プロセッサの実装と評価，修士論文，東京大学 (2018).
[11] Goshima, M., Nishino, K., Nakashima, Y., Mori, S. I., Kitamura, T. and Tomita, S.: A high-speed dynamic instruction scheduling scheme for superscalar processors, *Proceedings of the Annual International Symposium on Microarchitecture*, pp. 225–236 (online), DOI: 10.1109/micro.2001.991121 (2001).
[12] Perais, A., Seznec, A., Michaud, P., Sembrant, A. and Hagersten, E.: Cost-effective speculative scheduling in high performance processors, *Proceedings - International Symposium on Computer Architecture*, Vol. 13-17-June, Institute of Electrical and Electronics Engineers Inc., pp. 247–259 (online), DOI: 10.1145/2749469.2749470 (2015).
[13] Kim, I. and Lipasti, M. H.: Understanding scheduling replay schemes, *IEEE High-Performance Computer Architecture Symposium Proceedings*, Vol. 10, pp. 198–209 (online), DOI: 10.1109/hpca.2004.10011 (2004).
[14] 塩谷亮太，五島正裕，坂井修一：プロセッサ・シミュレータ「鬼斬弐」の設計と実装，先進的計算基盤システムシンポジウム SACSIS2009，pp. 269–273（オンライン），入手先 ⟨http://www.simplescalar.com/⟩ (2009).
[15] : CoreMark, https://www.eembc.org/coremark/.
[16] : SPEC CPU 2017, https://www.spec.org/cpu2017/.
[17] 小泉透， 杉田脩，塩谷亮太，入江英嗣，坂井修一：実用プログラムの STRAIGHT アーキテクチャにおける特性の解析，情報処理学会研究報告 (Web)，Vol. 2020, pp. 2020–242 (2020).
[18] Sassone, P. G., Rupley, J., Brekelbaum, E., Loh, G. H. and Black, B.: Matrix scheduler reloaded, *Proceedings - International Symposium on Computer Architecture*, ACM Press, pp. 335–346 (online), DOI: 10.1145/1250662.1250704 (2007).
[19] : Konata, https://github.com/shioyadan/Konata.