Vol.2022-ARC-250 No.6
2022/10/11

オペランドを命令間距離で指定する方式の性質に関する調査

小泉 透[1,a)]　塩谷 亮太[1]　入江 英嗣[1]　坂井 修一[1]

概要：アウトオブオーダー実行を行うプロセッサでは，一般にリネームやスケジュールに関わる処理がボトルネックとなる．このボトルネックを解消する取り組みの一つとして，オペランドをレジスタ番号でなく命令間距離で表す命令形式が研究されている．この命令形式ではプロセッサの回路を大きく簡略できる利点がある一方，命令数が大きく増加してしまう課題がある．この増加は主に，1) 指定できる距離や参照できる回数に上限があることから中継のコピー命令が必要なこと，2) 分岐/合流があっても距離を静的に表す必要がある制約からつじつま合わせのコピー命令が必要なこと，の二点に起因する．しかし，この命令数の増加が未熟なコンパイラに起因するのか，それとも命令形式の制約に由来する本質的なものなのかは，これまで十分な分析が行われておらず明らかではない．本稿では，コンパイラを陽には構成せず，標準的な RISC プログラムの実行系列から，1) 寿命および参照回数の分布を再確認し，2) 原理的に不可避な増加命令数を明らかにする．調査の結果，寿命や参照回数が N であるレジスタの出現頻度はおおよそ $1/N^2$ に比例することが判明した．また，この命令形式を持つ既存のアーキテクチャ，Dualflow や STRAIGHT における原理的に不可避な実行命令数の増加は，それぞれ 1.5 倍程度・1.2 倍程度であることが判明した．

1. はじめに

汎用プロセッサは，命令レベル並列性をより積極的に抽出することで，そのシングルスレッド性能を向上させ続けてきた．命令レベル並列性を可能な限り取り出すべくプロセッサの規模は増大し続け，今日では最大同時実行命令数が 10 を超えるプロセッサも多く発売されている [1], [2], [3], [4], [5]．一方で，アウトオブオーダー実行に欠かせないレジスタリネームや命令スケジュールなどの機構は，そのハードウェアの複雑性から規模を大きくすることが困難となりつつある．

このリネームやスケジュールにかかわる問題を解決すると期待されているのが，オペランド指定に**命令間距離**を用いた命令形式である．この形式では，命令間のデータの受け渡し地点をレジスタ名で指定するのではなく，消費者が何命令後/生産者が何命令前，といった距離を用いた形式でデータの授受を直接指定する．この形式の命令セットを持つアーキテクチャとして，Dualflow アーキテクチャ [6] や STRAIGHT アーキテクチャ [7] が提案されている．これらのアーキテクチャでは，リネームやスケジュールに多ポートのメモリが不要となり，それによりプロセッサのスケーラビリティが大幅に向上する．

このような距離を用いた形式では，通常のレジスタを用いた形式と比べ，**距離の調整に余分な命令実行が必要**となる．これは，分岐点/合流点をまたいだ参照について，命令間距離を静的に定めることが一般に不可能だからである．例えば，if 文の前にある生産者から if 文の後にある消費者までの命令数は，then 部を実行したかにより——つまり動的に——変化してしまい，静的に定められない．この問題の簡単な解決方法に，分岐点や合流点付近に距離を調整するコピー命令を追加するというものがある．生産者からの距離で指定する Dualflow アーキテクチャでは分岐点の直後に，消費者からの距離で指定する STRAIGHT アーキテクチャでは合流点の直前に，それぞれ生存変数の個数だけコピー命令を追加すれば，動作するプログラムとなる [7], [8]．この方法では冗長なコピー命令が多く追加されてしまうため，極力冗長な命令の追加を避けるためのアルゴリズムも研究されている [9], [10] が，それでも実行命令数が平均で 1.5 倍以上になるなど，多量の命令追加が発生している [9], [11].

本稿では，距離を用いた形式における本質的な命令数増加がどれほどであるかについて議論を行う．具体的には，前述した命令増加のうち，どの部分がコンパイラの未熟に由来するものなのか，どの部分が命令形式の制約に由来する本質的なものなのか，を明らかにする．実際にコンパイラを作り増加命令数を計測する方法は，コンパイラアルゴリズムが成熟していない現状では増加命令数の上限値を与えることしかできない．これに対し本研究では，標準的な

[1] 東京大学 大学院情報理工学系研究科
[a)] koizumi@mtl.t.u-tokyo.ac.jp

RISC プログラムの実行系列を用いて理論的に算出することを試みた．この方法は増加命令数の下限値を与えることとなり，コンパイラ最適化の余地がどの程度残されているかが明らかとなる．

2. オペランドを命令間距離で指定する命令セット

オペランドを命令間距離で指定する方法は二通りある．生産者が消費者までの距離を「何命令先」のように指定する方法と，消費者が生産者までの距離を「何命令前」のように指定する方法である．前者はプログラムの実行順序と同じ方向に距離を指定している．そこで**順方向形式**と呼ぶことにする．一方，後者はプログラムの実行順序をさかのぼる方向に距離を指定している．そこで**逆方向形式**と呼ぶことにする．

以下では実例として，順方向形式の Dualflow アーキテクチャと逆方向形式の STRAIGHT アーキテクチャのコードを使って，その概要を説明する．

```
   imm 3    2L    # 3
   imm 2    1R    # 2
   sub      XX    # 1
```
(1a) Dualflowにおける単純な引き算.

```
   addi $zero,3   # 3
   addi $zero,2   # 2
   sub  [2],[1]   # 1
```
(2a) STRAIGHTにおける単純な引き算.

```
   imm  a   2L    # a
   imm  b   1R    # b
   sub      1L,2L # a-b
   bneg NEG
   mov      2L    # a-b
   b    END
NEG:
   rsub 0   1L    # b-a
END:
   mov      XX
```
(1b) Dualflowにおける絶対値の計算.

```
   addi $zero,a   # a
   addi $zero,b   # b
   sub  [2],[1]   # a-b
   blt  [1],$zero,NEG
   mov  [2]       # a-b
   j    END
NEG:
   sub  $zero,[2] # b-a
   nop
END:
   mov  [2]
```
(2b) STRAIGHTにおける絶対値の計算.

図 1 Dualflow アセンブリの例と STRAIGHT アセンブリの例.

2.1 Dualflow アーキテクチャ

Dualflow アーキテクチャは，生産者が消費者までの距離を指定する方式のアーキテクチャである．Dualflow アーキテクチャの命令は，デスティネーションオペランドとして**結果を何命令後の命令の第何オペランドとして使うか**を記述する．ソースオペランドは，そこまでの実行結果から自動的に決まるため，陽に指定されない．

図 1(1a) に，単純に引き算をする Dualflow アセンブリを示す．第一の命令 `imm 3 2L` は即値 3 を生成する命令であり，その結果が二つ後の命令の左オペランドとして使われることが記述されている．第二の命令 `imm 2 1R` は即値 2 を生成する命令であり，その結果が一つ後の命令の右オペランドとして使われることが記述されている．第三の命令 `sub XX` は受け取った値の差を計算する命令であり，1 を生成する．その結果は本アセンブリの中では使用されないため，デスティネーションオペランドは仮に `XX` とした．

図 1(1b) に，絶対値を計算する Dualflow アセンブリを示す．第一の命令 `imm a 2L` は即値 a を生成する命令であり，その結果が二つ後の命令の左オペランドとして使われることが記述されている．第二の命令 `imm b 1R` は即値 b を生成する命令であり，その結果が一つ後の命令の右オペランドとして使われることが記述されている．第三の命令 `sub 1L,2L` は，受け取った値の差を計算する命令であり，その結果が一つ後の命令の左オペランドと二つ後の命令の左オペランドとして使われることが記述されている．一つ後の命令は `bneg NEG` だが，二つ後の命令は `mov 2L` と `rsub 0 1L` の二通りの可能性がある．第四の命令 `bneg NEG` は，受け取った結果が負である場合に `NEG` に分岐する命令である．第五の命令 `mov 2L` は，受け取った値をそのまま二つ後の命令の左オペランドに渡す命令である．実質的な計算をしない命令であるが，実行パスによらず静的な距離で表す制約を満たすために追加されている．第六の命令 `b END` は，`END` に無条件分岐する命令である．第七の命令 `rsub 0 1L` は，即値 0 から左オペランドを引く命令であり，符号反転が行われる．ここまでで，第八の命令 `mov XX` の左オペランドに絶対値の計算結果が送られる．

2.2 STRAIGHT アーキテクチャ

STRAIGHT アーキテクチャは, 消費者が生産者までの距離を指定する方式のアーキテクチャである．STRAIGHT アーキテクチャの命令は，ソースオペランドとして**何命令前の命令の結果を使うか**を記述する．デスティネーションオペランドは，命令と一対一に対応するため，陽に指定されない．

図 1(2a) に，単純に引き算をする STRAIGHT アセンブリを示す．第一の命令 `addi $zero,3` はゼロレジスタと即値 3 の和を計算する命令であり，3 を生成する．第二の命令 `addi $zero,2` はゼロレジスタと即値 2 の和を計算する命令であり，2 を生成する．第三の命令 `sub [2],[1]` は，二つ前の命令の結果から一つ前の命令の結果を引く命令であり，1 を生成する．

図 1(2b) に，絶対値を計算する STRAIGHT アセンブリを示す．第一の命令 `addi $zero,a` は即値 a を，第二の命令 `addi $zero,b` は即値 b を生成する．第三の命令 `sub [2],[1]` は，二つ前の命令の結果から一つ前の命令の結果を引く命令である．第四の命令 `blt [1],$zero,NEG` は，一つ前の命令の結果をゼロレジスタと比較し，小さければ `NEG` に分岐する命令である．第五の命令 `mov [2]` は，二つ前の命令の結果をコピーする命令である．実質的な計算をしない命令であるが，実行パスによらず静的な距離で表す制約を満たすために追加されている．第六の命令 `j END` は，`END` に無条件分岐する命令である．第七の命令 `sub`

$zero,[2] は，ゼロレジスタから二つ前の命令の結果を引く命令であり，符号反転が行われる．第八の命令 nop は何もしない命令であるが，命令間距離を調整するために追加されている．ここまでで，第九の命令が参照する「二つ前の命令の結果」が絶対値の計算結果となっている．この結果を生成したのは mov [2] と sub $zero,[2] のどちらの可能性もある．

3. オペランドを命令間距離で指定する命令セットにおける命令数増加

オペランドを静的な命令間距離で指定する命令セットでは，**静的な**距離でオペランドを指定しないといけない．この制約は非常に厳しいものとなっており，その解決には命令の追加が必要な場合が多い．以下では，どのような場合に命令の追加が必要かを分類し説明する．

3.1 命令のオペランド指定部が有限であることに由来する命令の追加

(A) 最大参照距離に由来するコピー命令の追加

順方向形式と逆方向形式の両方で発生する命令追加である．命令のオペランド指定フィールド長が有限であるため，ある一定の距離（最大参照距離と呼ぶ）を超えた参照関係を直接記述することができない．この制約を解決するためには，参照を“中継”するためのコピー命令を追加すればよい（図 2 左）．なお，参照が長い場合，一度スピルアウトし必要になったときにスピルインする，という方法で追加命令数を削減できる．

(B) 最大参照回数に由来するコピー命令の追加

Dualflow などの順方向形式に特有の命令増加である．命令のオペランド指定フィールドの個数が有限であるため，その個数（最大参照回数と呼ぶ）を超えたオペランドの記述を直接行うことができない．この制約を解決するためには，多数の消費者に複製して配るコピー命令を追加すればよい（図 3 左）．

3.2 実行パスによらない距離でオペランドを指定しないといけない制約に由来する命令追加

(C) ループ内定数の保持に必要なコピー命令の追加

順方向形式と逆方向形式の両方で発生する命令追加である．ループ外の命令が生産した値をループ内の命令が消費する場合，ループの反復ごとに生産者—消費者間距離が変化してしまうため，オペランド間距離を静的に定めることができない．この制約を解決するためには，ループ内部に自身を参照するコピー命令を追加すればよい（図 2 右）．

(D) 合流直前で値を生成できないことによる NOP 命令の追加

STRAIGHT などの逆方向形式に特有の命令増加である．プログラムの実行パスの合流点の一命令前では，合流点以降で使う値を生成できない．なぜなら，1) 実行パスの合流点の一命令前は，少なくともいずれかの実行パスで分岐命令であること，2) 分岐命令は実行結果を生成しないこと，3) 全ての実行パスで有効な値を生成しないと，合流点以降で参照が正しく行えないこと，の三点から従うからである．したがって，合流点の一命令前が分岐命令でない（フォールスルーである）場合でも，実行結果を生成しない命令（例えば，Store 命令など）を配置せざるを得ず，それも不可能な場合は NOP 命令を追加するしかない（図 3 右）．

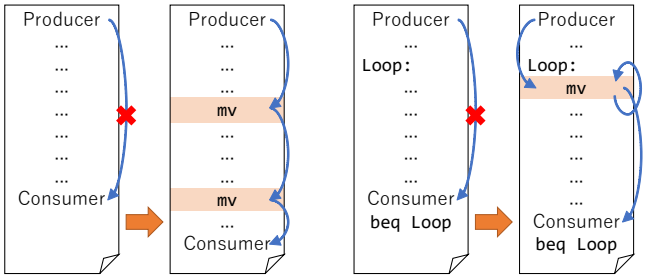

図 2 （左）最大参照距離を超える参照があった場合，それを“中継”するためのコピー命令を追加して解決できる．（右）ループ外の命令が生産した値をループ内の命令が消費する場合，ループ内に自身を参照するコピー命令を追加して解決できる．

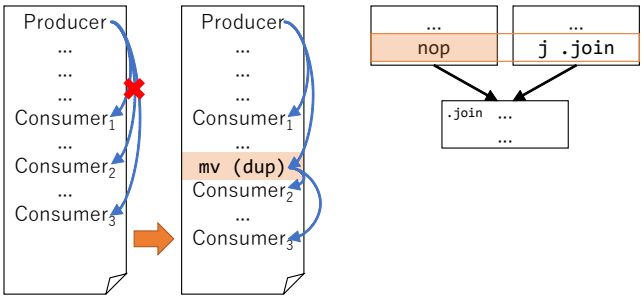

図 3 （左）Dualflow アーキテクチャでは，消費者の個数が多い場合，複製して分配するコピー命令の追加が必要となる．ここでは最大参照回数が 2 の場合を示した．（右）STRAIGHT アーキテクチャでは，合流点の直前でそれ以降使う値を生成できないため，フォールスルーであっても NOP 命令などの結果を生成しない命令を配置する必要がある．

(E) 実行順序制約が複数パスで一貫しない場合のコピー命令の追加

順方向形式と逆方向形式の両方で発生する命令追加である．Dualflow などの順方向形式では，ある値を分岐後に消費する命令が複数の実行パスにある場合，それらは分岐命令から等しい距離に配置されなければならない．この時，複数の消費者命令の間に実行順序の制約があり，それが複数パスで一貫しない場合，消費者命令の配置が不可能になる．一方 STRAIGHT などの逆方向形式では，ある消費者の特定のオペランドとして使われる値を生成する命令が複数ある場合，それらは合流点まで等しい距離に配置されなければならない．この時，複数の生産者命令の間に実行順序の制約があり，それが複数パスで一貫しない場合，生産

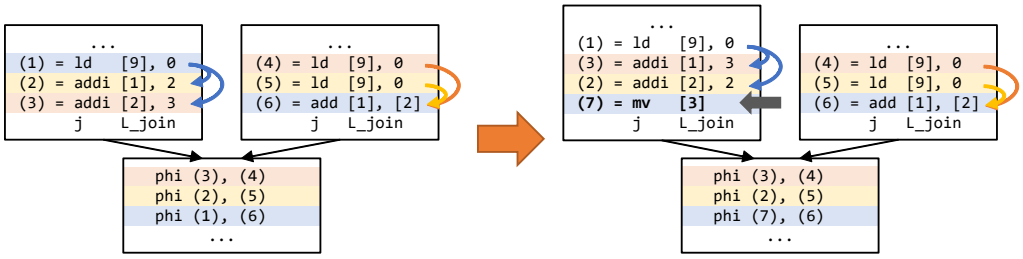

図 4 実行順序制約が複数パスで一貫しない場合，コピー命令の追加が必要となる．この図は逆方向形式の場合で，左の実行パスでは (1) の後に (2),(3) を，右の実行パスでは (4),(5) の後に (6) を実行する必要がある．合流点には (2)/(5) を取る phi 関数と (1)/(6) を取る phi 関数があるため，(1)/(6) と (2)/(5) の生成順序を揃えないといけないが，これは実行順序制約に反するため不可能である．コピー命令の追加により，この問題は解決される．

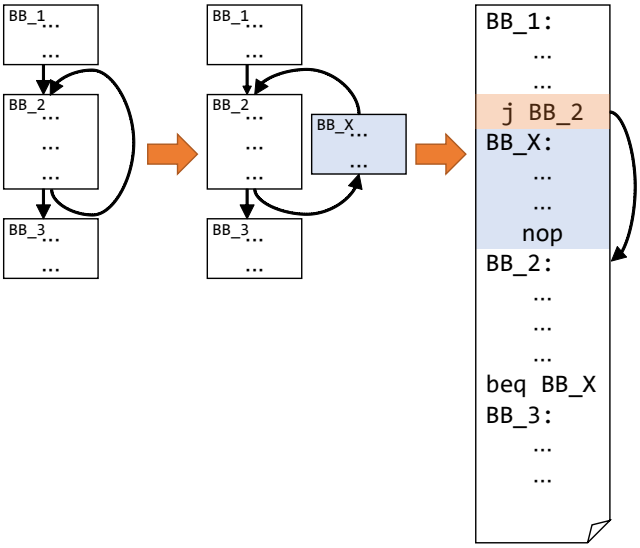

図 5 オペランドを命令間距離で指定する命令セットでは，危険辺を取り除くことで生じる基本ブロック（BB_X）が空でないことが多く，それを飛び越すために Jump 命令の追加が必要である．

者命令の配置が不可能になる．いずれの場合も，コピー命令を追加することで制約を緩めることができる（図 4）[10].

3.3 命令が増加することによるさらなる命令の追加

(F) 危険辺の除去に必要な Jump 命令の追加

順方向形式と逆方向形式の両方で発生する命令追加である．危険辺を除去することで生じる基本ブロックが空でない場合，飛び越えるために Jump 命令の追加が必要である（図 5）．標準的な RISC 命令セットの場合，レジスタ割り付けを適切に行うことで空にできることが多い．一方，オペランドを命令間距離で指定する命令セットでは，距離の調整のために何らかの命令を実行する必要が発生することが多く，当該基本ブロックを空にできないことが多い．

4. 調査

距離を用いた形式における本質的な命令数増加がどれほどかを調査した．調査に当たっては，標準的な RISC 向けにコンパイルしたベンチマークプログラムを実行したときのトレースを用いた．まず，命令数増加に大きな影響を与える寿命と参照回数の分布を測定した．次に 3 章で述べた各命令増加の原因について，プログラムを大きく変えない場合に不可避な命令数増加がどれほどかを測定した．最後に，スピルを追加した場合に命令数増加がどれだけ削減できるかを測定した．

4.1 調査方法

SPEC CPU 2006 および SPEC CPU 2017 をベンチマークとして用いた．SPEC CPU は，SPEC 社による実用プログラムでできたベンチマーク集である．本研究では SPEC CPU 2006 に含まれる 29 本と SPEC CPU 2017 に含まれる 20 本を全て用いた．ベンチマークの入力は，SPEC CPU 2006 の場合は `ref.0`，SPEC CPU 2017 の場合は `refspeed.0` とした．

コンパイルターゲットとなる標準 RISC 命令セットは RISC-V とし，コンパイルは gcc 8.1.0 を用いて行った．最適化オプションは`-O2` である．RISC-V は RISC 研究の最新の成果を反映したクリーンな命令セットであり，特定のハードウェア機構を想定した仕様を持たない．なお，レジスタ数は汎用レジスタが 31，浮動小数点数レジスタが 32 である．

各プログラムについてユーザーランドオンリーの実行系列を先頭から 10^{12}（1T）命令分取得し，解析を行った．ユーザーランドオンリーの実行系列を得るためには，鬼斬弐 [12] のスキップモードを使用した．解析は，実行系列をすべて取得してから行うのではなく，実行系列を取得しながらオンラインで行った．

まず，レジスタの寿命の分布とレジスタの参照回数の分布を調査した．レジスタの寿命とは，レジスタに値が書き込まれてから最後に参照されるまでの命令数のことである．レジスタの参照回数とは，レジスタに値が書き込まれてから最後に参照されるまでに参照された回数のことである．実行系列から，各寿命・参照回数についてその出現回

数をカウントした．

次に，3 章で述べた各命令増加の要因について，それぞれ追加数の下限を求めた．ここでは，RISC プログラムを“そのまま”Dualflow アーキテクチャや STRAIGHT アーキテクチャ向けに変換した場合を想定した．つまり，コピー命令や NOP 命令・Jump 命令の追加のみを行い，スピルの追加やプログラムの構造の変形は行わない場合の追加数を計測した．最大参照距離は 31，127，1023 の三種類の計測を行った．最大参照回数はすべて 2 回とした．その手順は以下のとおりである．

(A) 最大参照距離に由来するコピー命令

生産者命令と消費者命令の距離が k であるとき，D を最大参照距離として $\lfloor k/D \rfloor$ 個のコピー命令の追加が必要である．ただし，この方法ではループ内定数の保持に必要なコピー命令と重複計上になってしまうため，それが計上された際に距離を 0 にリセットする必要がある．

(B) 最大参照回数に由来するコピー命令の追加

同一のレジスタへの参照が $n(> R)$ 回行われた時，R を最大参照回数として $\lceil (n-R)/(R-1) \rceil$ 個の複製命令の追加が必要である．ただし，他の要因で必要なコピー命令と重複計上になってしまうため，それらが計上された際に参照回数を 0 にリセットする必要がある．

(C) ループ内定数の保持に必要なコピー命令

生産者命令がループの外であり，消費者命令がループの中であれば，ループ一周・生産者命令一つにつき一つのコピー命令の追加が必要である．ループであると判定するのは，実行系列中で同一のコールネストレベルで同じ PC が出現した場合とした．

(D) 合流直前で値を生成できないことによる NOP 命令の追加および (F) 危険辺の除去に必要な Jump 命令の追加

これらの命令の追加が常に発生したとする．すると，合流点の直前は必ず NOP 命令か Jump 命令となる．したがって，これらの命令の追加数は，RISC プログラムの実行系列において合流点の直前が Jump 命令でなかった回数に等しい．

ここで合流点とは，同一関数の複数命令の後続である命令のことを指す．関数先頭や間接関数呼び出しから復帰した地点も一種の合流点であるが，除外されるべきものである．

実際には，これらの命令の追加が常に発生するとは限らない．実行系列だけからこれらの命令追加が必要な条件を正確に求めることは困難であるが，先の値が上限を与える．

(E) 実行順序制約が複数パスで一貫しない場合のコピー命令の追加

この種のコピー命令の追加数を最小化するためには命令順序入れ替えが必須である．実行系列だけから命令順序入れ替えが可能かを判定することは困難であるから，本研究では取り扱わないものとする．

最後に，いくつかのスピル戦略に従ってスピルを行った場合に最大参照距離に由来する命令数増加がどれほどであるかを計測した．戦略としては，「動的なデスティネーションレジスタの寿命が k 以上だった場合はスピルする」というものと，動的なデスティネーションレジスタごとに最適にスピルを行う，というものを用意した．ここで，最適にスピルを行うとは，追加で実行されるコピー命令・ロード命令・ストア命令の合計数が最小化されることを言う．ただし，以下の三点から，順方向形式における値を計測することはできなかった．

- コピー命令で中継すべきか，スピルすべきかは，一般に寿命が切れるまで正確にはわからない．
- ロード・ストア命令を用いた場合，SP への参照が増え，複製のためのコピー命令を追加する必要がある．
- 複数のレジスタについて，SP への参照を介してこれらが相互作用する．

4.2 結果

4.2.1 レジスタの寿命の分布

レジスタの寿命の分布を図 6 に示す．寿命が N 以上のレジスタが定義される回数は，おおよそ $1/N$ に比例していた．したがって，寿命が N のレジスタが定義される回数は，おおよそ $1/N^2$ に比例すると計算される．

寿命が N 以上のレジスタが定義される回数がおおよそ $1/N$ に比例しているということは，以下の法則が成り立つことを意味している——N 命令生存したレジスタは，約 50%の確率でもう N 命令以上生存する．

また，寿命が 31・127・1023 を超えるレジスタの定義回数は，実行命令数に対する割合で，平均でそれぞれ 7.7%，1.8%，0.15%であった．

4.2.2 レジスタの参照回数の分布

レジスタの参照回数の分布を図 7 に示す．参照回数が N 以上のレジスタが定義される回数は，$N > 100$ の領域では，おおよそ $1/N$ に比例していた．したがって，参照回数が N のレジスタが定義される回数は，おおよそ $1/N^2$ に比例すると計算される．

参照回数が N 以上のレジスタが定義される回数がおおよそ $1/N$ に比例しているということは，以下の法則が成り立つことを意味している——N 回参照されたレジスタは，約 50%の確率でもう N 回以上参照される．

また，参照回数が 2 を超えるレジスタの定義回数は，実行命令数に対する割合で，平均で 9.7%であった．

4.2.3 命令数の増加

Dualflow などの順方向形式および STRAIGHT などの逆方向形式における，理論的な命令数増加を図 8 から図 10 に示す．

図 8 は最大参照距離が 31 の場合である．ループ内定数の保持に必要なコピー命令による命令数増加は，もとの

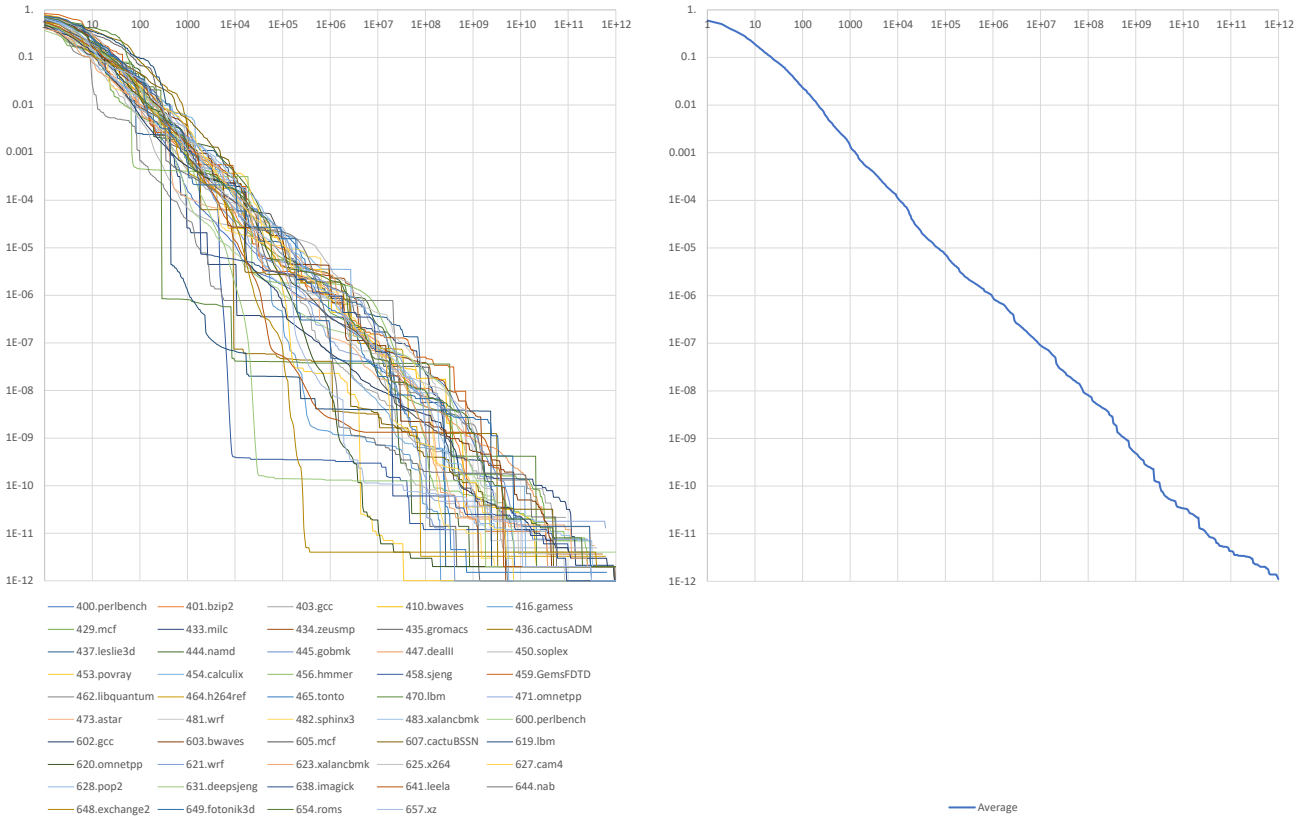

図 6 レジスタの寿命の分布．横軸は寿命，縦軸はその寿命以上のレジスタの定義頻度である．縦軸は実行命令数で正規化した．縦軸と横軸はともに対数目盛である．

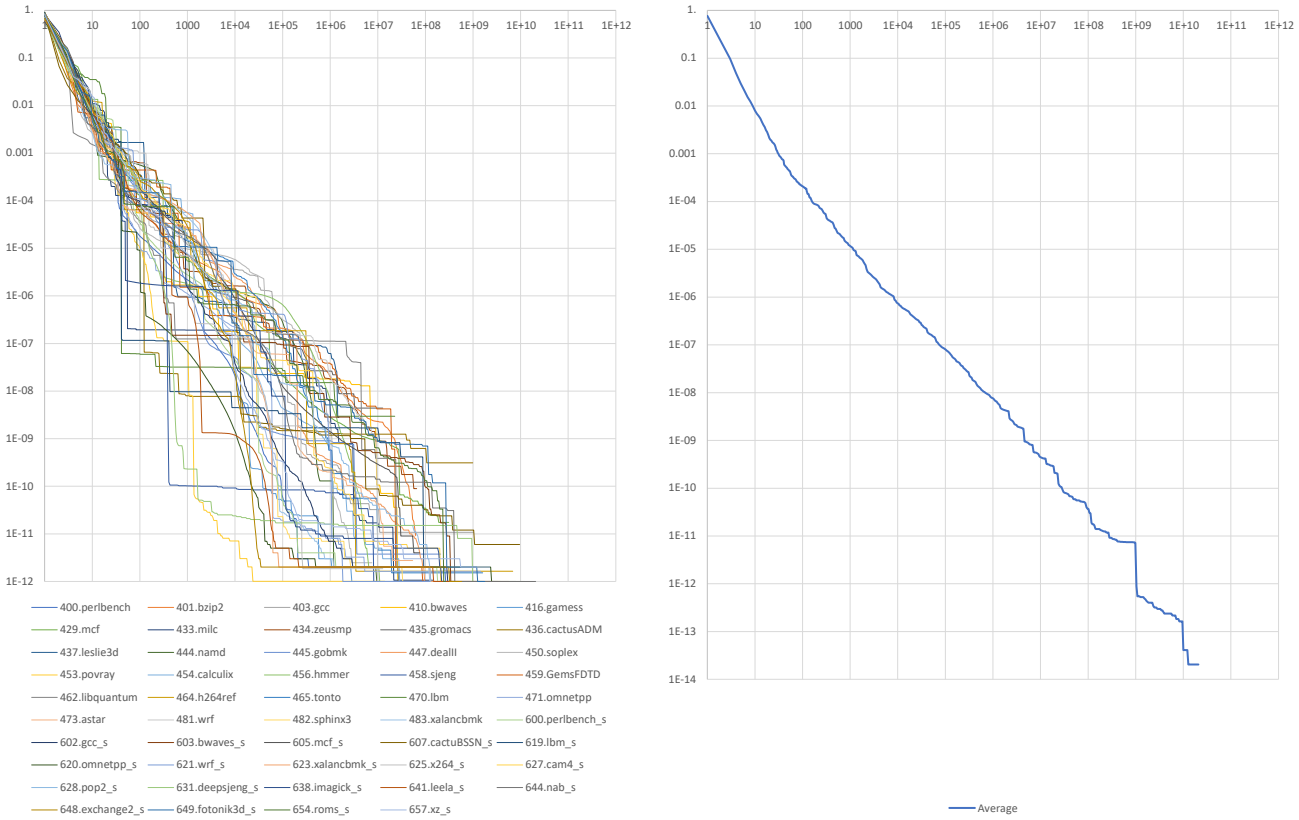

図 7 レジスタの参照回数の分布．横軸は参照回数，縦軸はその参照回数以上のレジスタの定義頻度である．縦軸は実行命令数で正規化した．縦軸と横軸はともに対数目盛である．

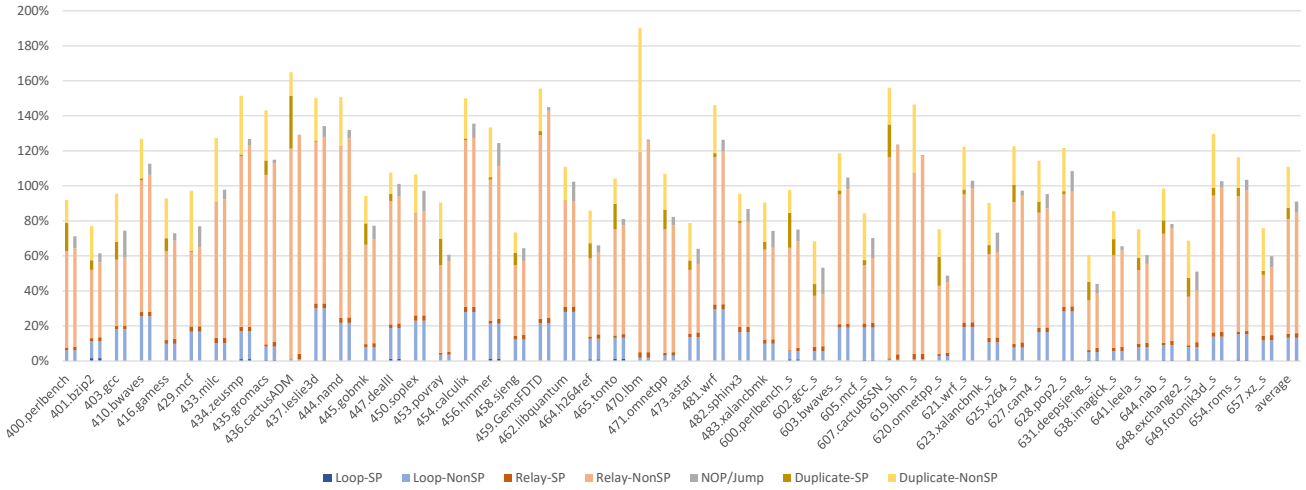

図 8 両形式における理論的な命令数増加．下から順に，ループ内定数の保持に必要なコピー命令（これは両形式で同数），最大参照距離に由来するコピー命令，最大参照回数に由来するコピー命令，`NOP` 命令および `Jump` 命令，の増加数を積み上げた．各コピー命令は，それが SP に対するコピーであるかで細分した．各ベンチマークについて二本の積み上げ棒があるが，左が順方向形式，右が逆方向形式についての結果である．RISC における実行命令数で正規化した．最大参照距離は 31．

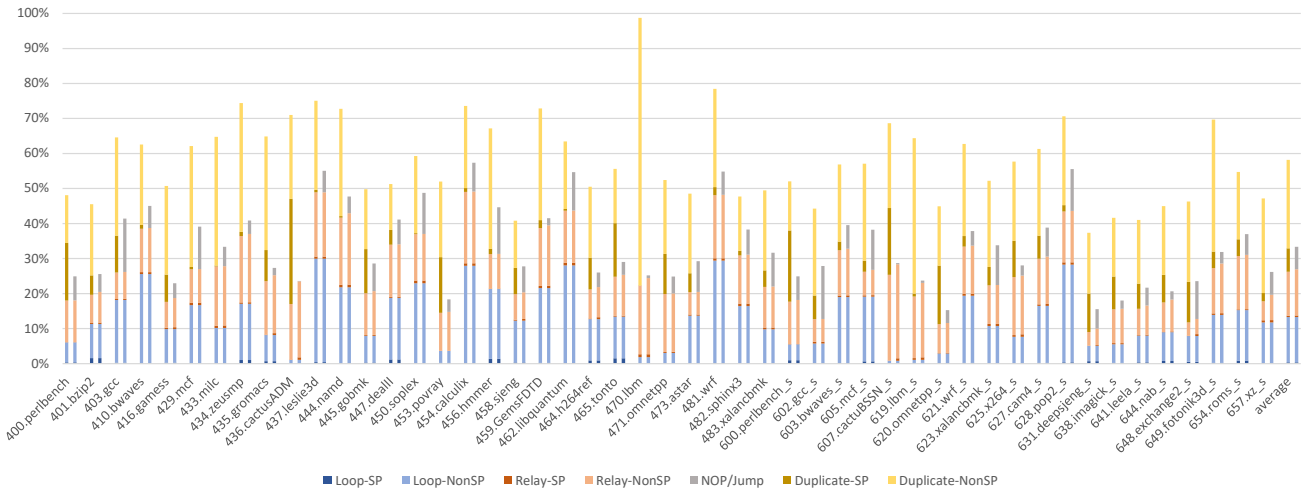

図 9 両形式における理論的な命令数増加．最大参照距離は 127．その他は図 8 と同様．

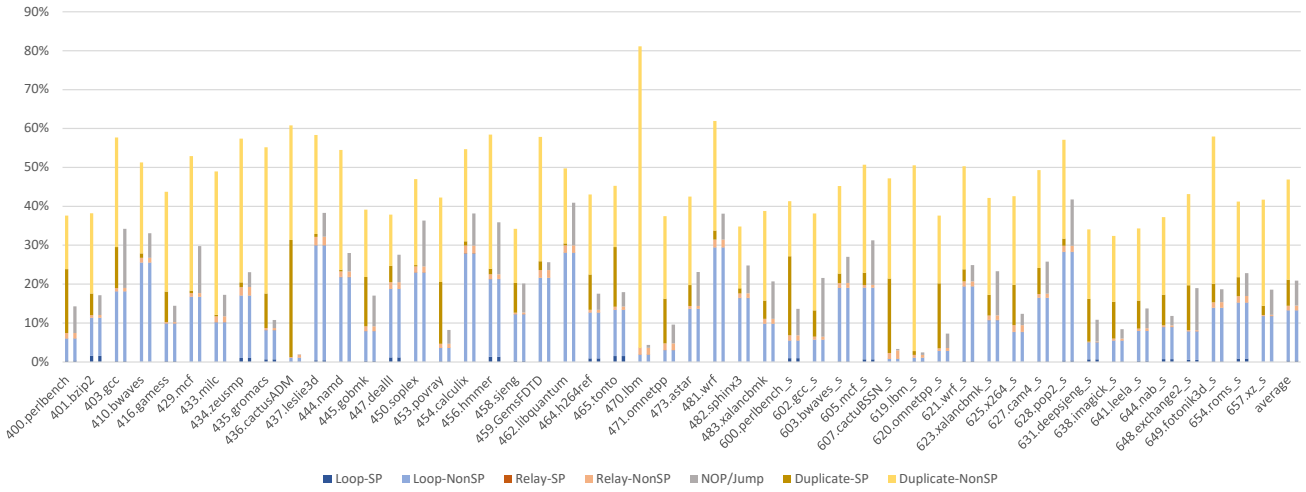

図 10 両形式における理論的な命令数増加．最大参照距離は 1023．その他は図 8 と同様．

RISC プログラムの命令数に対して平均で 13%であった．最大参照距離に由来するコピー命令による命令数増加は著しく，もとの RISC プログラムの命令数に対して平均で 68–71%であった．最大参照回数に由来するコピー命令の追加は，もとの RISC プログラムの命令数に対して平均で 30%であった．NOP 命令および Jump 命令の追加数の上限は，もとの RISC プログラムの命令数に対して平均で 6.4%であった．

SP に対するコピーは，ループ内定数の保持に必要なコピー命令と最大参照距離に由来するコピー命令では，それぞれ平均で全体の 2.7%，3.1–3.7%とその割合は小さかった．一方，最大参照回数に由来するコピー命令では，平均で全体の 22%と多くを占めていた．

次に，図 9 は最大参照距離が 127 の場合，図 10 は最大参照距離が 1023 の場合である．ループ内定数の保持に必要なコピー命令による命令数増加は，最大参照距離に左右されないため，先と同じく平均で 13%であった．最大参照距離に由来するコピー命令による命令数増加は，最大参照距離をのばすことで大きく削減され，それぞれ平均 12–13%，1.1–1.2%であった．最大参照回数に由来するコピー命令の追加は，どちらも平均 32%であった．わずかに増加したのは，最大参照距離に由来するコピー命令が減ったため複製が余分に必要になったことが原因である．NOP 命令及び Jump 命令の追加数の上限は，最大参照距離に左右されないため，先と同じく平均で 6.4%であった．

SP に対するコピーは，最大参照距離に左右されるか自明でないが，先ほどとほぼ同じく平均で全体の 2.7%（ループ内定数の保持に必要なコピー命令），2.9–3.6%（最大参照距離に由来するコピー命令），21%（最大参照回数に由来するコピー命令）であった．

4.2.4 スピルする戦略ごとの最大参照距離に由来する命令数増加

スピルする戦略ごとの最大参照距離に由来する命令数増加を図 11〜図 13 に示す．

図 11 は，最大参照距離が 31 の場合である．寿命が k 以上ならスピルするという戦略は，k が図に示した範囲内であればいずれも，両極端な戦略よりも少なくとも半分以下に命令数増加を抑えられた．これは，以下の二点が原因である．1) 多くの値は一時的な値で，スピルせずに受け渡せる．2) 長寿命なレジスタであっても，長期間参照され続けるということはまれであり，スピルすることで中継命令の追加を避けられる．

スピルすると判断する寿命の閾値は，最大参照距離に対して 2–4 倍程度の時が最も命令数増加を抑えられた．さらに，スピルするかを寿命で判断した場合でも，最適にスピルした場合と比べて 2.5%増と最適に近い命令数増加となった．

最適にスピルを行った場合，命令数増加は元の RISC プログラムの命令数に対して 13.6%に抑えられた．これは全くスピルしなかった場合の 71%に対して，約 1/5 の値である．

図 12 と図 13 は，最大参照距離がそれぞれ 127 と 1023 の場合である．一部の戦略は値が大きくなりすぎたため，棒を打ち切った上で合計値を表示してある．

スピルすると判断する寿命の閾値は，やはり最大参照距離に対して 2–4 倍程度の時が最も命令数増加を抑えられた．スピルするかを寿命で判断した場合でも，最適にスピルした場合と比べて 6%増，16%増と最適に近い命令数増加となった．

最適にスピルを行った場合，命令数増加は元の RISC プログラムの命令数に対してそれぞれ 2.6%，0.14%だった．これらは，全くスピルしなかった場合の 14%，1.2%に対して，それぞれ約 1/5，1/9 の値である．

5. 議論

5.1 最大参照距離に由来する命令増加

スピルを行わない場合，寿命が長い値の出現頻度は低いにもかかわらず，最大参照距離に由来するコピー命令が大量に追加される．4.2.1 節で示したように，寿命が 31 を超えるレジスタは全体の 10%程度しかない．既存研究でも同様の結果が得られている [6], [7]．しかしながら，最大参照距離に由来するコピー命令の増加は 68–71%と 10%よりはるかに多い．

最大参照距離に由来するコピー命令は，その寿命に比例した個数が必要である．例えば，最大参照距離が 31，寿命が 100 であれば，コピー命令は 3 つ必要である．したがって，最大参照距離に由来するコピー命令の数は，寿命が最大参照距離を超える頻度だけでなく，必要なコピー命令の数との積で考えなければならない．寿命 N のレジスタの出現頻度が $1/N$ に比例することを考慮に入れると，必要なコピー命令の数はプログラムの命令数を P として $\log P$ に比例した数となる．これが原因で，最大参照距離に由来するコピー命令の追加は，その出現頻度が低くても大量に必要となる．

4.2.4 節で示したように，適切なスピルを行うことによって，最大参照距離に由来する命令数増加を抑えることができる．したがって，オペランドを命令間距離で指定する命令セット向けのコンパイラは，RISC よりも積極的にスピルを行う必要がある．実際，既存のコンパイラは以下の枠組みで積極的にスピルしている．

- 全てのレジスタを呼び出し側保存とする [6], [7].
- ループ内で参照されない場合にスピルアウトする [6], [13].

5.2 最大参照回数に由来する命令増加

順方向形式の場合，参照回数が 2 より多いレジスタの定

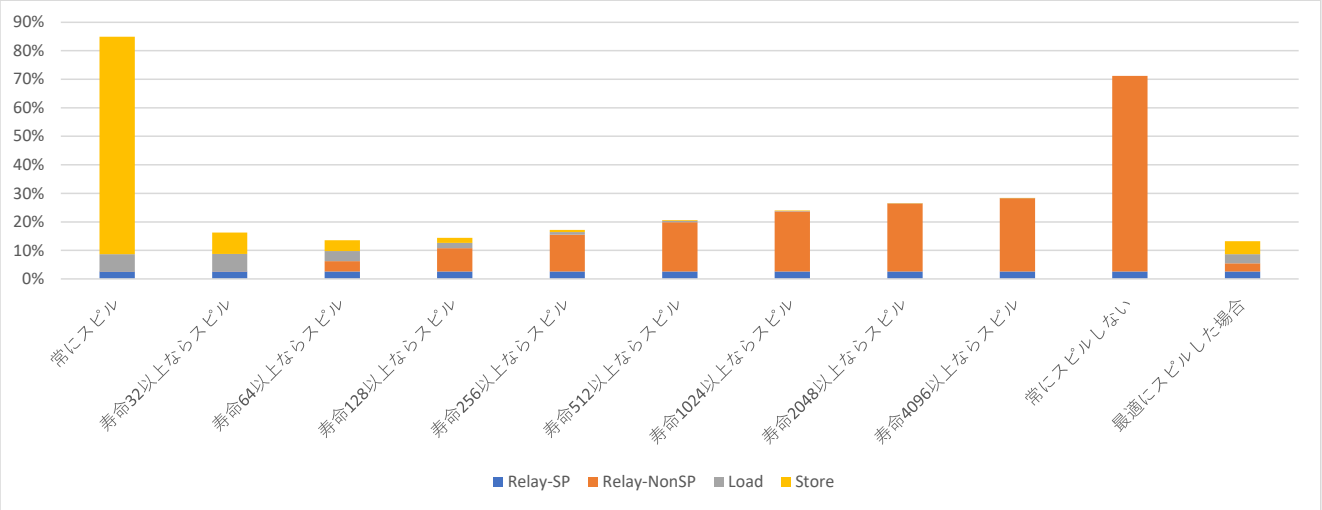

図 11 スピル戦略ごとの最大参照距離に由来する命令数増加．下から順に，SP の中継に必要なコピー命令（SP はスピルできない），SP 以外をスピルしなかった場合に中継に必要なコピー命令，スピルした場合のロード命令，スピルした場合のストア命令，の増加数を積み上げた．増加数は，RISC における実行命令数で正規化した．スピルしても常にロード命令を使うのではなく適宜コピー命令を使う，といった演算強度低減が考えられるが，ここでは行っていない．最大参照距離は 31．

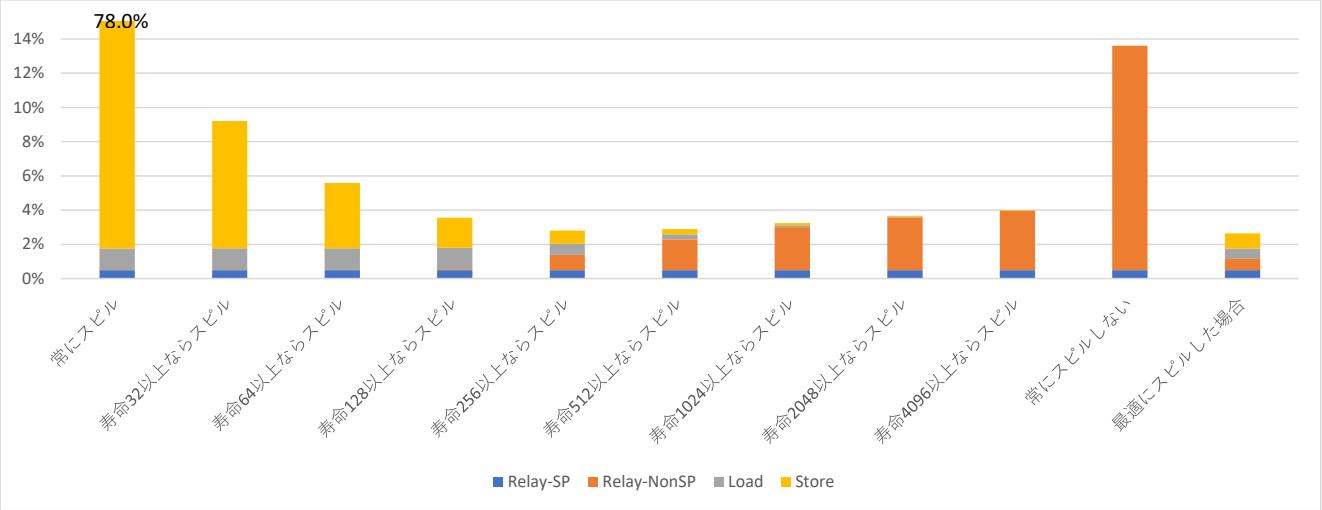

図 12 スピル戦略ごとの命令数増加．最大参照距離は 127．その他は図 11 と同様．

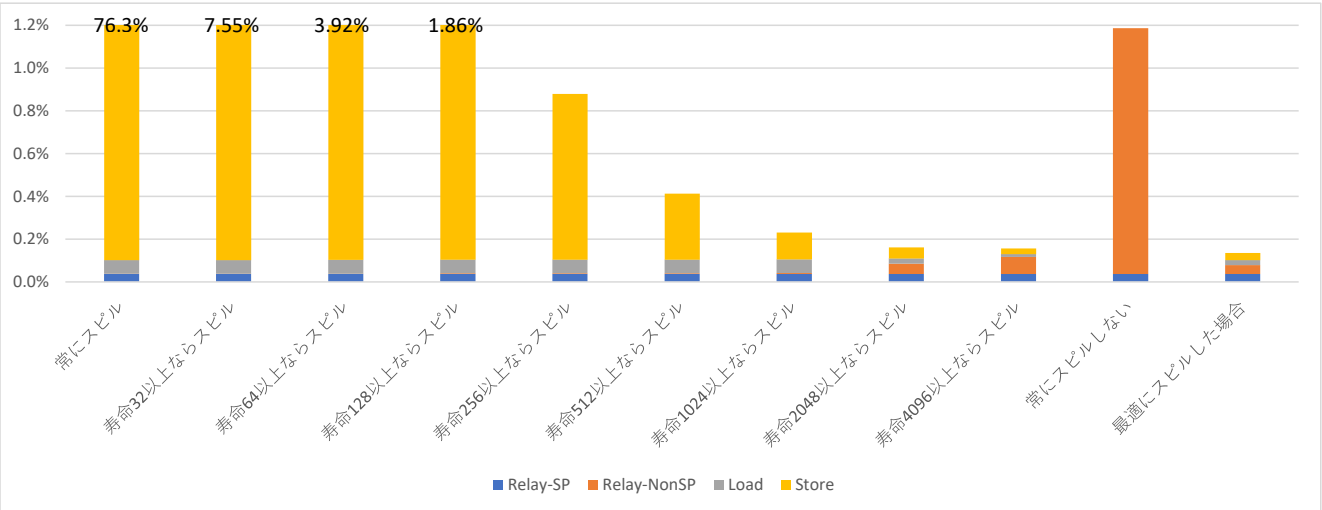

図 13 スピル戦略ごとの命令数増加．最大参照距離は 1023．その他は図 11 と同様．

義頻度は低いにもかかわらず，最大参照回数に由来するコピー命令が大量に追加される．4.2.2 節で示したように，参照回数が 2 を超えるレジスタは全体の 10%程度しかない．既存研究でも同様の結果が得られている [6]．しかしながら，最大参照回数に由来するコピー命令の増加は 30–32%と 10%よりはるかに多い．

最大参照回数に由来するコピー命令は，その参照回数に比例した個数が必要である．例えば，最大参照回数が 2，参照回数が 100 であれば，コピー命令は 98 個必要である．したがって，最大参照回数に由来するコピー命令の数は，参照回数が最大参照回数を超える頻度だけでなく，必要なコピー命令の数との積で考えなければならない．参照回数 N のレジスタの出現頻度が $1/N$ に比例することを考慮に入れると，必要なコピー命令の数はプログラムの命令数を P として $\log P$ に比例した数となる．これが原因で，最大参照回数に由来するコピー命令の追加は，その出現頻度が低くても大量に必要となる．

スピルを行っても最大参照回数に由来する命令増加は軽減することができない．複製命令を使うのではなくロード命令を使うことを考える．複製命令を一命令追加した場合，参照回数を $R-1$ 回増やすことができる．一方，ロード命令を一命令追加した場合，参照回数を R 回増やすことができる．これだけみると，ロード命令を使ったほうが複製の効率がよさそうに思われる．しかしながら，ロード命令を使うためには何らかのアドレスをレジスタから読み出す必要がある．多くの場合，そのための複製が追加で必要となるため，合計で必要な複製命令の数はほぼ同じとなる．もし SP が汎用レジスタではなく自由に読み出せる特殊レジスタに保存されている（STRAIGHT アーキテクチャはそのような設計になっている [7]）なら，SP 以外の値の複製に必要な命令の数を約 $(R-1)/R$ 倍に減らすことができる上，SP の複製に必要な命令をまったくなくすことができる．

5.3 不可避な命令数増加

上で調査した命令数増加は，あくまで RISC プログラムを“そのまま”Dualflow アーキテクチャや STRAIGHT アーキテクチャに変換した場合の値である．例えば，以下のようなコード変形を行えば，各種の命令追加を削減することができる．

- ループアンローリングを行うことで，ループ内定数の保持に必要なコピー命令の数を削減できる．
- 適切なスピルを行うことで，最大参照距離に由来するコピー命令を削減できる．
- 複製のためのコピー命令をロード命令にすることで，最大参照回数に由来するコピー命令を削減できる．

ここでは，以下の制約を課してコンパイルした場合に不可避な命令数増加を見積もる．

- ループアンローリングは RISC と同等の強度で行う．
- 適切にスピルを行い，命令数の増加を最小化する．
- 複製のためのコピー命令をロード命令に変換することは行わない．

4.2.3 節および 4.2.4 節の結果から，不可避な命令数増加を求める．Dualflow アーキテクチャでは最大参照距離 31 の場合に 57%程度の命令数増加が不可避である．STRAIGHT アーキテクチャでは最大参照距離 127 の場合に 22%程度の命令数増加が不可避である．その内訳としては以下のとおりである．

- ループ内定数の保持に必要なコピー命令，13%（両アーキテクチャで共通）
- 最大参照距離に由来する命令数増加，Dualflow アーキテクチャで最大参照距離が 31 の場合 14%，STRAIGHT アーキテクチャで最大参照距離が 127 の場合 3%
- 多数の消費者に複製して配るためのコピー命令，30%（Dualflow アーキテクチャのみ）
- 合流点付近での命令増加，高々 6%（STRAIGHT アーキテクチャのみ）

6. おわりに

オペランドを命令間距離で表す命令形式は，アウトオブオーダー実行のスケーラビリティを高めると期待されている．一方で，その特有の制約を満たすために命令の追加が必要なことが知られている．しかしながら，その追加量が多い原因がコンパイラ技術の未熟にあるのか，それとも本質的なものなのかは明らかではなかった．

本研究では，まず参照距離の分布を確認し，寿命が N 以上であるレジスタの出現頻度がおおよそ $1/N$ に比例していることを示した．これを用いて，RISC と同等のスピルしか行わなかった場合，最大参照距離に由来するコピー命令が多く追加されることを示した．また，参照回数が N 以上であるレジスタの出現頻度もおおよそ $1/N$ に比例していることを示した．これを用いて，順方向形式では最大参照回数に由来するコピー命令が多く追加されることを示した．

次に，RISC プログラムの実行系列から，理由別の命令増加数を求めた．また，スピルを最適に行った場合の命令数増加も求め，これらを使って Dualflow アーキテクチャ及び STRAIGHT アーキテクチャにおける，一定条件下で不可避な命令数増加を算出した．

命令数増加の下限を示したことで，コンパイラの生成するコードの品質を評価することができるようになった．今後はコンパイラの改良により，そのギャップを縮めることが目標となる．

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

参考文献

[1] Gwennap, L.: Cortex-A77 Improves IPC, *Microprocessor Report*, pp. 1–4 (2019).
[2] Suggs, D. and Bouvier, D.: The Path to "Zen 2", `https://www.slideshare.net/AMD/the-path-to-zen-2` (2019).
[3] Kanter, D.: Intel's Sunny Cove Sits on an Icy Lake, *Microprocessor Report*, pp. 1–4 (2019).
[4] Sadasivam, S. K., Thompto, B. W., Kalla, R. and Starke, W. J.: IBM POWER9 Processor Architecture, *IEEE Micro*, Vol. 37, No. 2, pp. 40–51 (2017).
[5] Jani, A.: Apple Ships Its First PC Processor, *Microprocessor Report*, pp. 1–5 (2021).
[6] 五島正裕，グェンハイハー，森眞一郎，富田眞治：DualFlow：制御駆動とデータ駆動を融合したプロセッサ・アーキテクチャ，情報処理学会研究報告，Vol. 1998-ARC-130, pp. 115–120 (1998).
[7] 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 51st Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 121–133 (2018).
[8] 五島正裕，グェンハイハー，縣　亮慶，森眞一郎，富田眞治：Dualflow アーキテクチャとそのコード生成手法，情報処理学会研究報告，Vol. 1998-ARC-130, pp. 115–120 (1999).
[9] 五島正裕，グェンハイハー，縣　亮慶，森眞一郎，富田眞治：Dualflow アーキテクチャの提案，並列処理シンポジウム JSPP2000，pp. 197–204 (2000).
[10] Koizumi, T., Sugita, S., Shioya, R., Kadomoto, J., Irie, H. and Sakai, S.: Compiling and Optimizing Real-world Programs for STRAIGHT ISA, *39th IEEE International Conference on Computer Design, ICCD 2021, Storrs, CT, USA, October 24-27, 2021*, pp. 400–408 (2021).
[11] 灘洋太郎，小泉　透，杉田　脩，塩谷亮太，門本淳一郎，入江英嗣，坂井修一：STRAIGHT アーキテクチャにおける C++コンパイラ開発と性能評価，情報処理学会研究報告，Vol. 2022-ARC-248, No. 3, pp. 1–7 (2022).
[12] Onikiri2, `https://github.com/onikiri/onikiri2`.
[13] Koizumi, T., Nakae, S., Fukuda, A., Irie, H. and Sakai, S.: Reduction of instruction increase overhead by STRAIGHT compiler, *2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*, pp. 92–98 (2018).