Vol.2021-ARC-244 No.36
Vol.2021-SLDM-194 No.36
Vol.2021-EMB-56 No.36
2021/3/26

ギャザー/スキャッタを効率化する Out-of-Step パイプライン

葛 毅[1]　依田 勝洋[1]　一場 利幸[1]　伊藤 真紀子[1]　吉川 隆英[1]　五島 正裕[2]

概要：A64FX や x86 プロセッサなどの汎用プロセッサ・コアの SIMD ユニットには，① SIMD の基本的な使用方法である水平計算によって生じる水平命令と，② ギャザー/スキャッタなどの不連続アクセス に，それぞれスケーラビリティの問題があり，SIMD 幅を拡大してもほとんど性能向上しない．計算の方法を垂直型にすれば水平命令を用いずに済むが，不連続アクセスを多用することになる．不連続アクセスの性能をスケーラブルにするためには，一次データ・キャッシュの多バンク化が避けられない．しかし従来のパイプラインでは，バンク衝突によるストール/フラッシュが毎サイクルのように発生するため，やはりスケーラビリティの問題に帰着する．本稿では，バンク衝突が起こってもストール/フラッシュを起こさない Out-of-Step バックエンド・パイプラインを提案する．机上計算により HPCG のカーネルを評価した結果，対 A64FX 比で，SIMD のビット幅を 1024 とした場合に 4.7 倍，2048 とした場合に 9.5 倍の性能向上を得られることが分かった．

1. はじめに

近年の汎用プロセッサ・コアは，主に SIMD 型のベクトル・ユニットのビット幅を拡大することによってベクトル性能を向上させてきた．「富岳」のプロセッサである A64FX [1] や最新の x86 プロセッサでは，SIMD ユニットの幅は 512 b，倍精度小数点数で 8 要素 分にもなる．

SIMD の幅の拡大によってベクトル性能の向上を図るアプローチは既に一般的なものとなっており，ARM SVE (Scalable Vector Extension) [2] や RISC-V Vector Extension [3] など，SIMD 幅が異なるマシンでも実行可能なバイナリを記述できる命令セットの普及も進んでいる．

しかし，SIMD の幅は，このまま 1024 b，2048 b と拡大できるようなものではない．Top500 に現れるようなピーク性能であれば，SIMD 幅の拡大によって向上させることができよう．しかし，そのようなピーク性能は，現実のアプリケーションの性能と乖離しているとの批判がある．

本稿でベンチマークとして採り上げる **HPCG** は，共役勾配 (Conjugate Gradient: CG) 法のベンチマークで，より現実に近いアプリケーションの性能を測るために提案されたものである [4]．コロナ禍の現在，咳による飛沫の飛び方やマスクの効果などを「富岳」を用いてシミュレーションした結果が公表されているが，このようなシミュレーションにおいても HPCG の結果に表れる能力が重要であると言われている [5]．したがって，HPCG の高速化は，一般の現実的なアプリケーションの高速化につながる．

しかし，HPCG の効率は，1 位の「富岳」で 3.6%，2 位の Summit で 2.0% に過ぎない [6]．我々は，この HPCG の著しく低い効率の原因について詳しく調査する過程で，それが SIMD ユニットの幅を拡大するアプローチの限界を示唆していることに気付いた．本稿では，その限界を打破し，HPCG の効率を 1 桁高めるコア・アーキテクチャと，そのポイントである **Out-of-Step バックエンド・パイプライン**について述べる．Out-of-step に対して，従来のパイプラインは **In-Step** と呼ぶことにする．In-/out-of-step の辞書的意味は，「歩調が合っている/いない」，「足並みが揃っている/いない」である．

本稿は，以下のように構成されている：2 章では，HPCG をはじめとする疎行列計算と従来 SIMD アーキテクチャの問題点についてまとめる．その議論を受けて，3 章と 4 章で，out-of-step 適用の対象となるコア・アーキテクチャについてまとめる．Out-of-step については 5 章で詳述する．Out-of-step について説明すると，なぜ今まで in-step だったのかとの質問をよく受ける．そこで 6 章では，その理由に関する考察を与えることにする．7 章では，HPCG のカーネル部を対象とした机上計算による性能評価の結果を示す．その他の関連研究については，8 章でまとめる．

[1] （株）富士通研究所
Fujitsu Laboratories Ltd.
[2] 国立情報学研究所
National Institute of Informatics

Vol.2021-ARC-244 No.36
Vol.2021-SLDM-194 No.36
Vol.2021-EMB-56 No.36
2021/3/26

2. 疎行列と SIMD のスケーラビリティ

本章では，HPCG をベンチマークとして，その低い効率の主要因となる，アプリケーション側とアーキテクチャ側，それぞれの特徴について述べる．

2.1 疎行列計算の特徴

HPCG [4] では，**疎行列** A の行と密ベクトル x の内積が計算の 8 割程度を占める．A の 1 行の非ゼロ要素数は 27 と非常に少なく，A は通常 Compressed Sparse Row (**CSR**) 形式で格納される．その結果，効率を低下させる以下のような特徴が生まれる：

短いベクトル長 内積計算のベクトル長は，26（対称ガウス・ザイデル法，SYMGS），または，27（疎行列ベクトル積，SPMV）と，著しく短い．

簡単のため，以下では 27 の場合について説明する．

ギャザー CSR により，A の行の読み出しは連続となる．一方で，A の非ゼロ要素に対応する密ベクトル x の要素の読み出しが，CSR のインデクスのリストを介した間接，不連続のロード，いわゆるギャザーとなる．

これら 2 つの特徴は，連立一次方程式の反復解法やグラフ処理などの疎行列の処理において一般的に現れるもので，HPCG に特有のものではない．A の 1 行の非ゼロ要素数 27 は HPCG のステンシルに起因する (3 次元の縦・横・斜めで $3^3 = 27$) が，一般には更に少ない（典型的には，縦・横で 7）．

これら 2 つの特徴は，それぞれ，

(1) 水平計算のスケーラビリティ と，

(2) 不連続ロード/ストアのスケーラビリティ

というアーキテクチャ側の問題を通して，効率を低下させる．以下，2.2，2.3 節でそれぞれについて述べる．

2.2 水平計算のスケーラビリティの問題

水平計算は，SIMD の基本的な使用方法である．しかし，スケーラビリティに問題があり，特に短いベクトル長でそのオーバーヘッドは深刻なものとなる．

水平計算

SIMD 命令では，v 個の要素をパックした SIMD レジスタをオペランドとすることにより，1 命令で v 回の演算を行うことが基本となる．したがって通常は，最内ループの方向を SIMD レジスタ内の要素の方向に一致させる．これを**水平計算**と呼ぶ．

図 1（左）に，512-bit SIMD（64 b × 8 要素）における水平型の内積計算の様子を示す．ここでは最も左（赤）の部分に注目されたい：

- まず，A の行と x それぞれの 0〜7 番目の要素を，それぞれ別の SIMD レジスタにロードして，対応する要素の積を求めることから始める．8 個の積は SIMD レジスタ ACC 0 に格納する，
- 8〜f 番目（以降）に対しては，積和命令を使用できる．積和の結果は，やはり ACC 0 に累積していく，ACC 0 の各要素には，8 要素おきの部分和が累積されることになる．
- そこで最後に，**水平加算命令**でそれら 8 個の部分和の総和を求める．

この計算方法には，以下の 2 つの問題がある：

(1) SIMD 幅に対するベクトル長の剰余

(2) 水平加算

いずれも，HPCG の 27 という短いベクトル長ゆえに顕在化する．以下，それぞれについて説明する．

(1) SIMD 幅に対するベクトル長の剰余

HPCG のベクトル長は 27 であり，A64FX の SIMD 幅 8 に対しては，$27 \bmod 8 = 3$ 要素の剰余が生じる．

SIMD 命令セットのマスク機能を用いれば，これらの剰余をスカラ・ユニットで計算する必要はなく，ベクトル長がある程度長い場合には問題にならない．

しかし，27 という短いベクトル長では，これだけで効率の上限は $27/32 = 84.4\%$ にまで低下することになる．

(2) 水平加算

最後に用いられる水平加算のような水平命令は，SIMD 命令セットの中でも最も変化の大きい命令の 1 つである．

水平命令は，最初期からサポートされていた訳ではなく，SIMD の幅 v が拡大されて一般的なベクトル処理に用いられるようになると，必要性が認められ導入された．

しかし，v が更に拡大されるにつれ，水平命令をナイーブに実装することは困難になってきた．通常の垂直命令のレイテンシが O(1) であるのに対して，水平命令のレイテンシは $O(\log v)$ であり，v の拡大にともなって，それらの差は大きくなる．その結果，垂直命令を基本とするパイプラインの中に直接水平命令を実装することの非効率性が許容できなくなってきたのである．

そこで最近では，ペアワイズに計算する命令を用いる方式が主流となっている．Intel SSE/AVX のように ISA で対応するもの [7] と，A64FX のように μOP で対応するもの [2], [8] があるが，実行の様子はほとんど変わらない．

水平加算のオーバーヘッドは，ベクトル長が十分に長い場合には問題にならないが，HPCG の 27 という短いベクトル長では重大な性能低下要因となる．たとえば A64FX では，8 個の部分和を求める部分より FADDV 1 命令の実行時間の方が長く，効率は更に半減する（7.2 節）．

水平計算と SIMD 幅のスケーラビリティ

水平計算の (1) ベクトル長の剰余，(2) 水平加算，いずれの問題も，SIMD 幅が増大するにつれて悪化する．すなわち，SIMD の基本的な使い方である水平計算には，SIMD 幅のスケーラビリティの問題があると言える．

Vol.2021-ARC-244 No.36
Vol.2021-SLDM-194 No.36
Vol.2021-EMB-56 No.36
2021/3/26

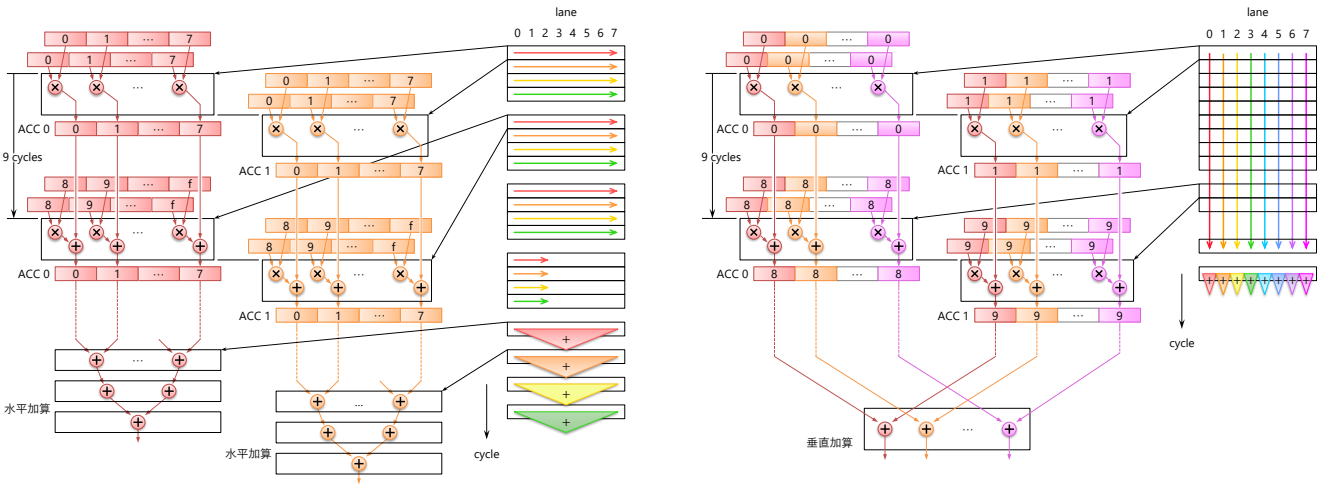


図 1　512-bit SIMD における HPCG の水平型（左）と垂直型（右）の内積計算

2.3　不連続ロード/ストアのスケーラビリティの問題

通常，不連続アクセスのスループットは，SIMD 幅に対してスケーラブルではない．連続アドレスへのロード/ストアはベクトル・ユニットと同等のスループットとなる一方で，ギャザー/スキャッタなどの不連続アクセスはスカラ・ユニット程度のスループットにとどまる．たとえば A64FX では，連続アドレスへの場合には，SIMD レジスタの幅に相当する 8 要素を 1 サイクルでロードできる．一方ギャザーの場合には，combined gather という工夫がなされているものの，1 サイクルあたり最大 2 要素にとどまる．連続アドレスへのロードとギャザーのスループットの比は 4〜8 倍にもなる [1].

これは，一次データ・キャッシュ（**L1D**）のポート数の制約による．たとえば A64FX の場合，1 つのアドレスで指定される連続 8 要素にアクセス可能なポートが，ロード/ストア・ユニットあたり 1 本しかない．ギャザーでは，不連続の，すなわち，異なるアドレスで，そのポートを逐次的に使用することになる．したがって，単に SIMD ユニットの幅を拡大するようなことは，不連続アクセスの性能向上にほとんど寄与しない．

多バンク L1D

不連続アクセスのスループットを幅 16〜32 要素のベクトル・ユニット相応へと向上させるためには，16〜32 個の異なるアドレスに並列にアクセスできる必要がある．メモリ・セルの面積はポート数の二乗に比例するため，フルポート，すなわち，16〜32 ポートのメモリを用いることは，回路面積の観点から現実的ではない．現実的な方法としては，疑似マルチポートとしてのマルチバンク化以外には知られていない．この場合，2〜4 バンク程度ではなく 32〜64 バンク程度もの **多バンク化** が必要となる．

多バンク化された L1D では，主にデータ・アレイを多数のバンクに分割し，アドレスをインターリーブして割り付ける．その結果，連続アドレスへのアクセスは別のバンクへと分散され，並列にアクセスすることができる．不連続アクセスに対しては，**バンク衝突** (bank conflict) が発生するものの，分散の度合いに応じたスループットを提供することができる．

多バンク L1D とバンク衝突

しかし実際には，不連続アクセスに対するバンク衝突の発生確率は極めて高い．バンク衝突の確率は，アクセス先のバンクがランダムであるとすると，誕生日問題と同形となる．誕生日問題では，n 人中に同じ誕生日（$b = 365$ 通り）のものがいる確率を；バンク衝突 では，n 個のアクセス中に同じバンク（b 通り）へのものがいる確率を考える．その確率は，以下の式で与えられる：

$$P(b,n) = 1 - \frac{b-1}{b} \times \frac{b-2}{b} \times \cdots \times \frac{b-(n-1)}{b}. \quad (1)$$

たとえば A64FX では，1 サイクルに $n = 16$ 個のアドレスに対してギャザーを行うため，バンク数をその倍の $b = 32$ としよう．この場合，バンク衝突の発生確率は $P(32,16) \simeq 99.0\%$ になる．

バンク衝突とパイプライン・ストール/フラッシュ

従来のパイプラインでは，バンク衝突に対して**パイプライン・ストール/フラッシュ**が避けられない．99% もの確率では，ストール/フラッシュは毎サイクルのように発生することになり，多バンク化の効果の大部分は失われることになる．バンク数をたとえば 1024 まで増やせば $P(1024,16) \simeq 11.1\%$ になるが，現実的ではない [9].

3.　コア・アーキテクチャの設計方針

前章での議論を踏まえ，以下の 2 つを基本的な方針とするコア・アーキテクチャを提案する：

(1) 垂直計算
(2) 多バンク L1D と Out-of-Step パイプライン

Vol.2021-ARC-244 No.36
Vol.2021-SLDM-194 No.36
Vol.2021-EMB-56 No.36
2021/3/26

3.1 垂直計算

水平計算（演算レイテンシの隠蔽）

2.2 節では，図 1（左）に示した HPCG の水平型内積計算について説明した．実際には，演算レイテンシの隠蔽のため次のサイクルには次の内積の命令を実行する．同図では，別の内積計算を別の色（赤，橙，…）で表している．たとえば A64FX の浮動小数点積和命令のレイテンシは 9 サイクルあるため，9 個の内積をインターリーブ実行する．

垂直計算

2.2 節ではまた，(1) ベクトル長の剰余 と (2) 水平加算が問題であることを示した．これらの問題は，垂直型にすることで解消される．

同図 1（右）に，垂直型内積計算の様子を示す．**垂直計算**では，最内ループの方向を，SIMD レジスタ内の要素の方向ではなく，サイクル経過の方向に一致させる．あるいは，SIMD の i 番目のレーンが i 番目の内積を計算すると考えた方が分かりやすいかもしれない．

水平型と同様，垂直型でも，演算レイテンシの隠蔽のために，9 要素おきに部分和を計算する．ただし水平型とは異なり，9 つの内積のではなく，1 つの内積の 9 要素ごとの積和を，インターリーブ実行することになる．

その途中結果は，9 本の SIMD レジスタに累積され，やはり水平型と同様，最後にそれらの総和を求めるが，その部分も水平加算ではなく，9 本の SIMD レジスタ間の垂直加算となる．

水平型の 2 つの問題は，垂直型にすることによって以下のように完全に解消される：

(1) SIMD 幅に対する剰余　垂直型ではベクトル長 27 の内積は 27 回のループによって計算されるため，水平型のような SIMD 幅の剰余は生じない．

(2) 水平加算　内積計算はレーン内で完結するため，水平加算等の水平命令は必要ない．

A64FX をモデルとした場合，水平型と垂直型の効率の比は約 2.6 倍にもなる（7.2 節）．そして，SIMD 幅が更に拡大されると，この比は更に大きくなる．

垂直計算の課題

ただし従来のアーキテクチャでは，垂直計算を行った場合かえって性能が悪化することになる．図 1（右）で 1 つの SIMD レジスタに異なる色の要素がパックされているように，垂直型では 1 つの SIMD レジスタに，1 つの内積の 8 要素ではなく，8 つの異なる内積の要素がパックされることになる．その結果，水平型で連続アドレスであった A へのアクセスまでもが等間隔，すなわち，不連続となる．2.3 節で述べたように，従来のアーキテクチャの不連続ロード/ストア性能は劣悪である．

結局，水平計算のスケーラビリティの問題もまた，不連続ロード/ストアのスケーラビリティの問題へと行きつくことになる．

3.2 多バンク L1D と Out-of-Step パイプライン

2.3 節で述べたように，L1D を多バンク化すればギャザー/スキャッタなどの不連続アクセスの性能をベクトル・ユニット相応に高めることができるが，通常のパイプラインでは高頻度で発生するバンク衝突に起因するパイプライン・ストール/フラッシュによる性能低下が避けられない．

そこで本稿では，バンク衝突の確率を下げるのではなく，バンク衝突が起こってもパイプライン・ストール/フラッシュを引き起こさない **Out-of-Step バックエンド・パイプライン**を提案する．

ストーリを逆にたどると，以下のようになる; すなわち，out-of-step によって MBL1D が実現可能になると，ギャザー/スキャッタを含む不連続アクセスの性能がスケーラブルとなり，不連続アクセスを多用する垂直計算も採用可能になる．その結果，HPCG をはじめとする疎行列計算の短いベクトル長とギャザーの問題が解消される．

次章ではまずコア・アーキテクチャについてまとめ，out-of-step については 5 章で改めて述べる．

4. コア・アーキテクチャの詳細

本章では，次章（5 章）で Out-of-Step バックエンド・パイプラインについて詳述する前に，その適用の対象となるコア・アーキテクチャについてまとめる．

図 2 に，提案コアの構成例のブロック図を示す．フロントエンドからスケジューラまでは，通常の out-of-order コアと同じものと考えてよい．特徴は，バックエンドの，以下の 3 点にある：

(1) レーンとレーン・グループ
(2) 多バンク化された L1D
(3) 多バンク化されたレジスタ・ファイル

以下，4.1〜4.3 節で，それぞれについて述べる．

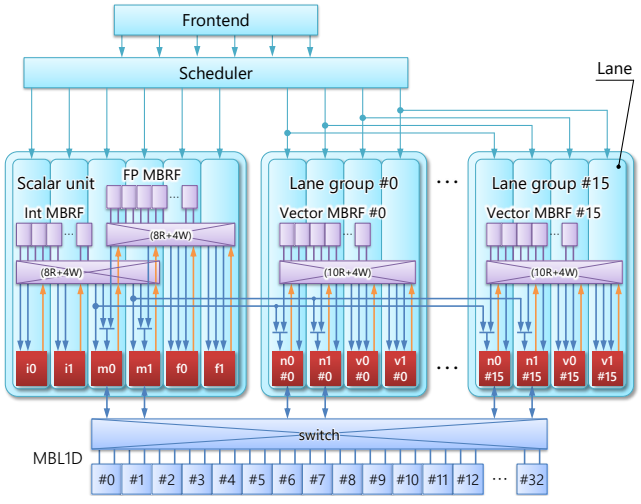


図 2　提案コア構成例のブロック図

Vol.2021-ARC-244 No.36
Vol.2021-SLDM-194 No.36
Vol.2021-EMB-56 No.36
2021/3/26

4.1 レーンとレーン・グループ

提案コア では，スケジューラの 1 つの発行ポートと，その発行ポートに接続された演算器と，その演算器に接続されたレジスタ・ファイルのポートを合わせて**レーン** (lane) と定義する．SIMD ユニットは，単一の幅広のユニットではなく，64 b ごとに**レーン・グループ** (lane group) に分割される．同図では，1024 b の SIMD ユニットが`#0`〜`#15` の 16 のレーン・グループに分割されている．

レーン・グループは，いくつかのレーンを持つことになる．同図では，2-issue のメモリ・ユニット (`n0`，`n1`) と 2-issue のベクトル演算ユニット (`v0`，`v1`) の計 4 つのレーンを持つ．レーン・グループは均質であり，スケジューラからの SIMD 命令はすべてのレーン・グループの各レーンに複製して発行される．スカラ・ユニットも，非均質なレーン・グループと考えてよい．

次章 (5 章) で詳述する out-of-step の制御の対象はレーンである．Out-of-step の機能により，レーン，そして，レーン・グループは，ある程度独立に動作する．

レーン間，レーン・グループ間のデータの授受

レーン間，レーン・グループ間のデータの授受は，通常の SIMD ユニットと変わらない．通常の SIMD ユニットでも，別の位置にある 64 b 間のデータの授受はシャフル・パーミュテーションなど，一部の命令によってのみ行われることに注意されたい：

- バイパスは，レーン・グループ内のレーン間でのみ行われる．
- レーン・グループ間のデータの授受は，シャフル・パーミュテーション命令や一部の特殊な命令によって行う．図には描かれていないが，`v0 #0`〜`#15`，`v1 #0`〜`#15` はそれぞれ，それらの命令用のネットワークによって接続されている．

4.2 多バンク L1D

次章（5 章）で述べる Out-of-Step の機能により性能への影響は緩和されるとはいえ，バンク衝突確率はより低い方が望ましい．そのため，MBL1D の総ポート数はアクセス数の 2 倍程度を想定する．図 2 では，アクセス数は 34 であるので，ポート数は 64 程度となる．

各バンクは 2 ポートとすると，バンク衝突と回路面積のバランスがよいと考えられる．その場合，64 ポートに必要となるバンク数は 32 となる．

4.3 多バンク・レジスタ・ファイル

L1D と同様に，レジスタ・ファイル (**RF**) も多バンク化することができる (**MBRF**)．

L1D の多バンク化は，数ポートの L1D をベースラインとすると，性能向上のための施策であると言える．一方 RF の多バンク化は，フルポートの RF がベースラインとなるので，面積・エネルギー削減のための施策と位置付けられる．

RF の面積・エネルギーは大きく，多バンク化の効果は非常に高い．山田らは，スカラ・ユニットの整数・浮動小数点 RF に多バンク化を施している．詳細な評価の結果，RF の回路面積/消費エネルギーをフルポートの 22.6%/17.0% にまで削減できるとしている [9], [10]．

バンク衝突のため，フルポートに比べると性能は低下しはするが，次章で述べる out-of-step の機能により，低下の度合いは低く抑えられると期待できる．

MBRF の具体的な構成は，今後シミュレーション等によって決定する必要がある．

4.4 構成比較

提案コアの位置づけを明らかにするため，本章の最後に既存のベクトル処理アーキテクチャと比較しよう．

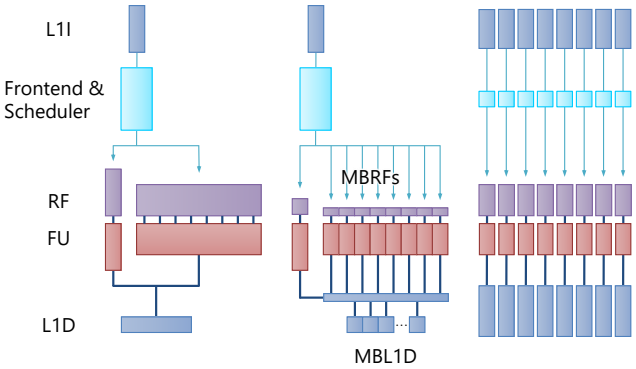
図 3 は，以下のコアを模式的に表したものである．図中の矩形の面積は，実際の回路面積をある程度反映している：

(a) A64FX など，従来の，SIMD ユニットを持つ out-of-order スーパスカラ・コア．
SIMD ユニットは 1 つの幅広の演算器として動作する．

(b) 提案コア．L1D と RF を多バンク化される．
次章で詳述する out-of-step によりバックエンド・パイプラインはレーンごとにある程度独立に動作する．

(c) In-order もしくはスカラのスモール・コアをより多く搭載したメニー・コア．図では，ベクトル演算性能が同一になるように正規化してある．
コアごとに完全に独立に動作する．

(a) に対して (b) は，不連続アクセスが高速で，垂直計算が可能である点で優れる．面積は，MBRF を採用すれば大きく減少する．



評価項目	(a)	(b)	(c)
不連続 ロード/ストア	×	○	◎ バンク衝突 なし
垂直計算	×	○	◎ バンク衝突 なし
面積	○	◎	× L1I/D が コア ごとに必要

図 3 方式の比較

Vol.2021-ARC-244 No.36
Vol.2021-SLDM-194 No.36
Vol.2021-EMB-56 No.36
2021/3/26

(a), (b) に比べ (c) は，不連続アクセスや垂直計算などの性能観点ではよりよい．垂直計算はスモール・コアではむしろ標準的な計算方法である．また，不連続アクセスによる性能低下はほとんどない．

しかしそれは，L1D をはじめとする資源をコアごとに複製していることによるものであり，面積効率は大きく劣る．一般には，スモール・コアは面積効率に優れると考えられがちであるが，図 3 からも読み取れるように，以下の点まで考えると正しくない：

- ビッグ・コアの制御部の面積はたしかに大きいが，SIMD ユニット幅の拡大により大幅に緩和される．
- メニー・コアは，特に L1I/D が複製となる．HPC アプリケーションでもコア間（SIMD ではレーン・グループ間）で共有可能な命令・データの量は多く，複製は無駄が多い．

5. Out-of-Step バックエンド・パイプライン

本章では，前章までで述べた提案コア・アーキテクチャ実現のポイントである Out-of-Step バックエンド・パイプラインについて詳しく述べる．

5.1 In-Step パイプライン

Out-of-order コアの命令パイプラインは，フロントエンド・パイプライン (**FEP**)，スケジューラ，バックエンド・パイプライン (**BEP**) に分解することができる．

通常，FEP/BEP 内の命令間の相対位置は，投入時から変更されることがない．この性質を **In-Step** (**InS**) と定義する．InS 命令パイプラインでは，命令の相対位置の変更は，FEP/BEP の間のスケジューラにおいて 1 回のみ行われることになる．

In-Step バックエンド・パイプラインの問題点

InS BEP 問題点は，「不測の事態」に弱いことである．

スケジューラでは想定し得ない，キャッシュ・ミスやバンク衝突などの「不測の事態」は，避けることができない．投機が普通になった現在のパイプラインにおいては，「不測の事態」は，キャッシュ・ヒット/ミス予測などの予測に基づく投機的なスケジューリングにおける予測ミス，**スケジューリング・ミス**として一般化されよう．

InS BEP では，スケジューリング・ミスが 1 つでも発生すると，その影響は BEP 全体に及ぶ．InS BEP は，ある命令が先に進むことができなくなったとしても，スケジューラによって与えられた命令間の相対位置を変更しない．変更せずに一貫性を維持する方法は，一般に，全か無のいずれかである：

全：ストール　スケジューラによるスケジューリングの結果を完全に維持するため，当該命令が進めるようになるまで BEP 全体を停止する．

無：フラッシュ・再スケジューリング　スケジューラによるスケジューリングの結果を完全に捨て，スケジューリングからやり直す．

いずれも，影響は BEP 全体におよぶ．そのため，特に並列度の高いパイプラインの場合，1 回あたりのペナルティは非常に大きい．

InS BEP のこの性質が今まであまり問題とされなかったのは，スケジューリング・ミスの確率が十分に低かったためである．しかし，2.3 節で述べたように，MBL1D におけるバンク衝突の発生確率は 1 に近い．InS のまま，これに対処することは困難である．

5.2 Out-of-Step バックエンド・パイプラインの概要

Out-of-Step (**OoS**) BEP では，スケジューラのスケジューリング結果を維持せず，BEP 内で独自に命令の相対位置を変更する．一方，スケジューラは，基本的には InS と変わらない．すなわち，OoS BEP は命令の相対位置を独自に変更するが，スケジューラは位置関係が変更されることを織り込む必要はない．

図 4 に，InS/OoS BEP の概念図を示す．

同図（左）の InS では，全てのレーンにまたがってステージ間に単一エントリのパイプライン・レジスタが存在する．投入された命令は，互いの相対位置を変えることなく，すべてのレーン，すべてのステージを揃って進むことになる．さもないと，いずれかのパイプライン・レジスタで書き潰しが発生することになる．

FIFO バッファ

一方，同図 4（右）の OoS では，RF リード前，演算ユニット前，RF ライト前のパイプライン・レジスタを，それぞれ 2〜4 エントリ程度の **FIFO バッファ**に変更している．

セクションとセクション・ストール

各レーンは，バッファによって他の部分と分離 (decouple) された**セクション**へと分割される．あるセクション

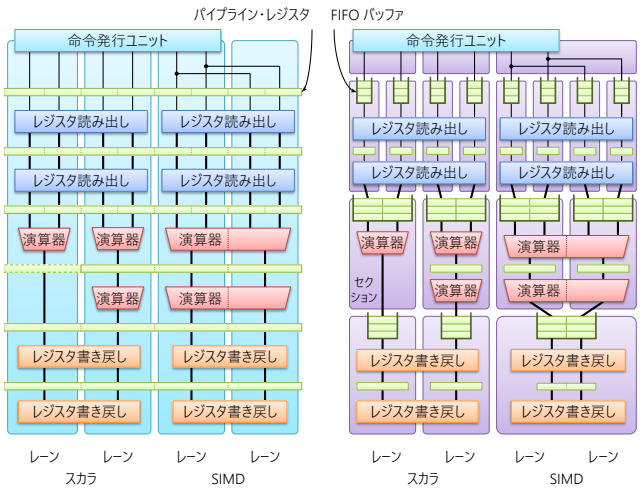


図 4　In-(左)/out-of-step(右) バックエンド・パイプライン

Vol.2021-ARC-244 No.36
Vol.2021-SLDM-194 No.36
Vol.2021-EMB-56 No.36
2021/3/26

で，MBL1D または MBRF でバンク衝突などのスケジューリング・ミスが起きた場合には，当該セクションは先に進むことができない．これを**セクション・ストール**と呼ぶ．

あるセクションでセクション・ストールが発生しても，BEP 全体がストール/フラッシュすることはない．バッファの分離機能により，同一レーンの上流・下流のセクションや，別のレーンのセクションは動作を続けることができる．

その結果，命令間の相対位置はスケジューラによるスケジューリングの結果から変化することになる．いわば，全でも無でもなく，修正されると言える．

次節ではまずこの OoS の効果について概説し，相対位置が変わっても命令が正しく実行される仕組みについては 5.4 節以降で述べる．

5.3 Out-of-Step バックエンド・パイプラインの効果

図 5 に，6 バンクの L1D に対して，毎サイクル，3 つのレーンからのアクセスがランダムに行われる様子を示す．図中，ボールの中の数字 1〜3 がレーンを表す．ボールと各サイクルの背景の色はアクセスを開始したサイクルごとに赤，緑，青，赤，…としている．したがって，命令間の相対位置を変えない InS では，ある色の背景のサイクルに別の色のボールがあってはならない．また OoS でも，各サイクルに同じ数字のボールが 2 つあってはならない．

サイクル 1〜4 の動作は以下のようになっている：

1. レーン 1〜3 のアクセスがバンク 4〜6 に分散しており，バンク衝突は起こっていない．
2. レーン 1〜2 のアクセスがバンク 2 に集中してバンク衝突が起こっている．その結果，
3. レーン 2 からのアクセスがここで行われている．InS ではストールにより，そのためのスロットを作っている．一方 OoS では，レーン 2 の該当セクションがストールするだけで，その他のレーン 1 と 3 は実行を継続できる．
4. OoS では，前のサイクルでセクション・ストールを起こしたレーン 2 のアクセスがこのサイクルで行われている．一方 InS では，パイプライン全体がストールしたため，レーン 2 のみならず，1，3 のアクセスもこのサイクルで行われている．

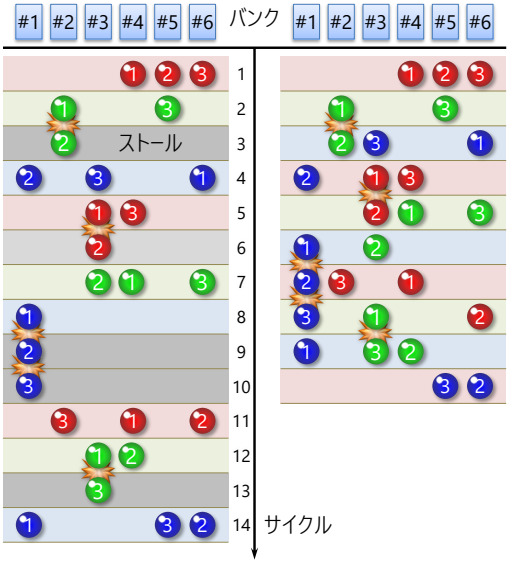


図 5 In-（左）/Out-of-Step（右）BEP における MBL1D アクセス

式 (1) より，バンク衝突の確率は $P(6,3) \simeq 44\%$ となり，同図でもその程度のバンク衝突が発生している．バンク衝突の発生自体は InS/OoS で変わらないが，その結果は InS/OoS で以下のように異なる：

InS バンク衝突の度にストールが発生し，実行時間も 1.44 倍程度になっている．

OoS 資源の許す最早のタイミングでアクセスが行われることになる．

2.3 節で述べたように，実際の MBL1D ではバンク衝突の発生確率は $P(64,32) \simeq 99.0\%$ 程度にもなる．更に，動図（左）ではサイクル 8〜10 にみられるような三重以上の衝突が発生する．その結果，InS の性能低下は 2 倍以上になる．

5.4 Out-of-Step バックエンド・パイプラインの詳細

本節以降で，OoS BEP の詳細について述べる．以下ではまず，5.2 節では触れなかった全体的な振る舞いについて述べる．相対位置が変わった命令を正しく実行する仕組みについては次節以降で述べる．

FIFO バッファ

各バッファが FIFO であることは重要で，レーン内での命令の追い越しは想定されない．一方レーン間では，命令の追い越しは起こる．

バッファ・フル

セクション・ストールは，直下のバッファがフルになった場合にも起こる．前節で述べた例は，バッファ・フルが起こらなかった場合のものである．

あるセクションのストールが長期間に及ぶと，バッファ・フルを介してストールは上流へと伝わり，最終的にはそのレーンへの命令発行が停止することになる．

スケジューラの変更点

このバッファ・フルに起因するレーンごとの発行の停止が，スケジューラの唯一の変更点である．

この変更すら必須ではなく，直下のバッファのうちいずれかがフルであった場合にスケジューラ全体を停止してもよい．バッファのサイズが十分であれば．大きな性能低下はないと思われる．

レーンごとの発行停止の効果は，今後シミュレーション等によって検証する必要があろう．

Vol.2021-ARC-244 No.36
Vol.2021-SLDM-194 No.36
Vol.2021-EMB-56 No.36
2021/3/26

Out-of-Step 動作を実現する仕組み

セクション・ストールによって相対位置が変わった命令を正しく実行するには，以下の 2 つの仕組みが必要になる：

二次スケジューラ　実行ステージの手前のバッファでは，ソース・オペランドを待ち合わせる．

バイパス制御　位置関係が変わってしまっ命令間で実行結果を正しくバイパスする．

これら，特に後者が低コストで実現できるとの発見が，OoS のポイントである．以下，5.5 節と 5.6〜5.8 節でそれぞれについて述べる．

5.5　二次スケジューラ

InS BEP では，実行ステージにたどり着いた命令のソース・オペランドは，必ず既に計算されていて，BEP 中に存在している．それらは，既に RF から読めたか，バイパスから取得できる．より正確には話は逆で，InS では，そうなるように，命令はスケジュールされ，パイプライン中を in-step に進むのである．

一方 OoS BEP では，セクション・ストールのため，実行ステージにたどり着いた命令のソース・オペランドが揃っているとは限らない．そこで，実行ステージ前のバッファはソース・オペランドの待ち合わせを行う機能を持つ．

その結果このバッファの機能・構造は，最初期のスーパスカラ・プロセッサのリザベーション・ステーションに似たものとなる．本来のスケジューラを一次とすると，このバッファは **二次スケジューラ** と言える．

ただし，out-of-order の一次スケジューラなどと比べると，二次スケジューラのコストは無視できるほど小さい：

規模　Out-of-order の一次スケジューラは，性能のため，ある程度の統合 (unified) が必要で，エントリ数は数十〜百数十程度になる．

二次スケジューラは，演算器ごとに分散 (separate) してよく，1 つあたり 2〜4 程度でよい．

セレクト　Out-of-order 一次スケジューラは，優先順位に従って 2〜3 個の命令を選択する複雑なアービタを必要とする．

二次スケジューラは，FIFO であるため，先頭エントリのソース・オペランドがレディであるかチェックするだけでよい．

ウェイクアップ　Out-of-order の一次スケジューラのウェイクアップ・ロジックは，コアの中で最も複雑なロジックの一つである [11]．

二次スケジューラのウェイクアップは，次節以降で述べるバイパス制御そのものとなり，独自のウェイクアップ・ロジックは必要としない．

バイパス制御の方式

前述のとおり，特にこのバイパス制御が低コストに実現できるとの発見が，OoS のポイントである．以下，(1) 分散 CAM 方式 と (2) 依存行列方式 を示す．前者は，データ駆動の考え方をナイーブに実装したもので，コストはやや高いが理解は容易であり，いわば OoS が実現可能であるとの存在証明となろう．実際には，依存行列方式を用いる．

5.6　分散 CAM 式バイパス制御

図 6 に，分散 CAM を用いたバイパス制御方式を示す．

図中，FIFO Ctrl は，FIFO の制御を行う組み合わせ回路のブロックである．エンキュー/デキューの指示を受けて，FIFO バッファのペイロードであるレジスタに対して，現状態 (curr) から次状態 (next) を求める．

バイパスは一般に，以下のような回路構成を採る：すなわち，バイパス元は実行結果をバイパスへとブロードキャストし；バイパス先のセレクタで必要な実行結果を取り込む．

この構成を基に，分散 CAM 方式は，実行結果にタグを付加することによって，データ駆動的にデータの授受を実現する：すなわち，バイパス元は，実行結果と並列にデスティネーション・タグ（図中，tagD）をブロードキャストする．バイパス先ではそれをソース・タグ（同 tag1）と比較，一致すればその実行結果を取り込めばよい．

なお，図 6 では，tag1 レジスタの入力，すなわち，FIFO Ctrl の next を比較に用いて，エントリの内容が更新される場合とされない場合の両方に対処している．この回路構成は，次節で述べる依存行列方式においても重要である．

分散 CAM 方式は，タグをキーとする CAM によって二次スケジューラを実現するものである．この場合の二次スケジューラの回路構成は，最初期のスーパスカラ・プロセッサの CAM 式のリザベーション・ステーションにおい

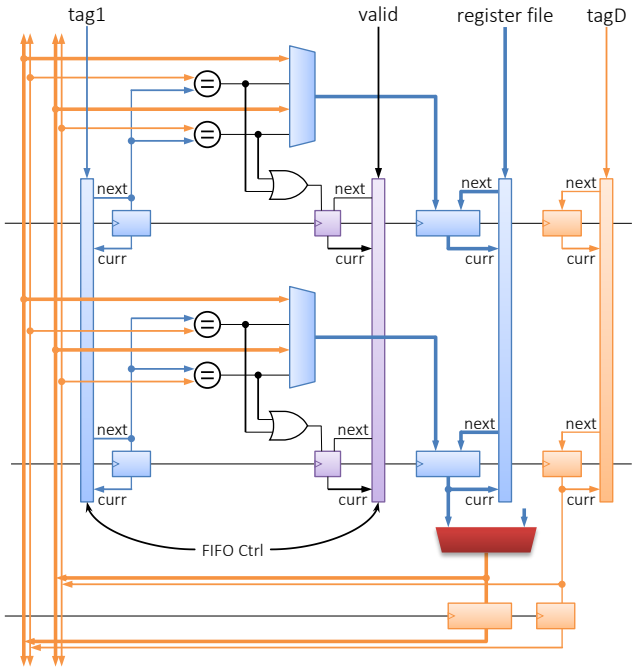


図 6　分散 CAM によるバイパス制御

Vol.2021-ARC-244 No.36
Vol.2021-SLDM-194 No.36
Vol.2021-EMB-56 No.36
2021/3/26

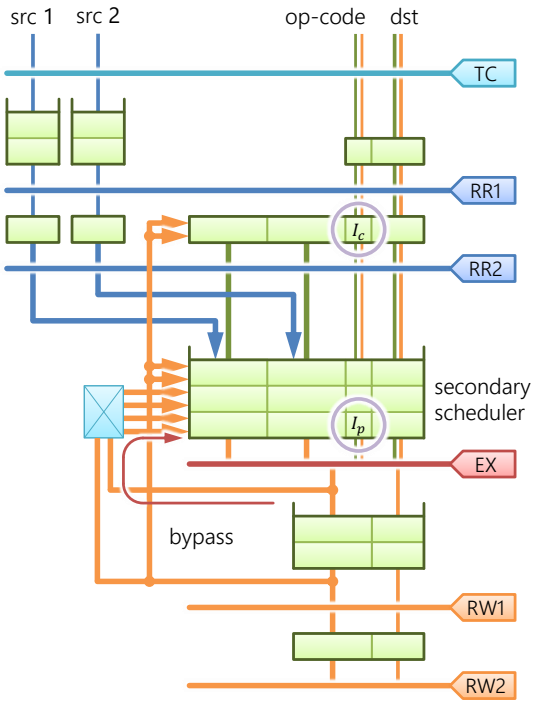


図 7 Out-of-Step バックエンド・パイプラインのブロック図

て，セレクトの方式を FIFO としたものとなる．

分散 CAM 方式では，64 b の実行結果に 6 b〜8 b 程度のタグを付加するため，データパスのビット幅が 1 割前後増加することになる．

5.7 In-Step 依存行列式バイパス制御

五島らは，CAM ではなく命令間の依存関係を表す行列を RAM に格納する一次スケジューラを提案している [11]．同様に，二次スケジューラとバイパス制御も，依存行列を用いて CAM を RAM に変更することができる．

OoS の依存行列式バイパス制御を考案した後で，通常の InS におけるバイパス制御も依存行列によって説明できると気付いた．そこで，以下ではまず通常の InS BEP におけるバイパス制御を依存行列を用いて説明する．ここでは，特に新しいことを提案する訳ではない．その後，次節（5.8 節）において，InS の拡張として OoS の依存行列を説明する．InS における依存行列を理解できれば，OoS 化のために付け加えるべきことは多くない．

In-Step バックエンド・パイプラインと依存行列

図 8（上）に，InS BEP のバイパス制御を行う依存行列を示す．この行列は，**図 7** のブロック図に対応している．このブロック図では，命令を格納するパイプライン・レジスタやバッファに着目するため，各ステージのロジックを太線によって表している．なお，このブロック図は OoS の説明のため，一部のパイプライン・レジスタがバッファに置き換えられている．本節では，単一エントリのパイプライン・レジスタとして理解されたい．

同図 7，8 は，**図 9** のパイプライン図に示した状況を表

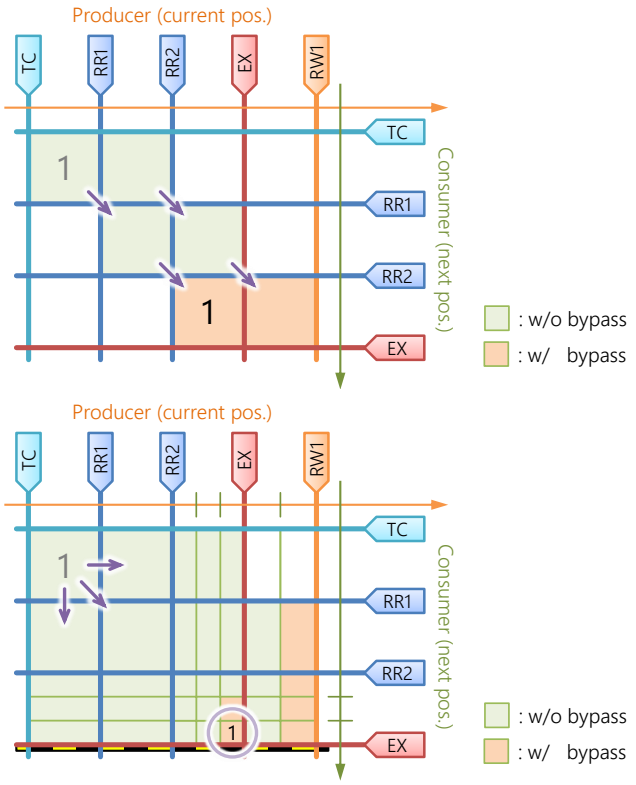


図 8 In-（上）/Out-of-Step（下）依存行列

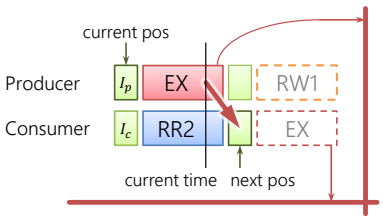


図 9 図 7，8 のパイプライン図

している．依存関係にある命令 I_p と I_c が連続するサイクルに発行され，現時刻においては I_p がまさに実行されている．I_c はその手前のパイプライン・レジスタにあり，次のサイクルには現在 I_p が置かれているエントリに移るとする．したがって，実行中の I_p の結果は，バイパスを通して現在 I_p が置かれているエントリのソース・フィールドに送られる．次のサイクルには，I_c はこの結果をソースとして実行される．

依存行列

依存行列の行/列は，生産者/消費者の命令が格納されるパイプライン・レジスタ（やバッファのエントリ）に対応する．前節では，消費側では FIFO Ctrl の next を使用する設計について触れた．同様に，依存行列においても，生産者は現在の位置を表すのに対して，消費者は次のサイクルの位置を表すことにする．同図においてセットされている下段の 1 は：

生産者 現サイクルに，EX ステージの直前のエントリに格納されて（実行されて）いる I_p と，

Vol.2021-ARC-244 No.36
Vol.2021-SLDM-194 No.36
Vol.2021-EMB-56 No.36
2021/3/26

消費者　次サイクルに，同じ EX ステージの直前のエントリに格納される I_c が
依存関係にあることを示す．

したがって，依存行列は以下の性質を持つ：

- レーン・グループ内のバイパスを行うソース/デスティネーションのペアごとに行列が 1 つ存在する．
- 行/列は消費者/生産者の命令のエントリの一部と考えてよい．各マスは，同時に生産者/消費者の双方のエントリに属す．
- あるソース・オペランドの生産者は高々 1 つであるから，各行（ソース）でセットされる列（生産者）も高々 1 つ，すなわち，**ワンホット**である．

依存行列の処理

依存行列の処理は以下のように進む：

生　成　InS BEP では，命令の発行直後に，そのソースのタグを先行する命令のデスティネーションのタグと比較して，依存関係を求める．図 7 のブロック図中では，タグ比較を TC としている．

図 8（上）の依存行列では，タグ比較の結果，消費者 I_c に対応する 1 番上の 1 行が生成される．2 つあるマスのうち，左側に 1（灰色）がセットされ，2 つ前ではなく 1 つ前の I_p に依存することが示される．

シフト　InS BEP では，命令間の相対位置が変わらず，生産者 I_p も消費者 I_c も 1 サイクルに 1 ステージずつ進む．したがって，依存関係の各マスは，1 サイクルごとに必ず右下へとシフトする．

使　用　依存行列は，生産者が実行ステージを通過するサイクル以降，バイパスを制御するために使用される．各行は，複数のバイパスからの実行結果を選択するセレクタのワンホット選択入力そのものになっている（図 6 参照）．

同図においてセットされている下段の 1 は，現サイクルに EX ステージの直前に格納されて実行されている I_p の結果を，次サイクルに I_c が格納される，同じ EX ステージの直前のエントリのソース・フィールドに送ることを指示する．

In-Step 依存行列の実装

右下方向にのみシフトするため, InS 依存行列は, 2b×3 段のシフト・レジスタで実装できる．実際の設計では，タグ比較の結果をパイプラインに沿って進ませてバイパス制御に用いると解すのが普通であるが，回路的には等価になる．

5.8　Out-of-Step 依存行列式バイパス制御

前節で述べた InS BEP における依存行列を拡張する形で，OoS 依存行列を説明する．

Out-of-Step バックエンド・パイプラインと依存行列

図 8（下）に，OoS の依存行列を示す．同図では，一部のステージ間では，正方形を複数のマスへと分割することによって複数エントリのバッファを表現している．たとえば図中，〇を付した 1 のあるマスは，3 エントリの二次スケジューラに対応して 3×3 に分割されている．5.9 節で説明するが，橙で塗られているマスは，対応するバイパスがあることを示す．

Out-of-Step 依存行列の処理と実装

依存行列は，レーンの命令の進行状況に合わせて右/下にシフトされる．生産者/消費者のいるセクションが独立にストールし得るから，依存関係を表す 1 は，とどまるか，右か下か右下に移動する．

したがって，行列全体は 2 次元のシフト・レジスタとして実装される．これは，図 6 に示した FIFO Ctrl を縦横に組み合わせると実現できる．

5.9　バイパスの削減

OoS BEP に特徴的な最適化として，バイパスの削減がある．このことは，依存行列上で考えることができる．

依存行列とバイパス

図 8 の依存行列の 1（黒）のあるマスについて考える．このマスは，back-to-back に実行される命令間のバイパスに対応している：

InS BEP　仮にこのバイパスがなければ，I_c はバイパスから結果を受けることなく EX ステージに進んでしまうことになる．

OoS BEP　仮にこのバイパスがないとしても，I_c は，EX ステージに進むことなく，二次スケジューラで待たされる．行列上では，1 がこれより下に進むことはない（行列下部の黄黒のバー）．

I_c が待たされている状態で I_p がパイプライン中を進むと，1 は右に移動していく．1 が行列の右端から抜けていくことは，I_p が実行結果を渡すことなくパイプラインからいなくなってしまうことを意味する．逆に，たとえば右端の橙で塗られているマスにさえバイパスがあれば，必ずそれらのバイパスから結果を受けることができる．その他のバイパスは，正しい実行という観点からは必要ない．

バイパス削減の実際

ただし性能のためには，図中で 1 があるマス周辺の，back-to-back で実行される時に使用されるバイパスはあった方がよい．図 8（下）のように，2 列に限定するとすると，バイパスのデータパスは，レーンあたり 2 本で済む．図 7 のブロック図のバイパスと見比べられたい．

実際にどの程度のバイパスが省略できるかは，今後シミュレーション等によって確かめる必要がある．

6.　考察

Out-of-step について説明すると, なぜ今まで in-step だったのかとの質問をよく受ける．そこで本章では，その理由に関する考察を与えることにする．

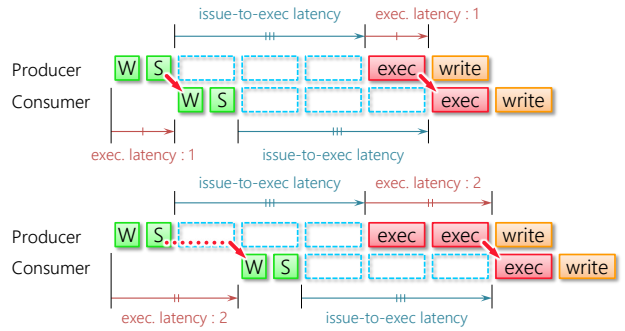


図 10 InS BEP におけるスケジュール〜実行までのパイプライン

バックエンド・パイプラインの設計規約

BEP が InS であることは，スケジューラを実装可能にするための設計規約と位置付けられる．

図 10 に，スケジュールから実行までのパイプライン図を示す．InS BEP では，スケジューラは各命令が発行レイテンシ後に実行されると想定できる．ここでは，**発行レイテンシ**は発行から実行開始までのレイテンシを指し（図では，issue-to-exec latency）[*1]，InS BEP では固定となる．

スケジューラは，この想定の下でスケジューリングを行う；すなわち，生産者を発行してからその実行レイテンシの後に消費者を発行すればよい．発行時の生産者・消費者間の相対位置は，固定の発行レイテンシだけ平行移動して，生産者・消費者間の実行結果の授受として再現される．

Out-of-order のスケジューラはコア内で最も複雑なロジックの 1 つである [11] ため，これ以上のことまで織り込んでスケジューリングすることは難しい [12]．図 10 を見て，発行レイテンシが可変であった場合のスケジューリングを想像することは難しいのではないか．

この規約が動かし難いというのは正しい．しかし逆に，この規約をインタフェースとして守ってさえいれば，BEP が命令の相対位置を独自に変更することは可能だった訳である．

歴史 1 — ロード命令のレイテンシ予測

InS BEP が定石であるとの認識の歴史は前世紀にまで遡る．最初は，L1D ヒット/ミス予測をはじめとするロード命令のレイテンシ予測の研究であったと思われる [13]．

たとえば L1D ヒット/ミス予測では，ロード命令が L1D にヒットするかミスするかを予測し，投機的にスケジューリングするものである．ロード命令は，ヒットすると予測されれば通常通りに；ミスすると予測されれば，ミス処理が終わった直後に L1D アクセス・ステージにたどり着くよう，ウェイクアップされる．このような L1D ヒット/ミス予測は，現在では高性能なコアに必須の技術となっている．

この技術も，投機的スケジューリングのミス時にはパイプラインをストール/フラッシュする以外にないとの前提に立っている．一旦スケジューリング・ミスが起こればペナルティが避けられないので，その発生確率を減らすべきと考えるのである．

[*1] スケジューリング・ミスのペナルティを考える時には，実行終了までとする方が都合がよい．

歴史 2 — ミスを仮定したパイプライン

2010 年代には，RF を対象として，ミスを仮定したパイプラインを用いる研究がある [9], [10], [14]．

通常，パイプラインはヒットを仮定するものである．たとえば通常のパイプラインには，L1D アクセスを行うステージはあるが，L1D ミス時の L2 アクセスはステージとしては存在しない．L2 アクセスは，パイプラインの流れの外で行われる．

それに対して，ミスを仮定したパイプラインとは，ミス時の処理をステージとしてパイプライン内に組み込むものである．たとえば，Non-latency Oriented Register Cache System は，レジスタ・キャッシュ・ミス時のメインの RF へのアクセスをステージとしてパイプラインに組み込む．その結果，レジスタ・キャッシュ・ミス時には，メインの RF へのアクセスがパイプラインの流れの中で行われるので，ストール/フラッシュは必要ない．すなわち，ミスを仮定したパイプラインとは，そのミスをスケジューリング・ミスとはしないパイプラインの方式と言える．

これらは，InS BEP を深く理解したうえでのもので，InS を極めたとも言えるが，InS に囚われていたとも言える．これらの論文では，スケジューリング・ミス時に当該命令を選択的に遅らせること——すなわち OoS 制御 —— は，不可能とし，その理由を以下のように説明している；当該命令を遅らせるには，当該レーン内の後続の命令とそれらに依存する命令と，それらの後続の命令とそれらに依存する命令と…を再帰的に検出し，遅らせる必要が生じる．そのためには，高コストなスケジューラを BEP 内にもう 1 つ用意することになる．

InS の前提からこのように考えに至るのは無理からぬことである．振り返ってみれば，この説明は，半分は正しく，半分は間違っていた．スケジューラを BEP 内にもう 1 つ用意することになる（二次スケジューラ）との認識は正しかったが，それが必ず高コストになるとの認識は間違っていた訳である．なおこれらの論文中に，「BEP 内のスケジューラ」について，具体的な説明はない．

7. 性能評価

HPCG を対象とした提案コアの性能見積もりを，机上計算により行った．より正確な評価が可能な cycle-accurate なシミュレータを現在構築中である．

7.1 評価モデル

評価対象のモデルは，以下の 4 つである：

a64fx A64FX 相当．ベースラインとする．

a64fx-CG a64fx から combined gather 機能を除いたもの．Combined gather については後述する．

Vol.2021-ARC-244 No.36
Vol.2021-SLDM-194 No.36
Vol.2021-EMB-56 No.36
2021/3/26

a64fx+MBL1D　a64fx の L1D を多バンク化したもの．バンク衝突時のパイプラインの動作は，フラッシュではなくストールとした．

a64fx+MBL1D+OoS　a64fx+MBL1D の BEP を OoS 化したもの．

これらのモデルに対して，SIMD 幅を 512 b，1024 b，2048 b とした場合の性能を見積もる．MBL1D を持つモデルでは，そのバンク数も SIMD 幅に比例して増加させる．

Combined Gather

A64FX は，アラインされた 128 B のブロック内から最大 2 要素を一度にギャザーできる **Combined Gather** という機能を持つ [1]．

HPCG の内積計算における密ベクトル x に対するギャザーは，3 次元ステンシルに起因して，連続 3 要素ごとに飛び飛びのアクセスとなる．Combined gather では，この連続 3 要素のうちの 2 つは同時にアクセスできることになる．その結果，内積 1 つあたりのアクセス数は 27 から 18 へと削減される．

7.2　水平型/垂直型の演算の効率

まず，内積計算の浮動小数点演算に要するサイクル数を見積もる．A64FX (512-bit SIMD × 2-issue) をモデルとし，HPCG の疎行列密ベクトル積 (SPMV) の，ベクトル長 27 の内積計算の所要サイクル数を机上で見積もった．

なお，この見積もりは，論理・物理レジスタの本数やスケジューラのウィンドウ・サイズなどの制約は無視し，完全な最適化か，あるいは，完全な out-of-order 実行のいずれかが可能であった場合のものである．

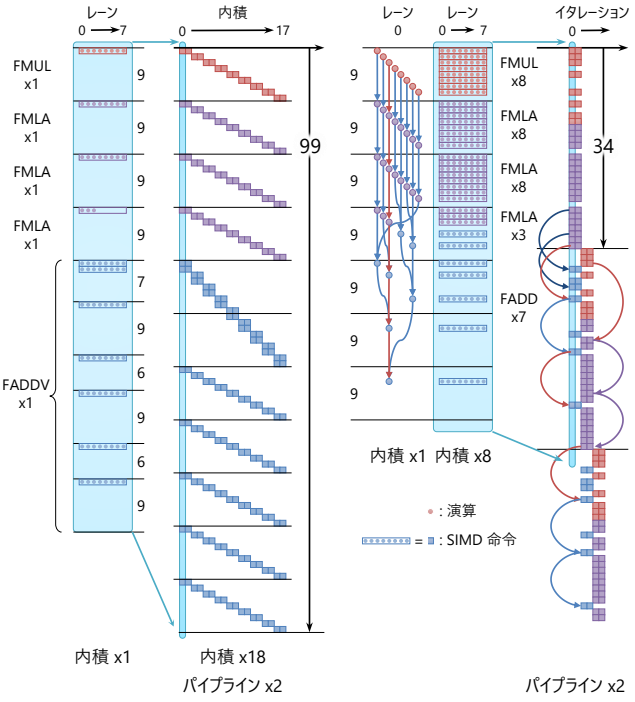


図 11　水平型（左）/垂直型（右）による内積計算

水平型

3.1 節で述べたように，**図 11**（左）の水平型では，SIMD 乗算/積和命令 (FMUL/FMLA) を用いて 1 つの SIMD レジスタに 8 要素ごとの積和の結果を累積し，最後に垂直加算命令 (FADDV) で総和をとる．FMUL，FMLA の演算レイテンシは，9 サイクルである．FADDV は，7 つの μOP に分解され，全体のレイテンシは 46 サイクルである．

A64FX の SIMD パイプライン 2 本を用いて，2 つの内積計算を並列に実行する．更に，演算レイテンシの隠蔽のため，9 つの内積計算をインターリーブ実行する．

図 11（左）に示されるように，$9 \times 2 = 18$ の内積に 99 サイクルを要する．内積 1 つあたりの理想の演算数 $27 \times 2 = 54$ FLOP と，A64FX のピーク性能 32 FLOP/cycle から，効率は $54 \times 18/99/32 = 30.7\%$ となる．

2.2 節で述べたように，まず，3 つ目の FMLA では剰余の 3 要素しか計算しないため，効率が低下する．また，FADDV に，FMUL/FMLA 以上の時間がかかっており，効率が大きく低下している．レイテンシの 46 もさることながら，7 つの μOP に分解して実行するため，発行幅の消費も大きい．計算時間の後半では，パイプラインは FADDV の μOP によって占められている．

垂直型

3.1 節で述べたように，図 11（右）の垂直型では，1 つの内積を 1 つのレーン（グループ）に割り当てる．SIMD パイプラインあたり 8 レーン（グループ）あるので，SIMD パイプライン 2 本で 16 の内積が並列に計算される．演算レイテンシの隠蔽のため，この図では，8 つの内積をインターリーブ実行し，最後に，7 個の（垂直型）FADD 命令からなるツリーによって部分和の総和を求めている（9 つの内積をインターリーブしても，ツリーがバランスせず，効率は向上しなかったため）．

図から，$8 \times 2 = 16$ の内積に 34 サイクルを要することが分かる．効率は $54 \times 16/34/32 = 79.4\%$ となる．効率が 100% にならないのは，総和のための FADD 命令 7 個によるもので，この 79.4% を現実的な上限と考えてもよい．水平型 (30.7%) と比べると，効率は約 2.6 倍になる．

7.3　演算とロードの効率

次に，ロードに要するサイクル数も織り込んだ結果について述べる．**図 12** に，SIMD 幅を 1024 b とした時の内積 1 つにかかるサイクル数を示す．

積み上げ棒グラフの要素，`a.v` と `a.i`，`x` は，疎行列 A の値とインデクスの連続ロード，密ベクトル x のギャザーにかかるサイクル数をそれぞれ表す．stall は，バンク衝突によるストールのサイクル数で，今回は式 (1) などの確率モデルによる推定である．

棒グラフに重ねて示した横棒は，前節で求めた水平型/

Vol.2021-ARC-244 No.36
Vol.2021-SLDM-194 No.36
Vol.2021-EMB-56 No.36
2021/3/26

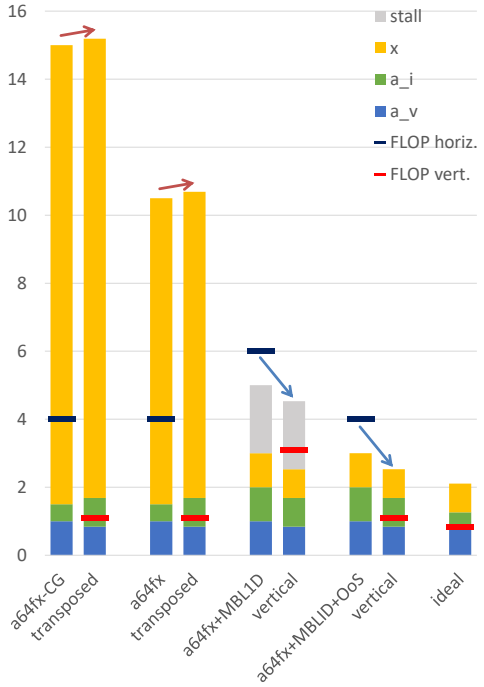


図 12 内積にかかるサイクル数

垂直型の浮動小数点演算にかかるサイクル数である．

同図には，（右端の ideal を除き）2 本の棒のペアが 5 つある．それぞれのペアは各モデルに対応し，ペア内の 2 本の棒は，水平型（左）/垂直型（右）に対応する．ただし，MBLID を持たない a64fx と a64fx-CG は，図中 transposed と示しているように，事前に転地を行い，`a.v` と `a.i` に対するロードが連続になるようにした．ideal は，ピーク性能を達成した場合のものである．

グラフからは，以下のことが読み取れる：

MBL1D を持たないモデル a64fx-CG，a64fx

- ギャザーに要するサイクル数 x が極めて多い．
- その結果，ロード・バウンドとなっており，演算効率を向上させる垂直型は効果がない．
  逆に垂直型では，32 b の `a.i` のロードに 64 b の `a.v` と同じサイクル数を要するため，サイクル数はかえって微増する（図中，赤矢印）．

MBL1D を持つモデル a64fx+MBL1D，+MBL1D+OoS

- MBL1D によってギャザーに要するサイクル数が激減した結果，水平型は演算バウンドになるが；
- 垂直型によって演算に要するサイクル数が激減した結果，垂直型は再びロード・バウンドとなる（青矢印）．
- a64fx+MBL1D+OoS は ideal にかなり近い．
  結果，32 b の `a.i` のロードに 64 b と同じサイクル数を要することが相対的に目立つようになっている．

OoS だけの効果は，垂直型における a64fx+MBL1D と a64fx+MBL1D+OoS の差であり，ギャザーにおけるバンク衝突によるストール・サイクル数 (stall) の削減になる．今回この値は推定値であるので，今後シミュレーションによる詳細な評価が必要である．

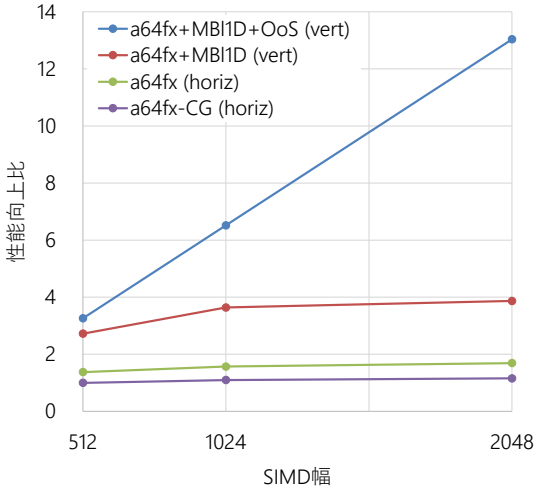


図 13 SIMD 幅に対する性能向上比

7.4 SIMD 幅に対するスケーラビリティ

図 13 に，前節で比較した各モデルの SIMD 幅に対する性能向上比を示す．a64fx (trans) の SIMD 幅 512 b の場合を 1 として正規化してある．水平型/垂直型は，それぞれのモデルでより性能の高い方（MBL1D なしは水平型，ありは垂直型）とした．

グラフからは，以下のことが読み取れる：

- MBL1D を持たない a64fx-CG，a64fx はビット幅 512b からほとんど性能向上しない．これは，ギャザー性能がビット幅によらず一定のためである．
- a64fx+MBL1D は，1024 b まではわずかに性能向上しているものの，それ以降は向上しない．これは，MBL1D のバンク数を SIMB 幅に比例的に増加させたとしても，バンク衝突の状況は悪化するからである．式 (1) より $P(32, 16) = 99.0\%$ であるので，これ以上確率が増大することはない．しかし，三重以上の衝突の確率が増加し，ストール・サイクル数は増加する．
- a64fx+MBL1D+OoS では，2048 b まで性能向上している．SIMD ビット幅が 1024 b で 4.7 倍，2048 b で 9.5 倍の性能向上となった．

7.5 回路面積・エネルギーとレイテンシに関する考察

最後に，回路面積・エネルギーとレイテンシについて，A64FX を基準として簡単に考察する．

MBL1D

MBL1D は，スイッチ部が追加的に必要になるが，A64FX 比では回路面積は増加しない可能性がある．A64FX の L1D は，combined gather の他，アラインされていない連続する 64 b × 8 要素 = 512 b を 1 サイクルでロードする能力を持つ．この 2 つを実現するためのスイッチが，提案の

MBL1D のそれとちょうど同じ規模になると推定される．

その場合，レイテンシもほとんど変化しない．

MBRF

4.3 節で述べたように，山田らは，スカラ・ユニットの整数・浮動小数点 RF に多バンク化を施している．詳細な評価の結果，RF の回路面積/消費エネルギーをフルポートの 22.6%/17.0% にまで削減できるとしている．RF の面積・エネルギーは非常に大きく，面積・エネルギーの削減の効果は大きい．RF 周辺はチップ内でも温度の高いホット・スポットであり，面積・エネルギーの削減はチップの電源電圧・動作周波数の向上に寄与する．

レイテンシは，従来の RF と同等程度以下になると考えられる．ポート数が大幅に減るので，各バンクのレイテンシは激減する．その減少分は，スイッチによる増加分を上回ると推定される [9], [10],

MBL1D と MBRF

L1D は，整数/浮動小数点数，それぞれの RF と同程度の面積である [9], [10]．仮に，L1D と整数/浮動小数点数，それぞれの RF の面積を 1，SIMD RF の面積を v，RF の多バンク化によってそれぞれの面積が 1/4 になるとすると，合計の面積の比は $(1+2/4+v/4)/(1+2+v)$ となる（図 3）．この値は，$v=8$ の時 $7/22 \simeq 31.8\%$，$v=16$ の時 $11/38 = 28.9\%$ となり，合計の面積は 1/3 程度に減少すると考えられる．

FIFO バッファ

OoS のためのバッファは少エントリかつ FIFO なので，通常のパイプライン・レジスタとレイテンシ等はほとんど変わらない．特にシフト・レジスタを用いて構成した場合，入力の容量の増加により setup time は若干増加するが，clock-to-data delay は増加しない．

MBL1D，MBRF と Out-of-Step

以上をまとめると，MBL1D，MBRF と OoS により，性能向上と回路面積・エネルギー削減の両方が実現できると予想される．

8. 関連研究

垂直計算

垂直計算については多くの関連研究がある．

SIMD 命令を基本としないメニー・コア系では，垂直計算が普通である．4.4 節で述べたように，メニー・コア系との大きな違いは面積効率である．

CUDA GPU では，メニー・コア系と同様，スレッドの概念から垂直計算が想起される．しかし実際には，連続アドレスでないと coalescing が働かず，メモリ性能が劣悪となるため，最適化を進めると水平計算となることが多い．

多バンク・レジスタ・ファイル

6 章で述べたように，山田らは，MBRF アクセスに複数のステージを割り当てることによりパイプライン・ストール/フラッシュを避ける Skewed Multistaged Multibanked RF を提案している [9], [10].

やはり 6 章で述べたように，これは InS の範疇である．また，SIMD や MBL1D に関する言及はない．

9. おわりに

本論文では，HPCG をはじめとする疎行列の計算を効率よく処理するアーキテクチャと，そのポイントとなる Out-of-Step バックエンド・パイプラインについて述べた．

HPCG の低い効率の原因について調査を進めた結果，短いベクトル長とギャザーがアプリケーション側の問題であることが分かった．前者は水平計算のスケーラビリティ，後者は不連続ロード/ストアのスケーラビリティの問題として顕在化する．

そこで，垂直計算の使用と多バンク L1D を柱とするコア・アーキテクチャを提案した．このアーキテクチャのポイントである Out-of-Step バックエンド・パイプラインは，スケジューラによって与えられた命令の位置関係を BEP 内で変更する技術で，バンク衝突の発生時にパイプライン・ストール/フラッシュを回避することができる．Out-of-Step によって多バンク L1D が実現可能になると，不連続アクセスの性能がベクトル・ユニット相応に高められ，不連続アクセスを多用する垂直計算も可能になる．その結果，HPCG の短いベクトル長とギャザーの問題が解消される．

机上計算の結果，対 A64FX 比で，SIMD のビット幅を 1024 とした場合に 4.7 倍，2048 とした場合に 9.5 倍の性能向上が得られることが分かった．

今後は，詳細な評価を進める必要がある．現在，鬼斬 [15] をベースに，cycle-accurate なシミュレータを構築中である．また，より詳細な評価のため，RISC-V コア RSD [16] をベースに，FPGA をはじめとするハードウェア実装を計画している．

参考文献

[1] Yoshida, T.: Fujitsu High Performance CPU for the Post-K Computer, Hot Chips 30 (online), available from ⟨https://www.fujitsu.com/jp/Images/20180821hotchips30.pdf⟩ (accessed 2021).
[2] ARM Ltd.: *ARM Architecture Reference Manual Supplement – The Scalable Vector Extension (SVE), for ARMv8-A, issue A.i* (2021).
[3] msyksphinz（訳）: RISC-V “V” Extension, RISC-V International (online), available from ⟨https://msyksphinz-self.github.io/riscv-v-spec-japanese/⟩ (accessed 2021).
[4] Dongarra, J., A. Heroux, M. and Luszczek, P.: a New Metric for Ranking High Performance Computing Systems, Technical Report UT-EECS-15-736, Electrical Engineering and Computer Science Department (2015).
[5] SankeiBiz: 「世界 4 冠」連覇のスパコン富岳 コロナ克服へ性能の期待高まる，（オンライン），入手先

⟨https://www.sankeibiz.jp/business/news/201119/cpc2011190640001-n1.htm⟩ (参照 2021).
[6] TOP500.org: HPCG – June 2020, TOP500.org (online), available from ⟨https://www.top500.org/lists/hpcg/06/⟩ (accessed 2021).
[7] Intel Corp.: *Intel Architecture Instruction Set Extensions and Future Features Programming Reference* (2020).
[8] Fujitsu Ltd.: *A64FX Microarchitecture Manual* (2020).
[9] Yamada, J.: Register Files of Superscalar Processors for Area and Energy Efficiency, PhD Thesis, The University of Tokyo (2017).
[10] Yamada, J., Jimbo, U., Shioya, R., Goshima, M. and Sakai, S.: Skewed Multistaged Multibanked Register File for Area and Energy Efficiency, *IEICE Trans. Info. and Syst.*, Vol. E100.D, No. 4, pp. 822–837 (online), DOI: 10.1587/transinf.2016EDP7414 (2017).
[11] Goshima, M., Nishino, K., Nakashima, Y., Mori, S., Kitamura, T. and Tomita, S.: A High-Speed Dynamic Instruction Scheduling Scheme for Superscalar Processors, *Int'l Symp. on Microarchitecture (MICRO)*, pp. 225 – 236 (2001).
[12] Yamada, J., Jimbo, U., Shioya, R., Goshima, M. and Sakai, S.: Bank-Aware Instruction Scheduler for a Multibanked Register File, *Journal of Information Processing*, Vol. 26, pp. 696–705 (online), DOI: 10.2197/ipsjjip.26.696 (2018).
[13] Yoaz, A., Erez, M., Ronen, R. and Jourdan, S.: Speculation Techniques for Improving Load Related Instruction Scheduling, *Proc. Int'l Symp. on Computer Architecutre (ISCA)*, pp. 42 – 53 (online), DOI: 10.1109/ISCA.1999.765938 (1999).
[14] Shioya, R., Horio, K., Goshima, M. and Sakai, S.: Register Cache System Not for Latency Reduction Purpose, *Proc. Int'l Symp. on Microarchitecture (MICRO)*, pp. 301–312 (online), DOI: 10.1109/MICRO.2010.43 (2010).
[15] Shioya, R.: Onikiri, (online), available from ⟨https://github.com/onikiri/onikiri2⟩ (accessed 2021).
[16] Shioya, R.: RSD RISC-V Out-of-Order Superscalar Processor, (online), available from ⟨https://github.com/rsd-devel/rsd⟩ (accessed 2021).