システムＬＳＩ設計技術　92－3
（１９９９．９．２０）

パイプライン段数を考慮したASIP設計最適化の検討

篠原 克哉　武内 良典　今井 正治　北嶋 暁

大阪大学 大学院基礎工学研究科 情報数理系専攻
〒560-8531 大阪府豊中市待兼山町 1-3
TEL: 06-6850-6626
FAX: 06-6850-6627
E-mail: peasv@vlsilab.ics.es.osaka-u.ac.jp

あらまし　従来のパイプライン・プロセッサ設計最適化手法では，初期にアーキテクチャの基本となるパイプライン段数を固定し，最適化を行ってきた．本稿では，パイプライン・プロセッサ設計最適化においてパイプライン段数もパラメタとして追加し，設計探索空間を広げることによって，一層適用範囲の広いASIP設計最適化手法を検討した結果について報告する．評価実験からプロセッサのパイプライン・ステージ数，動作周波数，プロセッサが使用するハードウェア・コストと，プログラムの実行時間の間にはトレード・オフが存在することが知られた．

キーワード　パイプライン段数最適化, 動作周波数最適化, ハードウェア・ソフトウェア・コデザイン

A Case Study of a Design Optimization for Pipelined ASIPs in Consideration of the Number of Pipeline Stages

Katsuya Shinohara　Yoshinori Takeuchi　Masaharu Imai　Akira Kitajima

Department of Informatics and Mathematical Science
Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama-cho, Toyonaka
Osaka, 560-8531 Japan
TEL: +81 6 6850 6626
FAX: +81 6 6850 6627
E-mail: peasv@vlsilab.ics.es.osaka-u.ac.jp

Abstract　In conventional design optimization methods for pipelined processors, the clock frequency is fixed first and the optimization method is performed on its architecture. In this report, a processor optimization method that treats processor pipeline stages as a parameter of the design to expand design exploration space of ASIPs is proposed. The experimental results show that the trade-offs among the number of pipeline stages, clock frequency, hardware cost of the processor and the execution time for a program exists.

Key words　Optimization of Processor Pipeline Stage, Clock Frequency Optimization, Hardware Software Codesign

1 はじめに

現在, ASIP(Application Specific Integrated Processor : 特定用途向きプロセッサ) の開発にハードウェア・ソフトウェア・コデザインの手法を用い, 最適なプロセッサの設計を行なう数々の研究が行なわれている [1, 2, 3, 4]. ハードウェア・コスト制約下で高性能なプロセッサを設計する時, 総実行サイクル数と動作周波数が重要となる. 従来の最適化手法の多くでは動作周波数が固定であると考えられていたが, 実際にはクロック周期と実行サイクル数の積で決定される実行時間が性能を決定し, 実行サイクル数の少ない設計が必ずしも最適な設計とはならない. そこで我々は, 動作周波数も最適化のパラメタとして考慮した手法を提案してきた [5]. プロセッサの動作周波数はパイプライン段数に大きく依存するが, パイプライン段数を考慮した最適化設計支援に関する研究はこれまではほとんど行なわれておらず, パイプライン演算器のクロック周期最適化 [6] に焦点をあてていた.

本稿では, プロセッサのパイプライン段数を, 動作周波数を決定する要因の 1 つとして最適化対象に含んだプロセッサの演算器構成最適化手法を提案する. 実験によってプロセッサのパイプライン・ステージ数, 動作周波数, プロセッサが使用するハードウェア・コストと, プログラムの実行時間の間にはトレード・オフが存在することを示す.

2 パイププライン段数とプロセッサ性能

一般に, 演算回路をパイプライン化することにより, 個々の演算の実行時間は短くならないが, 単位時間あたりの演算数は増加し, プログラムの平均実行時間は短縮される. スカラ・プロセッサにおいて, パイプラインが理想的に動作すると, 1 サイクルに 1 命令の動作が可能となる. しかしながら, 実際にはパイプラインの流れは種々の原因で待ちが生じ, 乱れる. 原因には, 1) 多サイクル・ステージ, 2) 資源の競合, 3) 時間逆転がある. 1) 多サイクル・ステージによる乱れは, パイプライン段数が増加することによって, 段数が少ない場合には 1 サイクルで終了する処理が複数サイクルを要することによる乱れである. パイプライン段数が多すぎる場合, 高速なメモリを使用しても, メモリアクセスに複数サイクルが必要となってしまう. 2) 資源の競合による乱れは, 処理をパイプライン化することによって, 同じ資源を複数の異なるステージが使用する場合に生じる. 資源競合は資源を多重化することで解決できる場合が多い. 3) 時間逆転による乱れは, 処理をパイプライン化するために, 先行命令の後続命令で必要とされるデータが, 後続命令が実行される際に, 利用できない状態に生じる. 例えば, パイプライン段数が多くなると分岐命令で計算した次命令のアドレスを得るまでに複数クロック・サイクル数必要となる. この時の遅延は一般に分岐遅延 (branch delay), 空いたサイクルは分岐遅延スロット (branch delay slot) と呼ばれる. コンパイラ技術によるスケジューリングで分岐遅延スロットを有効活用する改善方法があるが, スケジューリングによる改善には限界があるため, 常に分岐遅延によるペナルティを 0 にすることはできない. したがってプロセッサでは, パイプラインの段数が多くなると, パイプラインが理想的に動作することが難しくなる.

一方, パイプラインの段数が多ければ, 動作周波数は高くなり, プログラムの実行サイクル数が同じであれば動作周波数が高いほど高性能となるという利点もあるが, 総実行サイクル数とクロック周波数間にはトレードオフが存在し, パイプライン・ステージ数が大きく関与している. したがって, パイプライン段数と動作周波数を考慮したプロセッサ性能最適化問題を考える必要がある.

3 諸定義

3.1 プロセッサ性能

プロセッサのアプリケーション実行時間は式 (1) のように表せる.

$$\text{実行時間} = \frac{\text{総実行クロック・サイクル数}}{\text{クロック周波数}} \quad (1)$$

本稿では, ALU, レジスタファイル, プログラムカウンタなど, プロセッサを構成するための最小構成を, "*kernel*" と呼ぶ. プロセッサで使用する乗算器, 除算器などの機能ユニットを n 種類とすると, プロセッサ構成は "*kernel*" を含めた $(n+1)$ 組の変数 X で表せる.

$$X \triangleq (kernel, x_1, \cdots, x_n) \quad (2)$$

x_i : 機能 #i の実装方法
n : 機能ユニットの種類.

"$kernel$" は，レジスタ数，ステージ数などの組からなる．すなわち，

$$kernel = (\text{レジスタ数}, \text{ステージ数}, \cdots) \quad (3)$$

本稿では，アプリケーションが与えられた際の，プロセッサの性能を実行時間 $T(X)$ で評価する．

$$T(X) \triangleq \frac{1}{f_{ck}(X)} \sum_{j=1}^{N_{BB}} \sum_{k=0}^{F_j} t(B_j, X, k) \quad (4)$$

ここで各記号の意味は次の通りである．
$T(X)$: プロセッサ構成が X のときのアプリケーション・プログラム実行時間.
N_{BB} : アプリケーション・プログラムに含まれるベーシック・ブロックの数.
F_j : ベーシック・ブロック B_j の実行頻度.
$t(B_j, X, k)$: プロセッサ構成 X のときの，k 回目に実行されたときの，ベーシック・ブロック B_j の実行サイクル数. ここで，$t(B_j, X, 0) = 0$. $t(B_j, X, 0)$ は，ベーシック・ブロック B_j が実行されなかったときに用いられる.
$f_{ck}(X)$: プロセッサ構成 X で表されるプロセッサの動作周波数.

ベーシック・ブロックは，プログラムの制御フローによらず，ブロックの最初から最後まで実行されるような一連の命令列である [7]．遅延分岐やマルチサイクルを行う機能ユニット，パイプライン化された機能ユニットを含むパイプライン・プロセッサでは，入力データにより実行サイクル数が変わりうる．これは分岐命令による遅延，パイプライン・ストールの発生て実行サイクル数がデータに依存する機能ユニットなどが，プログラムの静的な制御フローではなく，入力データに依存するためである．以上の理由から，ベーシック・ブロックの実行されたる履歴を，ベーシック・ブロックの実行サイクル数のパラメタとして考慮しなければならない．

3.2 動作周波数

$f_{ck}(x_i)$ を機能ユニット x_i の最大動作周波数とするときのプロセッサ構成 X でのプロセッサの動作周波数 $f_{ck}(X)$ は以下のように求められる．

$$f_{ck}(X) = \frac{\min\limits_{x_i \in X} f_{ck}(x_i)}{1 + \min\limits_{x_i \in X} f_{ck}(x_i) \times T_{trans}} \quad (5)$$

$f_{ck}(x_i)$: 機能ユニット x_i の最大動作周波数.
T_{trans} : 制御部を含めたモジュール間信号伝達遅延時間.

3.3 ハードウェアコスト

論理合成システムを用いて機能ユニットを生成すると，ハードウェア・コストはクロック周波数が高くなるにつれて大きくなる傾向にある．動作周波数によって最適な機能ユニットの面積は変わり得るため，面積を表す関数のパラメタに動作周波数を考慮すべきである．したがって，演算器構成 X でプロセッサを構成した際の，プロセッサ全体のハードウェア・コスト $A(X)$ は，

$$A(X) \triangleq a(kernel, f_{ck}(X)) + \sum_{i=1}^{n} a(x_i, f_{ck}(X)) \quad (6)$$

$A(X)$: プロセッサ構成 X で構成した際のプロセッサの面積
$a(x_i, f)$: 機能ユニット x_i の動作周波数 f での面積

と定義される．

4 提案手法

命令セット最適化問題は，性能最適化問題，ハードウェアコスト最小化問題，消費電力最小化問題の 3 つの種類に分類できるが，本稿では性能最適化問題を対象とする．

4.1 最適化問題の定式化

本稿の性能最適化問題は，制約条件 (8) のもとで，目的関数 (7) の値が最小になるプロセッサ構成 X を求める最適化問題として定式化できる．

$$T(X) = \frac{1}{f_{ck}(X)} \sum_{j=1}^{N_{BB}} \sum_{k=0}^{F_j} t(B_j, X, k) \quad (7)$$

$$A(X) \leq A_{max} \quad (8)$$

A_{max} : ハードウェア・コストの制約条件.

4.2 アルゴリズム

性能最大化問題は組み合わせ問題となる．この問題は，ハードウェア・コストと実行サイクル数などのパラメータを用いて解かれる．ハードウェア・コスト

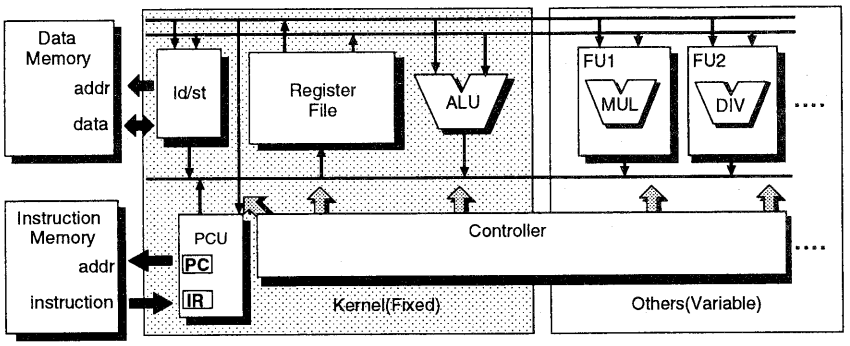

図 1: Structure of CPU core

と実行サイクル数は，プロセッサの構成に依存するが，一般に全てのプロセッサ構成を調べるための全探索は膨大な設計空間の探索となることから，非現実的である．そこで，性能最大化問題を，式 (9) で表される下限関数を用いて分枝限定法で解く．

$$\text{Lower bound} = \frac{1}{f_{ck}(S_d)} \sum_{j=1}^{N_{BB}} \sum_{k=0}^{F_j} t(B_j, X', k) \quad (9)$$

ここで $X' = S_d \cdot S'_d$ である．$\cdot$ は連接を意味する．サフィックス d は探索中の節点の深さを表し，S_d はその節点ですでに決まっているプロセッサの構成を示す d-組である．S'_d は探索中の節点の子孫のうちで，クロック周波数が十分に小さい際の最小クロックサイクル数で実行可能なプロセッサ構成に必要な実装方法の $(n-d+1)$ 組である．

5 評価実験

5.1 目的

提案手法の効果を調べるために評価実験を行った．実験の目的は，

- 入力された C プログラムを実行するのに最適なプロセッサ構成にあたえる，パイプライン・ステージ数の効果
- 提案手法の実行時間

を調べることである．

5.2 ターゲット・アーキテクチャ

実験では，プロセッサモデルとして図 1 にあるモデルを用いた．本モデルの特徴は以下の通りである．

- アーキテクチャ・タイプ
  - スカラ・アーキテクチャ
  - ロード/ストア・アーキテクチャ
  - ハーバード・アーキテクチャ
  - 3,4,5 段のパイプライン・ステージ
    5 段の場合は，IF, ID, EX, MEM, WB [8]，
    4 段の場合は IF, ID, EX/MEM, WB,
    3 段の場合は IF/ID, EX/MEM, WB.
    EX ステージの長さはパイプライン化/マルチサイクル化によって，1 サイクル以上となりうる．$(EX_0, EX_1, \cdots, EX_n)$
- 命令セット
  - DLX (32bit RISC プロセッサ)[8] 命令セットのサブセット
- レジスタファイルの構成
  - 汎用レジスタ方式
  - 32 個の 32 ビットレジスタ
- 機能ユニット
  - Non-pipelined/Pipelined, non-multicycle/multicycle
- メモリモデル
  - オンチップ高速 SRAM

表 1: ターゲット・アーキテクチャで使用される機能ユニットの数

機能	意味	数
ld/st	ld/st Unit	1
ALU	Arithmetic Logic Unit	1
PCU	PC Control Unit	1
MUL	multiplier	0 or 1
DIV	divider	0 or 1

表 3: カーネル部分の性能一覧

加算器のアルゴリズム	段数(名前)	MF (MHz)	面積 (K gates)
ripple carry	3(a)	35.6	37.7 ~ 38.4
	4(c)	51.0	38.7 ~ 39.5
	5(e)	56.3	39.3 ~ 40.1
carry look ahead	3(b)	52.3	37.3 ~ 38.3
	4(d)	61.6	40.5 ~ 42.1
	5(f)	113.9	41.1 ~ 45.7

"MF" : 最大動作周波数

5.3 使用する機能ユニット

DLX 命令サブセットを実装するには，表 1 の 5 種類の機能が必要となる．

- **ld/st** メモリ制御部を制御し，データメモリとのやり取りを行う機能
- **ALU** 算術論理演算を行う機能
- **PCU** プログラム・カウンタを制御し，命令メモリとのやり取りを行う機能
- **MUL** 乗算演算を行う機能
- **DIV** 除算演算を行う機能

本実験においては，ALU, PCU, ld/st Unit, 制御部とレジスタ・ファイルが "*kernel*" となる．機能 MUL と DIV は "*kernel*" で実行されるソフトウェア実装の可能性がある．ソフトウェア実装は，高性能を得るために, ライブラリコールでなくインライン展開でおこなわれる．表 2 は実験で用いる機能ユニットの一覧である．**演算遅延** (R) は演算器にデータが入ってから結果が得られるまでのクロックサイクル数をあらわし，**発行遅延** (I) は命令が発行されてから次の命令が発行されるまでのクロックサイクル数を表す．機能ユニットのハードウェアコストは．動作周波数によって変化する．一般的に，クロック周波数が高ければコストは増え，低ければコストも減る．表 3 は，実験で用いるカーネルの性能一覧である．パイプライン段数が増えるほど面積が増加し，クロック周波数も高くなる．

表 4: 各機能ユニットに発行される命令の割合

機能名	PCU (%)	ld/st (%)	ALU (%)	MUL (%)	DIV (%)
ADPCM	40.8	5.8	53.4	0	0
FFT	19.4	29.8	43.2	7.6	0
PARCOR	18.7	30.4	30.2	17.8	3.0

本稿では，ハードウェアコストと遅延時間は Synopsys 社の論理合成システム Design Compiler と VLSI テクノロジー社の VSC753d (0.5μm CMOS) ライブラリを用いて合成した結果を用いている．

5.4 サンプル・プログラム

アプリケーション・プログラムとそれに対する入力，使用する機能ユニットの性能，最大クロック周波数，面積などの情報をもったデータベースが与えられた際に，プロセッサ上でのアプリケーションの実行サイクル数が計測される．実装方法はモジュール・データベースから利用できる．実験では以下の 3 つのプログラムを用いた．

- ADPCM (Adaptive Differential Pulse Code Modulation)
- 128 点 FFT (Fast Fourier Transform)
- 3-rd order PARCOR (PARtial auto-CORrelation) フィルタ

3 つのプログラムで発行される命令の割合は，表 4 の通りである．アプリケーション・プログラムの特徴として，ADPCM は乗除算命令を発行せず．FFT は除算命令を除く命令を発行し，PARCOR フィルタは乗除算を含め，全ての命令を発行する．本最適化手法では，PARCOR フィルタを対象のアプリケーション・プログラムとした際には，設計空間が一番広くなり，ADCPM の場合は，動作周波数とパイプライン段数に関しての最適化となる．

表 2: 機能ユニットの性能一覧

FU	特徴		R/I	MF (MHz)	面積 (K gates)
MUL	Seq.		32/2	130.0	54.7~65.2
			32/4	148.3	24.8~29.1
			32/8	155.2	12.0~13.6
			32/16	170.6	5.7~ 6.7
			32/32	174.5	2.7~ 3.2
		Booth	16/2	126.5	29.4~33.7
			16/4	112.8	13.8~15.5
			16/8	125.1	6.5~ 8.1
			16/16	156.2	3.0~ 4.6
			17/17	158.7	2.9~ 3.2
	Par.		1/1	22.2	19.9~20.1
			2/1	36.4	11.2~14.6
			4/1	44.1	14.0~15.9
		Booth	1/1	32.5	15.6~23.6
			2/1	40.1	18.6~26.7
		CSA	1/1	93.4	41.6~42.6
			2/1	152.6	46.5~49.0
	software		非定数	113.9	0

FU	特徴		R/I	MF (MHz)	面積 (K gates)
PCU			1/1	113.9	39.3~45.8
ld/st			1/1		
ALU			1/1		
DIV	Seq.		34/33	145.7	4.4~ 5.1
		2bit	18/3	99.1	33.4~35.7
			18/6	113.3	18.0~20.6
			18/9	107.6	12.7~14.3
			18/17	125.1	5.7~ 6.4
	Par.		1/1	8.4	16.6~21.2
	software		非定数	113.9	0

"FU" : 機能ユニット
"R/I" : 演算遅延 / 発行遅延
"MF" : 最大動作周波数
"software" : ソフトウェア実装
"Seq." : 逐次型
"Par." : 並列型

5.5 実験環境

アプリケーション・プログラムの実行時間を計算するためには，プログラムの実行サイクル数が必要となる．実行サイクル数を計算するには，パラメータ N_{BB}, F_j と $t(B_j, X, k)$ が必要となる．対象となるアプリケーション・プログラムをコンパイルする際に，ベーシック・ブロックの数 N_{BB} は計算可能である．このコンパイル時に,gcc-2.7.2.3(GNU C Compiler)[9] を基にしたプロセッサ構成 X に対するパイプライン・スケジューリングが行われる．この際，プロセッサ構成 X に対するベーシック・ブロック B_j の k 回目の実行サイクル数 $t(B_j, X, k)$ が計算される．実験ではメモリモデルとして，メモリアクセスが1サイクル以内に終了するメモリモデルを仮定している．また，使用する機能ユニットの結果遅延と演算遅延は入力に依存しない一定の値となるので，ベーシック・ブロックの正確な実行サイクル数が計算できる．ソフトウェアによる実装が選択された場合でも，コンパイル時にインライン展開されるので，結果遅延と演算遅延が入力に依存することはない．今回使用するターゲット・アーキテクチャでは，$t(B_j, X, k)$ は0を除くどのような k に対しても同じ値となる．対象となるアプリケーション・プログラムをコンパイルした後，そのプログラムは動的プロファイルを得るために実行される．シミュレータ上でプログラムを実行することにより，動的プロファイルのデータ F_j が得られる．

6 実験結果と考察

6.1 トレードオフ

ハードウェア・コスト制約に対する，最小実行時間を検討する．図2，図3，図4はそれぞれ対象アプリケーションがADPCM，FFT，PARCORフィルタの際のハードウェア・コスト制約と最小実行時間の関係をあらわしている．横軸がHWコスト制約を示し，縦軸に最小実行時間を示す．グラフ中に，パイプライン・ステージ数が3段,4段,5段の場合のプロセッサ構成での結果を示した．

初めに，図2のADPCMの結果に対して考察する．ADPCMでは，ハードウェア・コスト制約が厳しい場合，パイプライン・ステージ数が3段のプロセッサ構成が選択され，制約が緩くなった時点で，5段のプロセッサ構成が選択される．すなわち，ハードウェア・コスト制約が厳しい場合には，5段のパイプランでプロセッサを構成できず，ハードウェア・コストが十分である場合には，高速な5段パイプライン構成が選択されるようになる．ADPCMでの実験では，4段パイプライン構成が，いずれのハードウェア・コスト制約下でも選択されていない．そこで，各パイプライン段数での，面積，実行時間，周波数とク

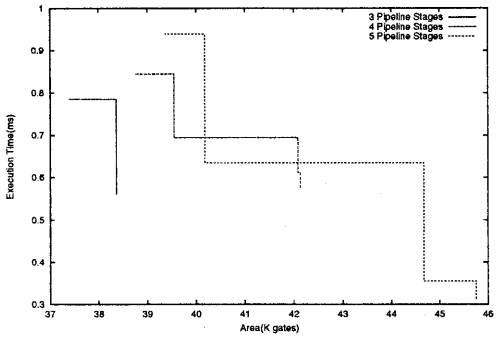

図 2: ハードウェア・コスト制約と実行時間の関係 (ADPCM)

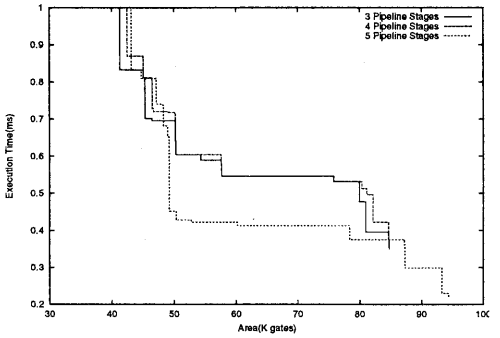

図 3: ハードウェア・コストと実行時間の関係 (FFT)

表 5: 各段数でのトレードオフ (ADPCM)

可否	面積 (K gates)	実行時間 (ms)	段数 (名前)	周波数 (MHz)	サイクル数
○	37.3	0.78	3 段 (b)	37.3	29343
○	38.3	0.56		52.3	
×	38.7	0.84	4 段 (c)	42.0	35502
×	42.1	0.57	4 段 (d)	61.6	
○	44.6	0.35	5 段 (f)	100.7	35758
○	45.7	0.31		113.9	

ロックサイクル数のトレードオフについて考察する. ADPCM を対象とした際の, 各パイプライン段数に対する面積, 実行時間, 周波数, クロックサイクル数を表 5 に示す. 表中の可否は, トレードオフ条件を満たすかを示している. 4 段構成の場合は, 3 段パイプライン構成よりも動作周波数は高いものの, クロック・サイクル数が少ないため, クロック周期とクロックサイクル数の積である実行時間が, 結果として同じ面積上では小さくなり, 3 段構成が選択される.

次に図 3 の FFT の結果に対して考察する. このアプリケーションでは, 乗算機能も最適化の対象となるので, ADPCM よりも設計空間が広い. 図 3 からわかるように, ADPCM の場合よりも, パイプライン・ステージ数がアプリケーション実行時間に対して影響をあたえる. ADPCM の場合と同様に, ハードウェア・コスト制約が緩い場合には, 3 段パイプライン構成しか選択されない. 4 段パイプライン構成が可能となる 4 万ゲート程度のハードウェア・コスト制約では, ADPCM の場合と同様に 4 段パイプライン構成は最適とならない. 制約条件が 4 万ゲート近辺では, 5 段構成が最適となる範囲も存在する. ハードウェア・コスト制約が 5 万ゲート弱になると, 5 段パイプライン構成が最適となる. ハードウェア・コスト制約が 8.5 万ゲートまではこの 5 段構成が最適である. 制約条件が 8.5 万ゲートを越えた時点で, 3 段パイプライン構成が最適となる. これは, ハードウェア・コストが比較的大きい, 高性能な演算器の動作周波数が, 3 段構成の最大動作周波数より高く, 4 段構成の最大動作周波数よりも低いために, 4 段構成が最適となったためである. ADPCM が対象の際には, 乗算機能がソフトウェア実装されるので, このような現象は起きなかった. この時点では, 5 段パイプライン構成と高性能な演算器は, ハードウェア・コスト制約のため同時に使用できない. ハードウェア・コスト制約が同時に使用できる程度になると, 再び 5 段パイプライン構成が最適なものとして選択される. この例から, ハードウェア・コストと動作周波数ごとに最適なパイプライン構成が存在することがわかる.

最後に, 図 4 の PARCOR フィルタの結果に対して考察する. このアプリケーションでは, 乗算機能および除算機能も最適化の対象となるので, ADPCM, FFT よりも設計空間が広大である. 設計空間が広大になった結果, 図 4 からわかるように, ADPCM, FFT の場合よりも, パイプライン・ステージ数がアプリケーション実行時間に対して大きな影響をあたえるようになった.

最適化問題を解くプログラムは, 分枝限定法に基づいて実装した. 表 6 に, 各制約条件の下で PARCOR

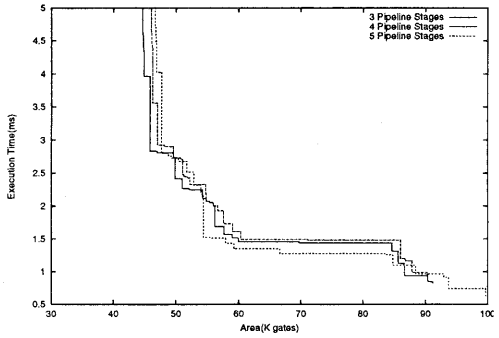

図 4: ハードウェア・コストと実行時間の関係 (PARCOR フィルタ)

表 6: PARCOR フィルタプログラムでの探索葉数

制約 (K gates)	探索節点数	実行時間 (分:秒)
40	18	3:22
50	317	4:03
60	636	4:37
70	1002	4:20
80	1446	4:22
90	1066	4:25
100	516	3:16
全探索	25872	14:14

フィルタ・プログラムの最適化の際に探索した節点の個数と時間を示す. 実験は Intel Pentium II (300MHz) 上で行なった. 全探索を行なった場合には制約によらず, 表 2 の演算器構成を使用する場合は,25872 個の節点の探索が必要であるが, 分枝限定法を用いることによって, 節点の探索数を少なく押さえることが出来ていることがわかる. しかしながら, 表 2 にある 18 点探索に必要な時間からわかるように, 現状では, 動的プロファイルを得るための時間が支配的である.

7 おわりに

動作周波数とパイプライン段数を考慮した設計最適化手法を提案した.

実験によってプロセッサのパイプライン・ステージ数, 動作周波数, プロセッサが使用するハードウェアコストと, プログラムの実行時間の間にはトレード・オフが存在することが知られた.

ハードウェア・ソフトウェア・コデザイン・システムとして性能を追求し, 本手法のように, ステージ数と動作周波数を最適化のパラメタとして用いると, 従来よりもより高性能なプロセッサが設計できることが示された.

今後の課題としては, 更に探索空間を広げるために, パラメタとして与えられるパイプライン段数を増やすことが考えられる. 最適化手法の時間を実験で示したが, 動的プロファイルを得るための時間が支配的なので, 動的プロファイルを得る時間の短縮も考えられる.

謝辞

本研究を進めるにあたり, 貴重なコメントを頂いた (株) 半導体理工学研究センターの小澤時典博士, (株) 日立製作所の堀田正生博士, 松下電器産業 (株) の村岡道明氏, および大阪大学 VLSI システム設計研究室の諸兄に深謝する. なお, 本研究の一部は (株) 半導体理工学研究センターとの共同研究による.

参考文献

[1] A. Alomary, T. Nakata, Y. Honma, M. Imai and N. Hikichi, "An ASIP Instruction Set Optimization Algorithm with Functional Module Sharing Constraint," Proc. of ICCAD-93, pp.526-532, 1993.

[2] N. N. Binh, M. Imai, A. Shiomi, N. Hikichi, "A Hardware/Software Partitioning Algorithm for Pipelined Instruction Set Proce ssor," Proc. of EURO-DAC '95, pp. 176-181, Brighton, UK, Sept. 1995.

[3] 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 EURO-DAC'96, pp.126-131, 1996.

[4] 大槻典正, 武内良典, 今井正治, 浜口清治, 柏原敏伸, 引地信之, "VLIW プロセッサ自動生成における演算器構成最適化の一手法," 信学技報 VLD97-92, pp.101-108, 1997.

[5] Katsuya Shinohara, Norimasa Ohtsuki, Yoshinori Takeuchi and Masaharu Imai, "A Performance Optimization Method for Pipelined ASIPs in Consideration of Clock Frequency," IEICE Trans., VLSI Design and Cad Algorithms, (to appear).

[6] H-P. Jaun, D. D. Gajski and S. Bakshi, "Clock Optimization for High-Performance Pipelined Design," Proc. EURO-DAC'96, pp330-335, 1996.

[7] Alfred V. Aho, and Jeffrey D. Ullman, *Principles of Compiler Design*, Addison-Wesley Pub., 1977

[8] David A. Patterson, John L. Hennessy, *Computer Architecture A Quantitative Approach Second Edition*, MORGAN KAUFMANN PUBLISHERS, INC., 1996.

[9] R. M. Stallman, *Using Porting GNU CC*, Free Software Foundation, Inc., 1995.