社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2007－SLDM－130（2）
2007／5／10

部分的なデータフォワーディング機構を持つプロセッサのための発見的命令スケジューリング手法

稗田 拓路† 田中 浩明† 坂主 圭史† 武内 良典† 今井 正治†

† 大阪大学 大学院情報科学研究科 集積システム設計学講座
E-mail: †{t-hieda, h-tanaka, sakanusi, takeuchi, imai}@ist.osaka-u.ac.jp

あらまし 部分フォワーディングは，プロセッサのパイプライン中にフォワーディングパスを部分的に持たせる手法である．部分フォワーディングに対応した命令スケジューリングを行うことで，部分フォワーディングに関する設計空間探索を行って命令の実行効率を落とさずに回路量を削減することができる．そのために整数計画法を用いた手法が提案されているが，大きなプログラムに対しては実用的な時間で終了しない．そこで本稿では，部分フォワーディングを有効に利用するための発見的な命令スケジューリング手法を提案する．実験では，ベンチマークプログラムに対して短いコンパイル時間で最適解に近い結果が得られたこと，また，部分フォワーディングに関する設計空間探索を行った場合に，整数計画法を用いた場合と同じパレート最適解候補が得られたことを示す．

キーワード 部分フォワーディング，命令スケジューリング，コンパイラ，設計空間探索

Heuristic Instruction Scheduling Method for Processors with Partial Forwarding Structure

Takuji HIEDA†, Hiroaki TANAKA†, Keishi SAKANUSHI†, Yoshinori TAKEUCHI†, and Masaharu IMAI†

† Integrated System Design Laboratory, Department of Information Systems Engineering, Graduate School of Information Science and Technology, Osaka University.
E-mail: †{t-hieda, h-tanaka, sakanusi, takeuchi, imai}@ist.osaka-u.ac.jp

Abstract Partial forwarding is a design method to put forwarding paths on a part of processor pipeline. To schedule instructions considering partial forwarding structure, the designer can reduce hardware cost of the processor without performance loss by design space exploration of the forwarding structure on a processor. Though a scheduling method with integer linear programming method for partial forwarding processor has already been proposed, compilation may not complete against large programs. In this paper, we propose a heuristic instruction scheduling method for processors with partial forwarding structure. Experimental results show that the proposed method can generate almost optimal scheduling results in short time and optimal solution candidates with the proposed method are the same as those with the integer linear programming method in design space exploration.

Key words Partial forwarding, Instruction scheduling, Compiler, Design space exploration

1. はじめに

近年，組込みプロセッサに対する高性能化への要求が高まるにつれ，VLIW やスーパースカラなどの，同時に複数の命令を実行できる組込みプロセッサが使用されている．また，動作周波数を向上させるために，パイプライン段数を大きくした組込みプロセッサが使用されている．しかしながら，組込みプロセッサに対しては高性能だけでなく，小面積や低消費電力も求められるため，組込みプロセッサの設計者は，性能向上を行いつつ面積や消費電力を削減するという相反する条件を満たしたプロセッサを設計することが求められ，組込みプロセッサは使用用途に特化したアーキテクチャとなることが多い．

プロセッサ設計者が面積削減・低消費電力化を行う手段の一つとして，データパスの回路量を減らす方法がある．代表的なものがクラスタ化 VLIW [1] である．クラスタ化 VLIW では演算器を複数のクラスタに割り当て，クラスタ内からのみアクセ

ス可能なローカルレジスタファイルを用意する．レジスタの読み書きや演算をできるだけクラスタ内部に限定させることで，全体のプロセッサ面積を小さくする．しかし，クラスタ間でデータのやり取りを行う場合には，専用のデータパスやグローバルレジスタが必要となり，クラスタ間フォワーディングを行うと，クラスタ数が多い場合，必要なコストが大きくなる．

クラスタ化 VLIW 以外の手法に，部分フォワーディングがある．部分フォワーディングは，パイプライン中の一部分のみフォワーディング回路を持たせる手法である．高い性能向上を達成できる部分にのみフォワーディング回路を持たせることができれば，全体的にフォワーディングを行ったプロセッサより少ないコストで同等の実行効率が見込めるが，部分フォワーディングを持つプロセッサでは，パイプライン中に存在する演算結果を利用できる命令位置が部分フォワーディングパスに依存するため，通常の命令スケジューリングでは十分に部分フォワーディングを利用することが難しい．そのため，部分フォワーディングに対応した命令スケジューリングを行う必要がある．また，フォワーディング回路を設けるパスの選択によって命令の実行効率が変わるため，有効なフォワーディングパスを見つける方法が重要となる．

本稿では，部分フォワーディングを有効に利用するための発見的命令スケジューリング手法を提案する．発見的手法を用いて部分フォワーディングに対応した命令スケジューリングを行うことで，短時間で部分フォワーディングを有効に利用した命令列を得ることができ，部分フォワーディングに関する設計空間探索を十分に行えるようになる．

フォワーディング回路を削減するという考え方は，クラスタ化 VLIW のクラスタ間フォワーディングを部分的に行う方法で使われた [2]．最初に部分フォワーディングという用語を提案した Ahuja ら [3] は，部分フォワーディングを用いることでプロセッサの設計品質を向上させられることを示したが，命令スケジューリングアルゴリズムについては考察にとどまっており，具体的なスケジューリング法は提案していない．部分フォワーディングに対する命令スケジューリングについては [4] [5] [6] などがある．[4] は部分フォワーディングによるハザードの検出に焦点を当てており，[5] は DSP やクラスタ化 VLIW などを主な対象としている．[6] は整数計画法を用いて命令スケジューリングを行う手法であるが，整数計画法を用いているため，命令スケジューリングに膨大な時間がかかるおそれがある．

本稿の構成は以下の通りである．まず 2 節で部分フォワーディングについて述べる．次に，3 節で命令スケジューリング手法について述べ，4 節で命令スケジューリングアルゴリズムの評価実験の結果と，設計空間探索の結果を述べる．最後に 5 節でまとめを述べる．

2. 部分フォワーディング

本節では，部分フォワーディングの特徴と，部分フォワーディングを行う際に発生する問題点について述べる．

2.1 部分フォワーディングの着想

一般的にパイプラインプロセッサでは，レジスタファイルから値を読み込むステージと，レジスタファイルに演算結果を書き込むステージは異なるため，命令がフェッチされてから演算結果がレジスタに書き込まれるまでに数サイクルかかる．そのため演算結果を別の命令で使用する場合，命令フェッチを遅らせてレジスタファイルから演算結果を読み出せるようになるまで数サイクル待つ必要があり，実行効率を大きく落とす要因となる．通常は，パイプライン中の演算結果を，演算ユニットの入力に直接送るためのフォワーディング回路を設けることで後続命令がレジスタに書き込まれる前の演算結果を使用できるようにする [7] [8]．

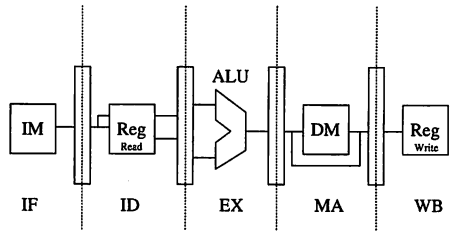

図 1　5 段パイプラインを持つ DLX プロセッサ

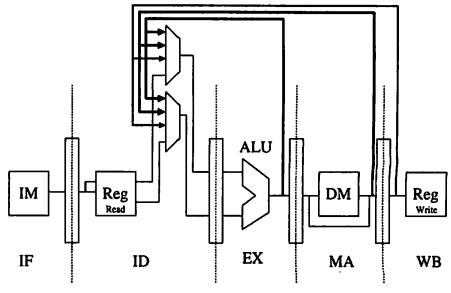

図 2　完全フォワーディング機構を持つ DLX プロセッサ

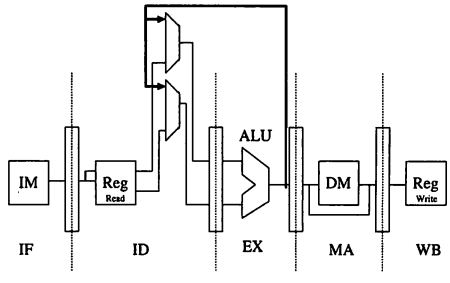

図 3　EX ステージからのみフォワーディング機構を持つ 5 段パイプライン DLX プロセッサ

例えば，図 1 に示す 5 段パイプライン DLX プロセッサ [7] は，2 ステージ目でレジスタの値を読み込み，5 ステージ目でレジスタに演算結果を書き込む．そのため，ある命令の演算結果を別の命令が利用するためには，4 サイクル後に実行する必要がある．対して，図 2 のようにフォワーディングを行った場合は，1 サイクル後に実行することができる．

フォワーディングを行うために必要なコストはプロセッサに

表 1　演算結果を利用するために必要な待ちサイクル数の比較

サイクル数	0	1	2	3	4	5 以上
図 1: フォワーディングなし	×	×	×	×	○	○
図 2: (完全) フォワーディング	×	○	○	○	○	○
図 3: 部分フォワーディング (の例)	×	○	×	○	○	○

よって異なるが，パイプライン段数に比例してフォワーディング回路が占めるチップ面積は大きくなる．さらに，VLIW など 1 サイクルに複数の命令を実行できるプロセッサの場合，任意の命令スロットで実行された命令の演算結果を利用できるようにフォワーディングを行うと，フォワーディング面積は演算器の数の 2 乗に比例する [3]．したがって，チップ面積や消費電力などの設計制約が厳しい場合，フォワーディングに必要なコストが無視できない場合がある．そこで，部分的にフォワーディングを行うことで，制約を満たしつつ命令の実行効率を上げる手法が提案されている [3]．これを部分フォワーディングという．図 3 は部分フォワーディングの例である．図 2 との違いは，MA ステージからのフォワーディングを省いていることである．また，図 2 のように全ステージからのフォワーディングを部分フォワーディングと区別する場合，完全フォワーディングと呼ぶ．

2.2　部分フォワーディングと命令スケジューリング

部分フォワーディングをプロセッサに適用する場合，前の命令の演算結果を利用できるタイミングが一般的なパイプラインプロセッサと異なる特性を持つため，部分フォワーディング構造を考慮した命令スケジューリングが必要となる．

表 1 は，図 1, 2, 3 の各プロセッサ上で演算命令を実行したとき，後続命令が演算結果を使用できるサイクルを，演算命令がフェッチされたサイクルを基準として示した表である．フォワーディングが存在しない図 1 のプロセッサ上では，前述したように演算結果を利用するためには 4 サイクル待つ必要がある．また，フォワーディングを持つ図 2 のプロセッサ上では 1 サイクル待てばよい．しかし，図 3 のような部分フォワーディング構造を持つプロセッサの場合，1 サイクル後では演算結果を使用できるが，4 ステージ目の MA ステージからのフォワーディング回路がないため，2 サイクル後では演算結果が使用できず，3 サイクル後からまた使用できるようになる．そのため，図 3 のプロセッサ上でプログラムを実行する場合，ある命令の演算結果を 2 サイクル後に別の命令で使用するような命令列が与えられると，後続命令は先行命令の演算結果を読み込めなくなり，プログラムは正しく動作しない．

図 3 の部分フォワーディングプロセッサに対して，従来のスケジューリング手法で命令スケジューリングを行う場合，演算結果が使用できるようになるまで 3 サイクルかかると仮定してスケジューリングを行い，先行命令から 3 サイクル以上後に後続命令が実行されるようにすればよいが，1 サイクル後に先行命令の演算結果を利用できる場合であっても，後続命令が 3 サイクル以上後で実行されるため部分フォワーディングの利点を十分利用しているとはいえない．したがって，部分フォワーディングを持つプロセッサの性能を十分に活用するためには部分フォワーディングに対応した命令スケジューリングを行う必要がある．

部分フォワーディングを行う場合，部分フォワーディング回路を設ける場所によって実行効率が変化するため，アプリケーションごとに設計空間探索が不可欠となる．設計空間探索を行う際には，様々な部分フォワーディング構造を持つプロセッサを対象としてプログラムをコンパイルする必要があるため，スケジューリングに要する時間が短いコンパイラが求められる．

3.　発見的スケジューリング手法

本節では，発見的手法による部分フォワーディングに対応した命令スケジューリングについて述べる．

3.1　有効命令距離

本手法では，基本ブロックごとに命令スケジューリングを行う．命令スケジューリングの入力は基本ブロック中の命令列のデータ依存グラフ [9] [10] である．データ依存グラフは命令列における命令の依存関係を表現し，点が命令を，有効辺が依存関係を，辺の重みは命令間の依存関係を満たすのに必要なサイクル数を表す．ただし，部分フォワーディングに対応した命令スケジューリングを行う場合，データ依存グラフに部分フォワーディング構造を反映するための表現が必要となる．本手法では，部分フォワーディング構造の情報をデータ依存グラフに持たせるために，有効命令距離を定義する．

後続命令が先行命令の演算結果を使用できる命令距離を有効命令距離と呼び，(D, L) で表す．ここで，命令距離とは命令間の実行サイクル数である．

L は演算結果を利用できることが保障される最小の命令距離を表す．先行命令と後続命令の命令距離が L 以上であれば，後続命令は先行命令の演算結果を利用できる．L は，レジスタに値が書き込まれるまでにかかるサイクル数，あるいは以後全てのステージで演算結果がフォワーディングされるサイクル数に対応する．

命令距離が L 未満の場合は，必ずしも演算結果を利用できるとは限らない．そのため，利用できる命令距離を個々に表現する必要がある．$D = \{d_1, d_2, d_3, \ldots\}$ は演算結果を利用できる命令距離の集合であり，D の要素の値は L 未満である．依存関係のある命令間の命令距離は D 中の値か，L 以上の値である必要がある．例えば，図 3 の DLX プロセッサの場合，ALU 演算命令の演算結果の有効命令距離は，$(\{1\}, 4)$ となる．また，完全フォワーディングプロセッサである図 2 の場合，有効命令距離は，$(\{\}, 1)$ となる．この場合，D は空集合である．

図 4 に，辺の重みとして有効命令距離を持たせたデータ依存グラフの例を示す．

3.2　命令スケジューリングアルゴリズム

本稿では，リストスケジューラをベースとして，部分フォワーディングを有効に利用できるような発見的手法を用いたアルゴリズムを提案する．提案手法の基本方針は以下の通りである．

- 部分フォワーディングを利用できる可能性を広げるため，後続命令が部分フォワーディングを利用できる可能性がある場合は，実行できる命令をあえて実行せずに別の命令を実行する

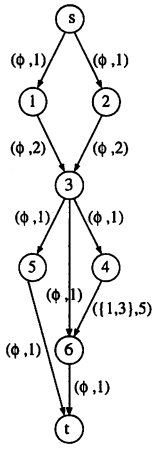

図 4　データ依存グラフの例

```
入力: データ依存グラフ DDG(V, E)
      V: 命令ノード集合 E: 依存関係を表す有効辺集合
出力: スケジューリングされた命令列 I = {i_1, i_2, ..., i_m}
begin
    cycle = 1; // サイクルを 1 に初期化
    while (DDG 中に未スケジュール命令ノードが存在する) {
        while (cycle で実行可能な未スケジュール命令ノードが存在する) {
            // 次にスケジュールする命令ノードを取得する
            v_k = GetScheduleNode(DDG, cycle);

            if (v_k ≠ NULL) {
                i_{v_k} = ノード v_k が示す命令;
                i_{v_k} を cycle サイクル目にスケジュールする;
                v_k をスケジュール済み命令ノードにする;
            }
            else {
                break; // while ループを抜ける
            }
        }
        cycle = cycle + 1; // サイクルを 1 進める
    }
    return I;
end
```

図 5　発見的手法による命令スケジューリングアルゴリズム：メイン

```
入力: データ依存グラフ DDG(V, E), 現在のサイクル cycle
出力: スケジュールする命令ノード v_sch
begin
    v_sch = NULL;
    while (v_sch = NULL) {
        READY = {v_j ∈ V|cycle で実行可能な命令ノード };
        v_k = ;
        for each (v ∈ DDG) {
            v_k = READY 中で最も最大パス長の長い命令ノード;
            v_succ = v_k の最長パス上にある後続命令;
            (D_k, L_k) = v_k と v_succ との有効命令距離;
            fastest = 0;
            c_min = inf;
            if (|READY| < L_k ) {
                for (i = 0..L) {
                    c_i = v_k を cycle + i サイクル目に実行した場合
                          v_succ が最短で実行できるサイクル;
                }
                fastest = i|c_i が最小となる i;
                if (fastest = 0) {
                    READY = READY − {v_k};
                    v_k = NULL;
                    if (READY = ϕ) {
                        break; // while ループを抜ける
                    }
                }
            }
            v_sch = v_k;
        }
    }
    return v_sch;
end
```

図 6　発見的手法による命令スケジューリングアルゴリズム：GetScheduleNode 関数

か NOP 命令を実行

- 実行可能な命令数が多く，レジスタに演算結果が書き込まれるまで別の処理を進めることが出来る場合はそのまま実行

提案するスケジューリングアルゴリズムを図 5, 6 に示す．スケジューリングアルゴリズムの概略は以下の通りである．

（1） 実行可能な未スケジューリング命令に対して，各命令の最大パス長を優先度として求め，最も優先度の高い命令をスケジュール候補とする

（2） スケジュール候補命令の各後続命令と候補命令との有効命令距離の L の値が現在実行可能な命令の数以下の場合，選んだ候補を現在のサイクルにスケジュールする．そうでない場合は，スケジュール候補命令の直接の後続命令を調べ，後続命令の最短スケジュール位置を求める

（3） スケジュール候補命令を現在のサイクルでスケジュールすることで，後続命令を最も早く実行できると判定されれば候補命令をスケジュールする．そうでないと判定された場合は次に優先度の高い命令をスケジュール候補として 2, 3 の手順を繰り返す

（4） 全命令をスケジュールするまで 1-3 を繰り返す

まず (1) で，クリティカルパス上の命令を優先的にスケジュールする．ただし，部分フォワーディングプロセッサでは，データ依存グラフの各辺の重みは命令距離の集合で表現される有効命令距離であるので，パス長をどう計算するかを考える必要がある．提案手法では，後続命令のパス長を計算する際には，各枝が持つ有効命令距離の最小値を重みとして計算する．つまり，D 中の最小値か，D が食う集合であれば L の値を重みとする．クリティカルパスを求めた後，クリティカルパス上で実行可能な命令があれば，その命令を，現在のサイクルで実行するかどうかをこの後のフェーズで決定する．以後この命令を発行候補命令とする．クリティカルパス上に実行可能な命令がなければ，クリティカルパス以外で最も長いパス上の命令を探す．発行候補命令が見つかるまでこれを繰り返す．

表 2　先行命令で定義した値が後続命令で使用可能となる命令間隔

先行命令からの命令距離	0	1	2	3	4
DLX_no	×	×	×	×	○
DLX_X	×	○	×	×	○
DLX_X	×	×	○	×	○
DLX_W	×	×	×	○	○
DLX_XM	×	○	○	×	○
DLX_XW	×	○	×	○	○
DLX_MW	×	×	○	○	○
DLX_XMW	×	○	○	○	○

次に (2) で，発行候補命令以外に実行可能な命令を数え，その数が閾値以上であれば，発行候補命令を現在のサイクルで実行するようにスケジュールする．この時，閾値は発行候補命令に従属する後続命令のうち，クリティカルパス上にある命令が満たすべき有効命令距離の L の値である．これは，発行候補命令の演算結果がレジスタに書き込まれるまで，別の命令を実行できる状態に対応する．

発行候補命令以外に実行可能な命令数が閾値を超えない場合は (3) に進み，発行候補命令が現在のサイクルで実行される場合と，それよりも遅いサイクルで実行される場合とで，発行候補命令に直接従属している後続命令が最短で何サイクル目に実行できるかを計算する．この時，クリティカルパス上の命令をすぐに実行することで命令列全体の実行サイクル数が短くなるであろうという仮定をおく．計算の結果，発行候補命令を現在のサイクルで実行するのが最短であれば，発行候補命令を現在のサイクルで実行するようにスケジュールする．そうでなければ，発行候補命令のスケジュールを保留し，別の命令を探す．これを，(4) で示したとおり，全命令がスケジュールされるまで繰り返す．もし，実行できる命令が存在しない，もしくは，どの命令も現在のサイクルで実行しないほうがよい，となった場合は NOP 命令をスケジュールする．

4. 評価実験

提案手法が有用であるかどうかを評価するために，提案手法を組み込んだコンパイラを作成し，最適解を求めることが出来る整数計画法によるスケジューラ [6] との比較を行った．部分フォワーディング回路を持つプロセッサを，ASIP Meister [11] を用いて作成し，そのプロセッサを対象としてコンパイルを行った．コンパイラは CoSy [12] を利用して作成した．

実験で作成したプロセッサは，整数命令のみを持つ DLX プロセッサに対して，フォワーディング回路に変更を加えた 8 つのプロセッサである．表 2 に，各プロセッサごとの有効命令距離の表を示す．

実験に使用した環境は以下の通りである．整数計画法による命令スケジューラは，0-1 整数計画問題のソルバである opbdp [13] [14] を使用している．

- CPU : Xeon 2.8GHz x2 (2 個)
- メモリ : 1024MB
- OS : Redhat Linux 7.2

表 3　作成したプロセッサの性能見積もり

	program		
processor	最大動作周波数 [MHz]	面積 [Gate]	消費電力 [mW]
DLX_no	141.64	34961	7.92
DLX_X	124.38	37241	8.11
DLX_M	128.87	38188	8.46
DLX_W	130.72	38853	8.77
DLX_XM	120.34	40383	8.59
DLX_XW	119.76	41083	8.90
DLX_MW	127.39	42071	9.24
DLX_XMW	119.05	44077	9.36

表 4　fir の実行時間

processor	提案手法 [μs]	整数計画法 [μs]	最適解からの増加率 [%]
DLX_no	15.9	15.9	0.0
DLX_X	18.0	17.6	2.3
DLX_M	17.8	17.0	5.0
DLX_W	16.7	16.7	0.0
DLX_XM	14.5	14.2	2.1
DLX_XW	19.0	18.3	3.8
DLX_MW	17.4	17.4	0.0
DLX_XMW	18.4	18.4	0.0

表 5　fir2dim の実行時間

processor	提案手法 [μs]	整数計画法 [μs]	最適解からの増加率 [%]
DLX_no	107.3	107.3	0.0
DLX_X	110.2	107.4	2.6
DLX_M	112.1	108.3	3.5
DLX_W	109.2	109.2	0.0
DLX_XM	115.0	114.5	0.4
DLX_XW	116.6	115.1	1.3
DLX_MW	109.3	109.3	0.0
DLX_XMW	115.7	115.7	0.0

コンパイル対象のテストプログラムには DSPStone [15] の 1 次元 FIR である fir と，2 次元 FIR プログラムである fir2dim を使用した．

4.1 整数計画法による命令スケジューリングとの比較

提案手法によるスケジューリング結果が最適解に近いものかどうかを確かめるために，コンパイル結果の命令列の実行時間を比較した．以下に，整数計画法による命令スケジューリング手法と提案手法の実験結果を示す．比較項目は，各サンプルプログラムの実行時間とコンパイルに要した時間である．実行時間は HDL シミュレーションを行った結果から，実際のプロセッサ上で実行した場合の実行時間を算出した．ここで，プロセッサの動作周波数は，論理合成を行って得られた最大動作周波数の見積もり値としている．各プロセッサの最大動作周波数，面積，消費電力の見積もり結果を表 3 に示す．

fir, fir2dim の実行時間の比較結果をそれぞれ表 4, 5 に示す．各表中の最後の列の項目は，整数計画法による命令スケジューリングを用いて得られた最適解の実行時間からどれだけ実行時間が増加しているかの割合を示す．提案手法による命令スケジューリングで最適解を得られた場合は，増加率は 0%となる．

実験結果では，提案手法によって得られるスケジューリング結果は最適解から 100〜105%の範囲に収まっていることが分か

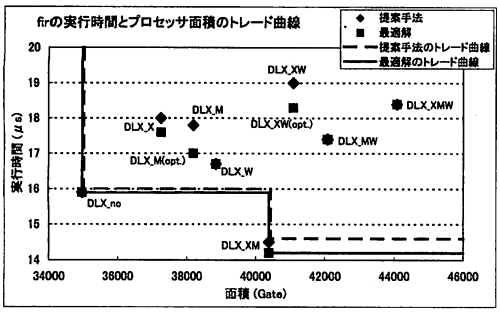

図 7 fir のトレードオフ曲線

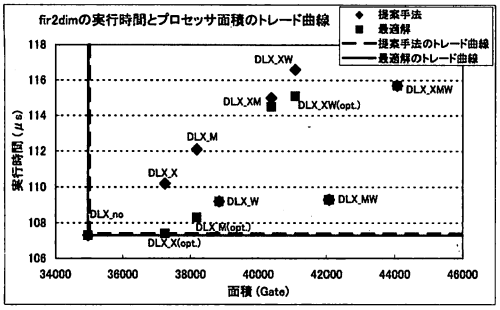

図 8 fir2dim のトレードオフ曲線

る．この結果から，提案手法によって得られる解が最適解と同程度の品質であるといえる．

コンパイル時間については，整数計画法では fir の場合平均 9.1 秒，最長で 54.5 秒で，fir2dim の場合平均 43.0 秒，最長で 267 秒であったが，提案手法を用いた場合は fir, fir2dim どちらの場合でも全て 0.001 秒でコンパイルが完了した．0.001 秒は測定可能な時間の下限である．これは，通常のコンパイルだけでなく，部分フォワーディングの設計空間探索を行う上でも十分早いコンパイル時間であるといえる．

4.2 設計空間探索

表 3 の見積もり結果に基づいて，整数計画法による手法と提案手法とで設計空間探索の結果に違いが出るかどうかを確かめた．整数計画法によるコンパイル結果が得られた fir と fir2dim について，実行時間と面積のトレード曲線をそれぞれ図 7, 8 に示す．プロセッサの動作周波数は最大動作周波数で動作させると仮定している．

各グラフにおいて，整数計画法による手法と提案手法とでパレート曲線上にあるプロセッサは変わらない．つまり，設計空間探索に違いは生じないことが分かる．これらの結果から，提案手法は整数計画法による手法と同等の性能を有するといえる．

5. ま と め

本稿では，部分フォワーディング回路を持つプロセッサに対応した発見的手法による命令スケジューリングアルゴリズムを提案した．部分フォワーディングを持つプロセッサに対して，提案手法を組み込んだコンパイラを用いて評価実験を行ったところ，整数計画法によって求めた最適解と比較しても遜色のない品質のコードを生成することを確認した．また，スケジューリングに要する時間は整数計画法によるものと比べて短く，実用的に使用できるものであることを確認した．実用サイズのプログラムに対しても，提案手法による命令スケジューリングアルゴリズムが適用できることも確認した．

また，提案手法と整数計画法を用いた手法とで設計空間探索の結果が変わらなかったことから，提案手法を用いることで，同じ時間でより多くのシミュレーションを行うことが可能となり，部分フォワーディングに関する設計空間探索を効率的に行えることを確認した．

今後の課題としては，アルゴリズムの改良のほかに，EDA ツールにおいて部分フォワーディングの設計が簡単に行えるような設計手法の確立と，部分フォワーディングコンパイラの自動生成手法の確立などがあげられる．

文 献

[1] A. Capitanio, N. Dutt, and A. Nicolau, "Partitioned register files for VLIWs: a preliminary analysis of tradeoffs, " Proc. of International Symposium on Microarchitecture (MICRO-25), pp.292-300, 1992.

[2] A. Abnous and N. Bagherzadeh, "Pipelining and Bypassing in a VLIW Processor, " IEEE Trans. on Parallel and Distributed Computing, vol. 5, no. 6, pp.658-664, 1994.

[3] Pritpal S. Ahuja, Douglas W. Clark and Anne Rogers, "The Performance Impact of Incomplete Bypassing in Processor Pipelines, " Proc. of MICRO-28, pp.36-45, 1995.

[4] A. Shrivastava, E. Earlie, N. Dutt and A. Nicolau, "Operation Tables for Scheduling in the Presence of Incomplete Bypassing, " Proc. of CODES+ISSS, pp.194-199, 2004.

[5] M. Kudler, K. Fan, M. Chu, R. Ravindran, N. Clark and S. Mahlke, "FLASH: Foresighted Latency-Aware Scheduling Heuristic for Processors with Customized Datapaths, " Proc. of Code Generation and Optimization (CGO), pp. 201-212, 2004.

[6] Takuji Hieda, Hiroaki Tanaka, Keishi Sakanushi, Yoshinori Takeuchi and Masaharu Imai, "Optimal Instruction Scheduling for Processors with Partial Forwarding using Integer Programming" Proc. of SASIMI2006, pp.274-279, 2006.

[7] John L. Hennessy, David A. Patterson, "Computer Architecture: A Quantitative Approach, " Moran Kaufmann, 1996. Second Edition.

[8] 橋本昭洋, "計算機アーキテクチャ, " 昭晃堂, 1995.

[9] A. V. Aho, R. Sethi, J. D. Ullman, "コンパイラ II, " サイエンス社, 1993.

[10] 中田育男, "コンパイラの構成と最適化, " 朝倉書店, 1999.

[11] 今井 正治, 武内 良典, 塩見 彰睦, 佐藤 淳, 北嶋 暁, "特定用途向きプロセッサ開発システム ASIP Meister," 電子情報通信学会技術研究報告, DSP2002-125, vol. 102, No. 399, pp. 39-44, Oct., 2002

[12] ACE - Associated Computer Experts bv. "The CoSy Compiler Development System," http://www.ace.nl.

[13] P. Barth, "A Davis-Putnam based Enumeration Algorithm for Linear Pseudo-Boolean Optimization, " Technical Report MPI-I-95-2-003, Max-Planck-Institut Fur Informatik, 1995.

[14] P. Barth, "OPBDP: A Davis-Putnam based Enumeration Algorithm for Linear Pseudo-Boolean Optimization, " http://www.mpi-sb.mpg.de/units/ag2/software/opbdp.

[15] V. Zivojnovic, J. Martinez, C. Schlger and H. Meyr, " "DSPstone: A DSP-Oriented Benchmarking Methodology, Proc. of ICSPAT'94 - Dallas, pp. 715-720, 1994.