計算機アーキテクチャ 121－5
設 計 自 動 化 82－5
（1996. 12. 12）

特定用途向き集積化プロセッサのレジスタ数最適化アルゴリズム

本間啓道† 今井正治†† 武内良典††

†豊橋技術科学大学大学院 工学研究科 システム情報工学専攻
〒441 愛知県豊橋市天伯町字雲雀ヶ丘1-1

†† 大阪大学大学院 基礎工学研究科 情報数理系専攻
〒560 大阪府豊中市待兼山町1-3

E-Mail: peasv@vlsilab.ics.es.osaka-u.ac.jp

あらまし PEAS-Iは, ASIPを開発するためのハードウェア／ソフトウェア・コデザイン・システムである. PEAS-Iを用いることにより, 設計条件(ゲート数, 消費電力)の範囲以内で特定分野の応用プログラムを最も高速に実行するCPUコアとそのCPUコアをターゲットとする応用プログラム開発環境を同時に生成することが出来る. 本稿では, 応用プログラムの実行サイクル数に対するレジスタの個数と命令セットのおよぼす影響について考察し, 与えられた設計条件のもとで応用プログラムの実行サイクル数が最小になるレジスタの個数と命令セットを決定するアルゴリズムについて述べる. また, 本手法を用いることで最適なレジスタ数と演算器構成を持ったCPUコアを短時間で設計できることを示す.

和文キーワード レジスタ数の最適化, ASIP, ハードウェア／ソフトウェア協調設計

A Register Count Optimization Algorithm for ASIPs

Yoshimichi Honma† Masaharu Imai†† Yoshinori Takeuchi††

†Department of Information and Computer Science, Toyohashi University of Technology
1-1 Hibarigaoka, Tenpaku-cho, Toyohashi, Aichi, 441 Japan

††Department of Computer Science, Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka, 560 Japan

E-Mail: peasv@vlsilab.ics.es.osaka-u.ac.jp

Abstract PEAS-I is a hardware/software codesign system for ASIP (Application Specific Integrated Processor) development. For a given set of application programs with their associated input data from an application domain, the PEAS-I can generate the highest performance CPU core design under given design constraints (gate count, power consumption, etc.) as well as a set of application program development tools (C compiler, simulator, etc.). This paper focuses on optimizing both a register file and the selection of functional units(FUs) to define an optimal instruction set, also describes a method to achieve the best tradeoff between the register count and selected FUs. The experimental results show that very short design turn-around-time has been achieved.

英文key words Register Count Optimization, ASIP, Hardware/Software Codesign

1 はじめに

近年の半導体集積度の向上に伴い，ASIP の規模が大きくなり，設計期間が長期化する傾向がある．ASIP の設計期間を短縮し，ハードウェアとソフトウェアのバランスがとれたシステムを設計するためには，協調設計が必要である．協調設計の目的は，ハードウェアとソフトウェアの最適な機能分割と，設計期間の短縮である．著者らは，ASIP 開発のための協調設計システム PEAS (Practical Environment for ASIP Development) を提案し，PEAS-I システムを試作した [1].

PEAS-I では IMSP (Instruction set implementation Method Selection Problem)[2]-[4] を組み合わせ最適化問題として定式化し，これらの最適化問題を解くことによって命令セットを決定している．これまでの PEAS-I システムは，最適なレジスタ数を自動的に決定する手段を持たないため，レジスタ数はユーザによって指定されていた．しかし，レジスタ数と命令セットの間にはトレードオフが存在し，レジスタ数を考慮した最適な命令セットを決定することが困難であった．

本稿ではレジスタの個数と命令セットを同時に最適化するアルゴリズムを提案し，実験によってその有効性を確認した結果について報告する．

2 PEAS-I システム

2.1 概要

PEAS-I システム (図 1) は，C 言語で記述された応用プログラムとデータを入力し，与えられた制約条件のもとで最大の性能を持つ CPU コアを自動設計すると共に，その CPU コアのための C コンパイラ，シミュレータ等の応用プログラム開発ツールを自動生成するシステムである [1].

PEAS-I では HDL として NTT で開発された高位合成システム PARTHENON の SFL (Structured Function description Language) を用いている [5]．ハードウェア生成系で生成された CPU コアの HDL 記述は，PARTHENON を用いて論理合成され，ネットリストの形式で出力される．また，C コンパイラは，GNU C コンパイラ [6] を利用して自動生成される．

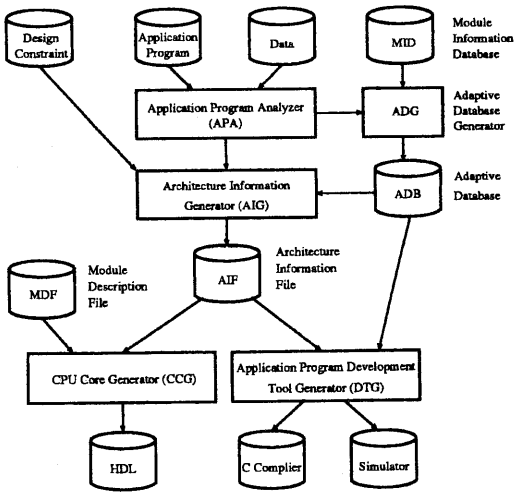

図 1: PEAS-I システムの構成

2.2 CPU コアのアーキテクチャ

図 2に PEAS-I で生成される CPU コアのデータパス部を示す．PEAS-I で生成される CPU コアは以下の特徴を持つ．

- 32 ビットのハーバード・アーキテクチャ
- ロード／ストア・アーキテクチャ
- 3 アドレスの汎用レジスタ方式
- GCC の RTL に対応した命令セット
- パイプライン制御

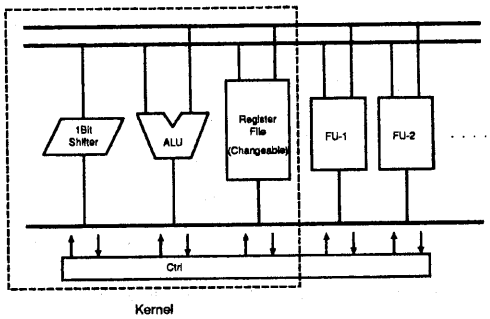

図 2: PEAS-I CPU コアのデータパス部

2.3 IMSP による命令セットの最適化

PEAS-I は IMSP (Instruction set implementation Method Selection Problem) を組み合わせ最適化問題として定式化し，この最適化問題を解

くことによって命令セットを決定している [2]-[4]. IMSP は応用プログラムの実行サイクル数，ゲート数，消費電力のうちの 2 つを設計条件とし残りの 1 つを目的関数として最適化を行う．

CPU コアの持つ演算器を応用プログラムの実行に最小限必要な機能を実現する kernel (図 2参照) とその他の演算器に分けて考え，kernel で実行する命令群を PRTL(Primitive RTL[1])，その他の演算器で実行される命令群を BRTL(Basic RTL) と呼ぶ．CPU コアがある BRTL を直接実行する演算器を持たない場合，その BRTL は PRTL を組合せてソフトウェアとして実現される．IMSP は与えられた設計条件のもとで目的関数に合わせて BRTL を実行する演算器の選択を行うことによって命令セットを決定する．IMSP は演算器の構成だけを最適化し，レジスタの個数を考慮していない．

2.4 レジスタ数と演算器構成のトレードオフ

PEAS-I で生成される CPU コアはロード/ストア・アーキテクチャであるため，演算に必要なデータはレジスタ上に置かれる必要がある．CPU コアが応用プログラムのワーキングセットすべてを載せるだけのレジスタを持たない場合はデータをメモリからロード，およびメモリへストアする必要があり応用プログラムの実行サイクル数は増加する．

CPU コアの面積が設計条件として与えられている場合を考える．このときはレジスタに割り当てられたハードウェア・リソースを演算器に割り当てることにより高速な演算器を使用することができることがある．すなわち，レジスタ数を少なくしてデータの退避/復帰に時間がかかっても，より高速な演算器を持つことによって演算時間を短くした方が応用プログラムの実行時間を短縮できる可能性がある．

このトレードオフを考慮し，レジスタの個数と演算器構成を決定しなければならない．

3 応用プログラムのモデル

レジスタの個数および演算器構成を決定するために，はじめに応用プログラムの実行時間モデルを作成する．応用プログラムの実行サイクル数を以下の式であらわす．

$$T(r, \boldsymbol{X}) = T_{sr}(r, \boldsymbol{X}) + T_c(\boldsymbol{X}) \qquad (1)$$

r : レジスタの個数

$\boldsymbol{X}$: 演算器の構成

$T(r, \boldsymbol{X})$: レジスタ数が r個，演算器構成が $\boldsymbol{X}$ の場合の応用プログラムの実行サイクル数

$T_{sr}(r, \boldsymbol{X})$: データの退避/復帰に要する実行サイクル数

$T_c(\boldsymbol{X})$: レジスタ上のデータの演算，操作に要する実行サイクル数

$T'_c(r)$: CPU コアのゲート数が一定であるときレジスタの個数を r個として残りのハードウェア・リソースで演算器を構成したときの演算，操作に要する実行サイクル数

$T_c(\boldsymbol{X})$ は演算に必要なデータがすべてレジスタ上に存在している場合の応用プログラムの実行時間を表す．設計条件を与えて IMSP を解くことによって数パーセントの誤差で見積もることが可能である [?]．図 3に $T_c(\boldsymbol{X})$ とハードウェアコストの関係を示す．横軸に面積，縦軸に時間をとると $T_c(\boldsymbol{X})$ は単調減少の階段状のグラフになる．

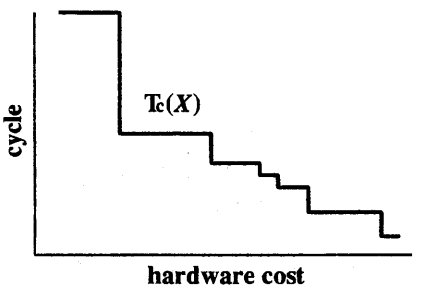

図 3: 演算に必要なサイクル数と演算器のハードウェアコストの関係

ここで，$\boldsymbol{X}$の上限と下限を $\boldsymbol{X}_F$, $\boldsymbol{X}_K$で表すことにする．$\boldsymbol{X}_F$は，すべての BRTL がライブラリ中のもっとも高速なハードウェアで実現されている演算器構成を表す．$\boldsymbol{X}_K$は，すべての BRTL がソフトウェアで実現されている演算器構成 (kernel 以外に演算器を持たない構成) を表す．

[1]RTL は，GCC の中間言語で用いられる抽象的な演算を記述する言語 (Register Transfer Language) である．

CPU コアの持つレジスタの個数が等しい場合，レジスタの退避/復帰に要する時間は演算器の構成によって決まる．，以下の関係を満たす．$T_{sr}(r, \boldsymbol{X})$ と $\boldsymbol{X}$，$\boldsymbol{X}_F$，$\boldsymbol{X}_K$ について以下の関係が成り立つ．

$$T_{sr}(r, \boldsymbol{X}_F) \leq T_{sr}(r, \boldsymbol{X}) \leq T_{sr}(r, \boldsymbol{X}_K) \quad (2)$$

ソフトウェアで実現した CPU コアは，すべての BRTL をハードウェアで構成した CPU コアに比べて BRTL に含まれる演算を行うために，中間結果やカウンタなどの中間変数を必要とする．これらの変数が増えるため，レジスタを確保できない変数が出る可能性がある．レジスタを確保出来ない変数はメモリにおかれ必要に応じてロード/ストアされるので $T_{sr}(r, \boldsymbol{X})$ も増加する．ソフトウェアで実現される BRTL が多いほど $T_{sr}(r, \boldsymbol{X})$ は多くなる．さらに，$T_{sr}(r, \boldsymbol{X})$ はすべての BRTL がソフトウェアで実現される演算器構成 $\boldsymbol{X}_K$ のとき最大になる．図 4にレジスタ数と $T_{sr}(r, \boldsymbol{X}_F)$，$T_{sr}(r, \boldsymbol{X})$，$T_{sr}(r, \boldsymbol{X}_K)$ の関係をしめす．$T_{sr}(r, \boldsymbol{X})$ は $\boldsymbol{X}$ を固定した場合，コンパイラのレジスタ割り付け戦略が適切に行われれば単調減少のグラフになる．$T_{sr}(r, \boldsymbol{X}_F)$ は $T_{sr}(r, \boldsymbol{X}_K)$ に比べて常に小さい．$T_{sr}(r, \boldsymbol{X})$ は $T_{sr}(r, \boldsymbol{X}_F)$ と $T_{sr}(r, \boldsymbol{X}_K)$ に囲まれた図中の斜線部に存在する．

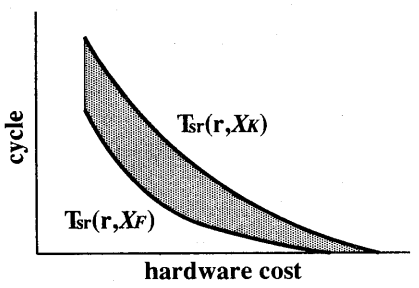

図 4: レジスタの退避/復帰に要するサイクル数とレジスタ数との関係

r と $T(r, \boldsymbol{X})$ の関係を図 5に示す．

4 レジスタ数と演算器構成の最適化

前節のモデルを用いて最適なレジスタ数と演算器の構成を決定する方法について述べる．

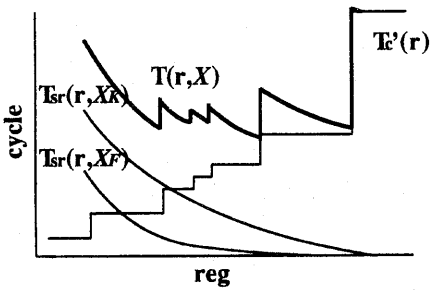

図 5: ゲート数を制約条件にした場合のレジスタ数と応用プログラムの実行サイクル数関係

4.1 記号の定義

以下では，次の記法を用いる．

A_R : レジスタ 1 個分の等価ゲート数
A_{max} : ゲート数に関する設計制約
$A(\boldsymbol{X})$: $\boldsymbol{X}$ を実現するのに必要なゲート数
A_F : フル構成の演算器を実現するのに必要なゲート数
A_K : kernel のみを実現するのに必要なゲート数
$\boldsymbol{X}_{opt}$: 最適な演算器構成
R_{opt} : 最適なレジスタ数
R_{min} : コンパイラがコード生成に最低限必要なレジスタ数
R_F : 演算器構成が $\boldsymbol{X}_F$ のときのコンパイラが要求する最大レジスタ数
R_K : 演算器構成が $\boldsymbol{X}_K$ のときのコンパイラが要求する最大レジスタ数
$T_{A_{max}}(r)$: CPU コアのゲート数一定としたときレジスタを r 個もった場合の応用プログラムの実行サイクル数．

$$T_{A_{max}}(r) = T_{sr}(r, \boldsymbol{X}) + T'_c(r) \quad (3)$$

ただし，ここでの $\boldsymbol{X}$ は IMSP に制約条件として $A_{max} - A_R \cdot r$ を与えて求めた演算器構成．

4.2 最適解の存在する範囲

定理 1 最適なレジスタの個数 R_{opt} について次の関係が成り立つ．

$$max\left(\frac{A_{max} - A_F}{A_R}, R_{min}\right) \leq R_{opt} \leq min\left(\frac{A_{max} - A_K}{A_R}, R_K\right)$$

証明 まずCPUコアの総ゲート数はゲート数に関する設計条件 A_{max} を越えてはならないので，

$$A(\boldsymbol{X}) + A_R{\cdot}r \leq A_{max} \tag{4}$$

が成り立つ．また，演算器を構成するために，最低限 A_K は確保しなければCPUコアが設計できないので，レジスタファイルを構成するのに使用できる面積は

$$A_R{\cdot}r \leq A_{max} - A_K \tag{5}$$

となる．一方，演算器を構成するのに必要な面積はたかだか A_F なので，

$$A_R{\cdot}r \geq A_{max} - A_F \tag{6}$$

となる．式(5)，(6)より，最適なレジスタの個数 R_{opt} は

$$\frac{A_{max} - A_F}{A_R} \leq R_{opt} \leq \frac{A_{max} - A_K}{A_R} \tag{7}$$

の範囲にあることがわかる．

次に，設計されたCPUコアが R_{min} より少ないレジスタしか持たない場合はコンパイラがオブジェクトコードを生成することが出来ないので，最低 R_{min} 個のレジスタが必要である．一方，R_K より多くのレジスタをCPUコアが持っていたとしてもコンパイラは R_K 個のレジスタしか使わないので，無駄になる．したがって，最適なレジスタの個数 R_{opt} は

$$R_{min} \leq R_{opt} \leq R_K \tag{8}$$

の範囲にあることがわかる．よって，最適なレジスタの個数 R_{opt} は式(7)，(8)の両式を満たす

$$max\left(\frac{A_{max} - A_F}{A_R}, R_{min}\right) \leq R_{opt} \leq min\left(\frac{A_{max} - A_K}{A_R}, R_K\right)$$

の範囲にあることがわかる． □

系 1 最適な演算器構成 $\boldsymbol{X}_{opt}$ について次の関係が成り立つ．

$$max(A_{max} - A_R{\cdot}R_K, A_K) \leq A(\boldsymbol{X}_{opt}) \leq min(A_{max} - A_R{\cdot}R_{min}, A_F)$$

証明 CPUコアの総ゲート数がゲート数に関する設計条件 A_{max} を越えてはならない(式(4)参照)．レジスタファイルを構成するために，最低限 $A_R{\cdot}R_{min}$ は確保しなければCPUコアが設計できないので，演算器を構成するのに使用できるゲート数は

$$A(\boldsymbol{X}) \leq A_{max} - A_R{\cdot}R_{min} \tag{9}$$

となる．一方，コンパイラが使用するレジスタ数はたかだか R_K なのでレジスタファイルを構成するのに必要なゲート数はたかだか $A_R{\cdot}R_K$ である．

$$A(\boldsymbol{X}) \geq A_{max} - A_R{\cdot}R_K \tag{10}$$

式(9)，(10)より，最適な演算器構成 $\boldsymbol{X}_{opt}$ は次式の関係を満たす．

$$A_{max} - A_R{\cdot}R_K \leq A(\boldsymbol{X}_{opt}) \leq A_{max} - A_R{\cdot}R_{min} \tag{11}$$

また，演算器構成の上限と下限はそれぞれ $\boldsymbol{X}_F$，$\boldsymbol{X}_K$ であるため，$\boldsymbol{X}_{opt}$ は次式の関係を満たす．

$$A_K \leq A(\boldsymbol{X}_{opt}) \leq A_F \tag{12}$$

ここで，A_K は $A(\boldsymbol{X}_K)$，A_F は $A(\boldsymbol{X}_F)$ をそれぞれあらわす．よって，最適な演算器構成 $\boldsymbol{X}_{opt}$ は式(11)，(12)をまとめた次式の関係を満たす．

$$max(A_{max} - A_R{\cdot}R_K, A_K) \leq A(\boldsymbol{X}_{opt}) \leq min(A_{max} - A_R{\cdot}R_{min}, A_F)$$

□

定理 2 任意の $r \geq 1$ について式 *(13)* を満たす R_0 が存在するならば $R_{opt} < r$ である．

$$T_c'(r) > T_{A_{max}}(R_0) \tag{13}$$

証明 データの退避/復帰に要するサイクル数 $T_{sr}(r, \boldsymbol{X})$ は0より小さくはならない．

$$T_{sr}(r, \boldsymbol{X}) \geq 0 \tag{14}$$

また，$T_c'(r)$ は右上がりの階段状になるため，次式の関係が成り立つ．

$$T_c'(i) \leq T_c'(j),\ (for\ i \leq j) \tag{15}$$

今，次式を満たす R_0 が存在したとすると，

$$T_c'(r) > T_{A_{max}}(R_0) \tag{16}$$

$T'_c(r)$ と $T_{sr}(r, \boldsymbol{X})$ の和である $T_{A_{max}}(r)$ は式 (14) より次式の関係を満たす．

$$T_{A_{max}}(r) > T_{A_{max}}(R_0) \qquad (17)$$

したがって，rは R_{opt}ではあり得ない．次に rより大きい r'を考える．式 (15) より

$$T'_c(r') > T_{A_{max}}(R_0) \qquad (18)$$

となり rと同様に r'は R_{opt}ではあり得ない．レジスタ数 r以上は最適なレジスタ数 R_{opt}ではあり得ないので $R_{opt} < r$である． □

上記の定理1と定理2により探索空間を限定することができる．

また，応用プログラムの実行サイクル数 $T(r, \boldsymbol{X})$ は階段状のグラフと単調減少のグラフの和になるので，最小になる可能性のある点は階段状に変化する点に限られる．よって図6の○印の点だけを調査すれば良い．

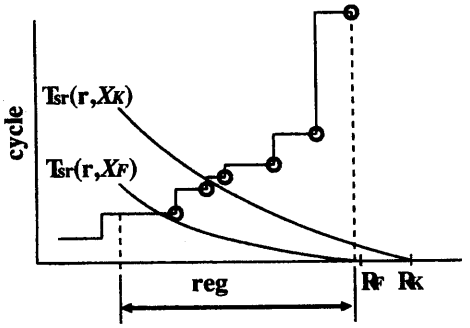

図 6: 最適な組合せの存在する範囲

4.3 最適化の手順

前節の定理を用いることにより設計条件 A_{max} の制約下における最適なレジスタ数 R_{opt}と最適な演算器構成 $\boldsymbol{X}_{opt}$は以下の手順により求めることができる．

1. $\boldsymbol{X}_K$用のコンパイラを用いてコード生成を行い，R_Kを求める．
2. IMSP を解き $T_c(\boldsymbol{X})$ を求める．
3. 設計条件に基づいて $T'_c(r)$ を求める．
4. 手続き RC_decision を実行する．

手続き RC_decision を図 7に示す．

```
procedure RC_decision
begin
    r := max( (A_max − A_F)/A_R , R_min );
    if (r は候補ではない) then
        r := next(r);
    R_end := min( (A_max − A_K)/A_R , R_K );
    T_opt := ∞;
    repeat
        if T'_c(r) + T_sr(r, X) < T_opt then
        begin
            T_opt := T'_c(r) + T_sr(r, X);
            R_opt := r;
            X_opt := X;
        end;
        r := next(r);
    until (r > R_end) or (T'_c(r) > T_opt)
    write ( R_opt, X_opt );
end.
```

図 7: 手続き RC_decision

関数 $next(r)$ は次のレジスタ数の候補 ($T'_c(r)$ が階段状に変化する r) を返す．$T(r, \boldsymbol{X})$ を求めるためにはシミュレーションが必要である．シミュレーションは応用プログラムの複雑さ (長さ，大きさ，動作内容) に応じたシミュレーション時間がかかるため，設計時間はシミュレーションの回数に比例する．手続き RC_decision は定理1，2に基づいて候補を絞り込み，シミュレーションの回数を減じている．

5 実験結果

5.1 予備実験

サンプルプログラムとして FFT を使用した．実験に用いたライブラリではレジスタ 1 個当たりの等価ゲート数 A_Rは 464 ゲート，最低限の演算器構成 $\boldsymbol{X}_K$を構成するために必要なゲート数 A_Kは 11206ゲート，演算器構成 $\boldsymbol{X}_F$を構成するのに必要なゲート数 A_Fは 30590ゲートであった．

また，実験に用いたコンパイラではコード生成に最低限必要なレジスタ数 R_{min}は 10 個，演算器が kernel のみの構成のコンパイラが要求する最大

表 1: 各パラメータの値

A_R	464
A_K	11206
A_F	30590
R_{min}	10
R_K	33

表 2: 演算器の組合せと必要なゲート数

No.	gate	modules
1	29697	kernel mul_csa div_2seq_p9
2	24761	kernel div_2seq mul_csa
3	24411	kernel mul_csa div_seq_p17
4	22349	kernel div_seq mul_csa
5	18953	kernel mul_csa
6	17324	kernel mul_3clk
7	16606	kernel mul_bpr_p8
8	14367	kernel mul_bpr
9	13599	kernel mul_seq
10	11206	kernel

のレジスタ数R_Kは33個であった．以上の予備実験の結果を表1に示す．また可能な演算器の構成を表2に示す．

5.2 アルゴリズムの適用例

ゲート数に関する設計条件A_{max}が3万ゲートの場合，定理1と表1の値から

$$max\left(\frac{30000-30590}{464},10\right)=10$$

$$min\left(\frac{30000-11206}{464},33\right)=33 \quad .$$

$T'_c(r)$は図8のようになった．この図から候補が7点に絞られる．以下RC_decisionを実行したときの様子を示す．

(1) 繰り返し1回目 r=11, $\boldsymbol{X}$は表2のNo.2. シミュレーションを行って$T(r,\boldsymbol{X})$=41447. T_{opt}, R_{opt},$\boldsymbol{X}_{opt}$を設定．次の候補はr=12< 33, $T'_c(r)$ = 23948< T_{opt}なのでループを継続する．

r	$\boldsymbol{X}$	$T(r,\boldsymbol{X})$	$T'_c(r)$
11	No.2	41447	
12	No.3		23948

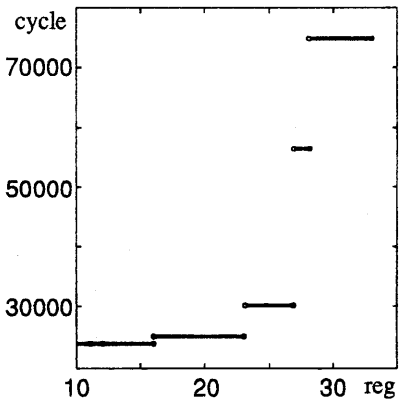

図 8: rと$T'_c(r)$

(2) 2回目 r=12, $\boldsymbol{X}$は表2のNo.3. シミュレーションを行って$T(r,\boldsymbol{X})$=40083. T_{opt}, R_{opt},$\boldsymbol{X}_{opt}$を更新．次の候補はr=16 < 33, $T'_c(r)$ = 23955 < T_{opt}なのでループを継続する．

r	$\boldsymbol{X}$	$T(r,\boldsymbol{X})$	$T'_c(r)$
11	No.2	41447	
12	No.3	40083	23948
16	No.4		23955

(3) 3回目 r=16, $\boldsymbol{X}$は表2のNo.4. シミュレーションを行って$T(r,\boldsymbol{X})$=32486. T_{opt}, R_{opt},$\boldsymbol{X}_{opt}$を更新．次の候補はr=23 < 33, $T'_c(r)$ = 25243 < T_{opt}なのでループを継続する．

r	$\boldsymbol{X}$	$T(r,\boldsymbol{X})$	$T'_c(r)$
11	No.2	41447	
12	No.3	40083	23948
16	No.4	32486	23955
23	No.5		25243

(4) 4回目 r=23, $\boldsymbol{X}$は表2のNo.5. シミュレーションを行って$T(r,\boldsymbol{X})$=29021. T_{opt}, R_{opt},$\boldsymbol{X}_{opt}$を更新次の候補はr=27 < 33, $T'_c(r)$ = 30177 > T_{opt}なのでループ終了．

r	$\boldsymbol{X}$	$T(r,\boldsymbol{X})$	$T'_c(r)$
11	No.2	41447	
12	No.3	40083	23948
16	No.4	32486	23955
23	No.5	29021	25243
27	No.6		30177

最適なレジスタの個数 R_{opt}= 23，最適な演算器構成 $\boldsymbol{X}_{opt}$は表 2 の No.5 の組合せに決定される．図 9に r=10～33 でシミュレーションを行なった結果を示す．$T(r, \boldsymbol{X})$ は r=23 のとき最小であることがわかる．

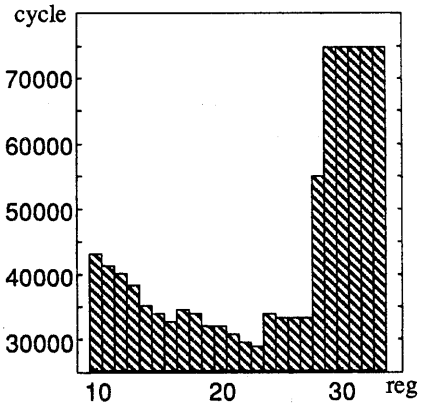

図 9: $T(r, \boldsymbol{X})$ $(10 \leq r \leq 33)$

レジスタの個数に対して総当たりで検査するためには 24 回のシミュレーションが必要となる．提案したアルゴリズムを用いた場合，1/6 の 4 回で最適なレジスタ数と演算器構成が決定できる．

6 考察

IMSP によって求めた $T_c'(r)$ は数%ではあるが誤差をともなう．RC_decision は探索を打ち切る条件として $T_c'(r) > T_{opt}$ を用いているので $T_c'(r)$ の誤差によって最適解を探し出せない可能性がある．これは探索範囲を限定するときに数%から 10%程度の余裕を持つことで回避できると思われる．

コンパイラによるレジスタの割り付けが適切に行われていなければデータの退避/復帰に要するサイクル数 $T_{sr}(r, \boldsymbol{X})$ は単調減少にならないため，応用プログラムの実行サイクル数 $T(r, \boldsymbol{X})$ が最小になる rが $T_c'(r)$ が階段状に変化する点だけに存在するとは限らなくなる．

7 おわりに

応用プログラムの実行サイクル数と CPU コアの持つレジスタ数，演算器構成の関係について考察し，特定用途向き集積化プロセッサ (ASIP) の CPU コアレジスタ数最適化アルゴリズムを提案した．提案したアルゴリズムは与えられた設計条件 (ゲート数) の基で応用プログラムのサイクル数が最小になるようにレジスタの個数と演算器構成を決定する．

実験により提案したアルゴリズムが最適なレジスタの個数と演算器構成を決定することを示した．また，単純に最適な組合せを探す方法に比べてシミュレーションの回数は 1/6 に削減された．

IMSP によって求めた $T_c'(r)$ の誤差，コンパイラのレジスタ割り付け失敗に対する対策は今後の課題である．

謝辞

本研究を進めるにあたり御討論いただいた静岡大学の塩見彰睦 講師，鶴岡高専の佐藤淳 講師，ならびに大阪大学 VLSI システム設計研究室の諸兄に感謝いたします．また，設計ツール及びライブラリ等を提供して頂いた NTT コミュニケーション科学研究所，VLSI テクノロジー社，ならびに研究をご支援頂く（株）SRA に感謝いたします．

参考文献

[1] Sato,J.,et al.: "PEAS-I: A Hardware/Software Codesign System for ASIP Development," IEICE Trans.,to apper

[2] Alomary,A.,et al.: "An ASIP Instruction Set Optimization Algorithm with Function Module Sharing Constraint," IEICE Trans., Vol.E76-A, No.10, pp1713-1720, 1993.

[3] N. N. Binh, M. Imai, A. Shiomi, N. Hikichi: "Optimal Instruction Set Design through Adaptive Database Generation," 電子情報通信学会論文誌,(英文誌 (A) 軽井沢ワークショップ特集), vol.E79-A, no.3, pp.347 - 353, March 1996.

[4] N.N. Binh, M. Imai, and A. Shiomi: "A New HW/SW Partitioning Algorithm for Synthesizing the Highest Performance Pipelined ASIPs with Multiple Identical FUs," Proc. of the European Design Automation Conference (EURO-DAC'96), pp. 126 - 131,Geneva, Swiss, Sep. 1996.

[5] NTT データ通信:「PARTHENON User's Manual」, 1989

[6] Stallman,R.: Using and Porting GNU C Compiler, Free Software Foundation, Inc., 1991