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

2003－ＳＬＤＭ－112 (16)
2003／11／28

非同期データパス合成における解探索空間の削減

川鍋 昌紀† 齋藤 寛†† 今井 雅†† 中村 宏†† 南谷 崇††

† 東京大学大学院情報理工学系研究科システム情報学専攻
†† 東京大学先端科学技術研究センター 〒153-8904 東京都目黒区駒場 4-6-1
E-mail: †{kawanabe,hiroshi,miyabi,nakamura,nanya}@hal.rcast.u-tokyo.ac.jp

あらまし 非同期データパス合成ツール Mercury は，与えられた Data Flow Graph (DFG)，リソースライブラリ，設計制約のもと，面積/性能が最適となるデータパス回路の集合を探索する．しかし，Mercury で行われている分岐限定法を用いた探索は，DFG のノード数 n に対して $O(3^{n(n-1)/2})$ の探索空間を要するため，規模の大きな DFG を合成できないといった問題点がある．そこで本稿では，合成されるデータパス回路の質をなるべくおとさずに，探索空間を効率よく削減する新しいフィルタを提案する．提案されたフィルタの利用によって，規模の大きな DFG も効率よく合成することが可能となる．実験として，提案されたフィルタを 4 つの DFG に適用し，その有効性を確かめた．
キーワード 非同期回路，DFG，データパス合成，分岐限定法，フィルタ

Design Space Reduction Filter in Asynchronous Data-path Synthesis

Masaki KAWANABE†, Hiroshi SAITO††, Masasi IMAI††, Hiroshi NAKAMURA††, and Takashi NANYA††

† Department of Information Physics and Computing, Graduate School of Information Science and Technology, the University of Tokyo
†† Faculty of Research Center for Advanced Science and Technology, the University of Tokyo Komaba 4-6-1, Meguro-ku, Tokyo, 153-8904
E-mail: †{kawanabe,hiroshi,miyabi,nakamura,nanya}@hal.rcast.u-tokyo.ac.jp

Abstract Asynchronous data-path synthesis tool Mercury explores a set of area/performance optimum data-path circuits from a data flow graph(DFG), a resource library, and design constraints. However, because the design exploration of Mercury based on branch-and-bound algorithm requires the design space of $O(3^{n(n-1)/2})$, it cannot synthesize data-path circuits from large DFGs. Therefore, in this paper, we propose a new filter to reduce the design space while suppressing the effect for the quality of data-path circuits. Because the design space explored by Mercury is efficiently reduced, it can synthesize data-path circuits from large DFGs. By applying our proposed filter to four DFGs, we confirmed the efficiency of our proposed filter.
Key words asynchronous circuit, DFG, data-path synthesis, branch-and-bound algorithm, filter

1. はじめに

クロックを用いず，事象駆動原理にもとづいて動作する非同期回路は，低消費電力，高信頼性，平均遅延動作，高いモジュール性など優れた性質があることが知られているが，実際の設計には広く浸透していない．その理由として，非同期回路設計を支援するツールが不足していることがあげられる．

そこで本稿では，非同期回路を合成する図 1 のような CAD 環境の構築の一環として，既存の非同期データパス合成ツール Mercury [1], [2] の解探索空間を削減するフィルタを提案する．

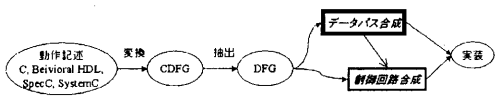

図 1 A flow of asynchronous circuit synthesis

Mercury は入力としてハードウェア合成やコンパイラの中間表現として広く用いられている Data Flow Graph (DFG) [3](図 2)，リソースライブラリ，設計制約を受け取り，面積／性能が最適となる非同期データパス回路の集合を合成し出力する．DFG

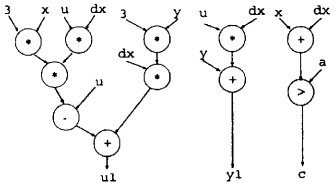

図 2 An example of DFG (differential equation solver)

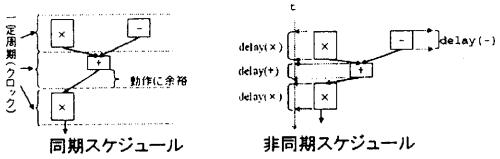

図 3 Difference of scheduling between synchronous and asynchronous circuit design

は同期／非同期，ハードウェア／ソフトウェアによらない中間表現であるため，ユーザは仕様の記述言語として，C，SpecC [4]，SystemC [5]，VHDL，VerilogHDL など任意の言語を選ぶことができる．

与えられた設計制約に対し，面積／性能が最適となるデータパス回路の集合を探索するために，Mercury は DFG 上で同じ処理を行う 2 つのノード間にリソースエッジと呼ばれるエッジを付加することによって，ノード間のスケジュールとリソースの共有を行う．どのようにリソースエッジを引くか，その組み合わせを分岐限定法 (branch-and-bound) を用いて探索し，解 (データパス回路構成) 候補を得る．しかしリソースの付加に対して，

（1） リソースエッジをつけない

（2） リソースエッジをつける

（3） (2) と向きが逆のリソースエッジをつける

の三通りの場合分けを行って探索を行うため，与えられた仕様の大きさ（DFG のノード数）に対して，探索空間の大きさが $O(3^{n(n-1)/2})$ となってしまう．このため，現実的な時間内に規模の大きな仕様（DFG）からデータパス回路を合成できなくなる．この問題を解決するためには，データパス回路の質をなるべく落とさずに，かつ効率よく探索空間を削減することが必須となる．

本稿では解の質をなるべく落とすことなく探索空間を効率よく削減する新規フィルタを提案する．提案された新規フィルタは，データパス回路合成の最中に得られた各ノードの実行開始時間をもとに，開始時間の早い方から開始時間の遅い方へのみリソースエッジを付加することによって探索空間の削減を行う．結果として，実験に示される通り，規模の大きな DFG を効率よく合成することが可能となる．

本稿の構成は以下のとおりである．Mercury の概要を 2 章で説明し，3 章で提案フィルタについて述べる．4 章で提案フィルタを用いた実験と結果の考察を行う．5 章でまとめを述べる．

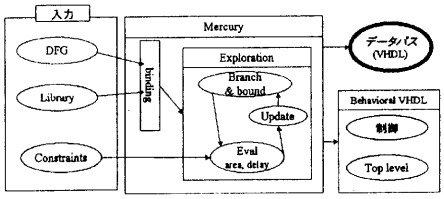

図 4 Synthesis flow in Mercury

2. Mercury によるデータパス合成

2.1 Data Flow Graph

Data Flow Graph は有向グラフで，$G\langle V, E\rangle$ として表される．ここで，頂点の集合 $V(V = \{v_1 \dots v_n\})$ は，データパス回路で実行される演算 $(o_1 \dots o_n)$ を表し，枝の集合 E は，演算間のデータ依存関係を表す．

2.2 データパス合成

データパス合成とは，DFG 等の動作記述を実行するために使用されるリソースの種類や数，スケジュールを決定し，データパス回路構成を決定する処理である．処理は以下の三つの要素からなる．

バインディング 与えられた仕様内の演算に対してどの種類のリソースを使用するかを決定

スケジューリング 各演算処理の開始・終了時間の決定

アロケーション 演算や変数へのリソースへの割り当て

これら 3 つの処理は互いに影響を与え合うので，最適なデータパス回路構成の発見は非常に困難である．そのため，設計者によって与えられた仕様（DFG，ライブラリ，制約）に対して，合成ツールはその仕様を満たす範囲内で，面積，速度，消費電力等の異なる複数のデータパス回路を出力することが望まれる．Mercury は与えられた仕様に対して面積と速度を指標として解候補を絞り込み，複数のデータパス回路を出力する．実際の設計にどの解を利用するかは設計者が決定する．

2.3 非同期データパス合成

非同期回路は同期回路とは異なり，クロック周期によるステップの切り分けがないため，各演算器は必要なデータがそろい次第処理を開始する．そのため，非同期回路のデータパスのスケジューリングはステップ（もしくはサイクル）ではなく，連続値で行われる．リソースライブラリの各リソースに対して，最大，最小，平均遅延をパラメータとして与えておくことによって，Mercury はこれらのパラメータを用いながらデータパス回路を合成する．同期回路では処理時間が短い演算も次のクロックを待ってデータを渡すが，非同期回路では処理が終わるとすぐ次の演算を開始することができるため (図 3)，各演算の処理時間にバラつきがある場合，非同期回路の方が性能の面で有利である．

2.4 Mercury の概要

Mercury は DFG，リソースライブラリ，設計制約（面積，速度，消費電力等）を入力データとして受け取る．データフォーマットは独自形式のテキストファイルである．出力は VHDL

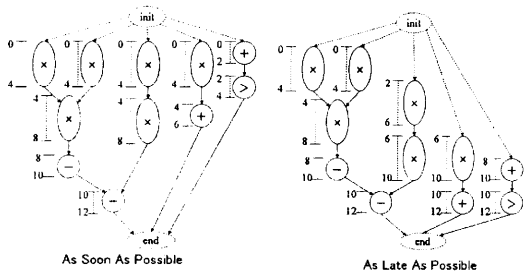

図 5　ASAP ALAP scheduling

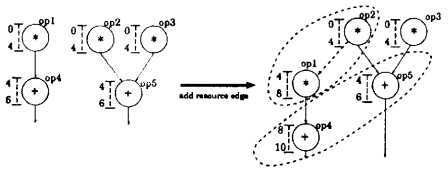

図 6　Generation of a new data-path circuit by adding a resource edge

```
Asynchronous-Left-Edge (list of operations I) {
  Sort I in ascending order of start time.
  instance = 1;
  foreach operation l in I {
    l_instance = instance;
    t = l;
    foreach operation k in I after l {
      if (k_min_start ≧ t_max_stop or
          there exists a path between t and k) {
        k_instance = instance;
        t = k;
        remove k from I;
      }
    }
    instance + +;
    remove l from I;
  }
}
```

図 7　Asynchronous Left-edge algorithm

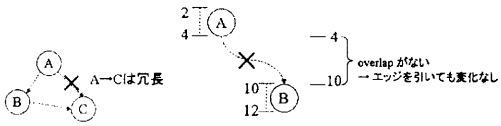

図 8　Remove Redundant, Avoid Implied Edge

で記述された非同期データパス回路の集合と，シミュレーションを行うための VHDL によるシミュレーションモデルである．

2.4.1　バインディング

DFG 内の演算に対し，ライブラリ中の対応するリソースを割り当てる．

2.4.2　スケジューリング

まず，ASAP，ALAP スケジューリング [3] を行う．ASAP はできるだけ各演算を早く開始させ，ALAP は逆に与えられた時間内に処理を終了させる制約の中で演算の開始をできるだけ遅くするスケジューリングである．図 5 の例では乗算の遅延を 4 単位時間，加算，減算，比較演算を 2 単位時間としている．非同期回路ではクロックがないため，同期回路の ALAP，ASAP と異なり，クロックステップに縛られない．ASAP で得られた最小処理時間を制約時間として ALAP を行うことで，DFG のクリティカルパスと，クリティカルパス上にない演算の自由度（クリティカルパスを伸ばさずに演算の開始を移動させられる範囲）が分かる．

2.4.3　リソースエッジによる解候補の生成と評価

Mercury は DFG にリソースエッジと呼ばれる演算の順序付けを表すエッジを付加して，新たな ASAP，ALAP スケジューリングを作る．リソースのアロケーションはスケジューリングによって一意に決まるため，リソースエッジの付加によって新しい解候補が生成される．

図 6 にリソースエッジの付加による新しい解の生成を示す．図の左側がエッジを加える前の DFG とそれに対するスケジューリングとアロケーション，右側がノード op2 から op1 へリソースエッジを引いた DFG とそれに対するスケジューリングとアロケーションである．左側の DFG では，ノード op1，op2，op3 が同時に乗算を行うため，各ノードに対し別個の乗算器が割り当てられる．右側の DFG ではノード op2 から op1 へリソースエッジが引かれているため，op1 は op2 の演算が終了してから演算を開始する．そのため，同時に処理が行われる乗算は op2 と op3 のみとなり，op1 は op2 に割り当てられた乗算器を共有することができる．また，op1 の開始時間が遅れたため，op4 の開始時間も遅れ，op4 と op5 とで ALU を共有することができる．

解候補の探索は分岐限定法によって行われる．リソースエッジの付加によって作られた解の質（面積，速度）を既存の解と比較し，良質な解が得られたなら解候補の集合に加え，さもなければ破棄し，また別のノード間で新たなリソースエッジの付加と解の評価を繰り返す．解候補の集合に含まれる解は，解候補が加えられるたびにその新たな解と比較され，もし面積，速度ともに新たな解より劣っている場合は解候補から削除される．

2.4.4　アロケーション

Mercury はスケジューリングが決定されると Asynchronous Left-Edge アルゴリズム [2](図 7) によって各ノードへのリソースの割り当てを行う．

非同期回路のアロケーションでは，割り当てはステップではなく各演算の開始・完了時間によって決定される．また，演算の完了を検出して次の演算を開始させることができるため，演算間にデータ依存がある場合はスケジューリングを見ることなくリソースを共有できる．ここで注意すべき点は，アロケーションはスケジューリングを変更しないということ，およびあるノード間にリソースエッジが引かれていても必ずしもリソースの共有を行うわけではないということである．

2.5 フィルタによる探索空間の削減

Mercury は，同じ種類のリソースを用いるノード A，B のペアに対してリソースエッジを引くとき，

（1） A から B へリソースエッジを引く

（2） B から A へリソースエッジを引く

（3） A，B 間にはリソースエッジを引かない

という 3 つの場合分けを行う．この場合分けをすべてのノードに対して行いデータパス回路候補の探索を行うため，探索空間の大きさは最悪で $3^{n(n-1)/2}$（n はノードの数）となる．そのため，データパス回路候補を効率的に探し出すために探索空間の削減が必須となる．Mercury は以下に述べるフィルタによって探索空間の削減を行う．どのフィルタを使用するかは合成前にユーザがオプションとして指定する．なお，使用するフィルタによっては解の質を落とすので，フィルタの選択は注意が必要である．

2.5.1 Remove Redundant

図 8 左のように，ノード A から B，B から C へというようにリソースエッジが A から C へ推移関係を成すとき，A から C へのリソースエッジは冗長になる．Remove Redundant フィルタはそうした冗長なエッジの付加を回避する．

2.5.2 Avoid Implied Edge

ある演算ノードの開始・終了時間が同じ種類の別のノードの開始・終了時間とまったく重ならない場合，リソースエッジを引くことなくそれら二つのノードを共有することができる．Avoid Implied Edge フィルタはそうした可能性を検出し，リソースエッジを引いた場合に発生する探索をカットするフィルタである (図 8 右)．

2.5.3 Minimal Latency

Minimal Latency フィルタは，回路全体の処理時間が延びるリソースエッジ，つまりクリティカルパスに影響を与えるエッジの付加を回避する．このフィルタによって探索空間を大幅に削減することができるが，当然のことながら速度最優先の解しか得られず，解候補の性質の幅が狭まってしまう．

2.5.4 Prune Equal Result

あるリソースエッジの付加が，DFG のどのノードのスケジューリングにも影響を与えない場合，Prune Equal Result フィルタはそのようなエッジの付加を回避する．しかし，エッジを引いたその時点ではあるノード A のスケジューリングが変化しなかったとしても，その後の探索で他のノードから新規エッジが加えられたときにノード A のスケジューリングが変化することがあるため，このフィルタを適用すると最適解が得られる保障はなくなる．

2.5.5 Maximally Shared

解探索中に，ある種類の演算器が一つしか使われないことが分かっている場合，既にリソース共有が最大限に行われているため，そのリソースを使うノード間にさらにエッジを加えても出力に変化は無い．Maximally Shared フィルタはそうした場合に無駄なリソースエッジが引かれることを防止する．

2.5.6 Hierarchical Exploration

Hierarchical Exploration（階層化探索）はフィルタではないが，合成時間の短縮の効果は今まで述べたどのフィルタよりも大きく，重要な手法である．

階層化探索ではまず，同じリソースを用いるノードをユーザが指定した大きさのブロックに分割する．次に各ブロックに対してブロック探索を行い，ブロック探索で解候補を作り出したリソースエッジ（クリティカルエッジ）の集合をマージする．DFG 全体の探索は，マージされたクリティカルエッジの集合に含まれるエッジのみを使って行う．ブロック探索とは，ブロック内にあるノード間でのみリソースエッジの付加を行い，ブロックに含まれないノード間にはリソースエッジを付加しない探索である．

階層化探索は探索時間を大幅に短縮するが，DFG 全体での最適解は保障されなくなる．例えば，ある最適解を作り出すリソースエッジ A が，階層化探索においてはブロック探索においてクリティカルエッジにならない場合，リソースエッジ A は DFG 全体の探索に用いられなくなるため，最適解は得られなくなる．

現状の Mercury を使って，ある程度以上の大きさの DFG を現実的な時間内に合成させるためには，ブロックの上限を小さく指定した階層化探索を行わなければならない．

3. 新規フィルタ

本稿で提案する新規フィルタ one direction は，合成処理の途中で得られる各ノードの実行開始時間を用いて，開始時間の早いノードから開始時間の遅いノードへの方向にのみリソースエッジを引く制約を課すことにより探索空間を削減する．

先にも述べたように，Mercury は 2 つのノードに対してリソースエッジを引く際に，3 通りの場合分けを行うので，探索空間の大きさは $3^{n(n-1)/2}$ となる．しかし，one direction を利用することによって，理想的な場合（すなわち全てのノードの実行開始時間は早い，遅いの関係にある）リソースエッジを引く際の場合分けは，一定の方向に引く，あるいは引かないの 2 通りになるため，Mercury における解空間の大きさを $2^{n(n-1)/2}$ まで削減することに匹敵する．

3.1 One Direction の概要

Mercury は探索中に ASAP，ALAP スケジューリングを行っているので，リソースエッジを加える前のノード A，B の開始時間，終了時間はあらかじめわかっている．また，回路の速度をなるべく落とさずにリソースの共有を行うためには，開始時間が早いノードから開始時間が遅いノードへとエッジを引いた方が良いことは明らかである．そこで one direction では，ノード A，B の ALAP スケジューリングにおける開始時間を比べ，開始時間の早い方から開始時間の遅い方へのみエッジを引くことによって，探索空間を削減する．なお，A，B の開始時間が等しい場合は，従来どおりの 3 通りの探索を行う．

ただし，ある時点ではノード A の開始時間の方がノード B より遅かった場合でも，のちのリソースエッジの加え方によってはノード B の方が遅くなることがあり得るため，このフィルタを適用した場合，最適解が得られる保証はなくなる．

例．図 9 に one direction フィルタの適用例を示す．ALAP

表 1 Experimental results: Differential Equation Solver, 4th-order LMS Filter

Differential Equation Solver

one direction フィルタ使用				既存フィルタのみ			
filter	探索数	時間 (s)	解	filter	探索数	時間 (s)	解
ml pe	124	0	2	ml pe	198	0	6
ml	4095	1	3	ml	11202	2	9
pe oo	5067	1	14	pe oo	41659	11	42
pe	6897	2	14	pe	59928	16	42
oo	96512	27	17	oo	861936	229	67
none	108591	30	17	none	1061052	279	67

4th-order LMS

one direction フィルタ使用				既存フィルタのみ			
filter	探索数	時間 (s)	解	filter	探索数	時間 (s)	解
ml pe	233	1	13	ml pe	275	1	13
ml	441	0	13	ml	525	0	13
pe oo	176662	86	397	pe oo	1866366	969	853
pe	176662	86	397	pe	1866366	965	853
oo	2728052	1291	341	oo	35143690	18564	1645
none	2728052	1290	341	none	35143690	18751	1645

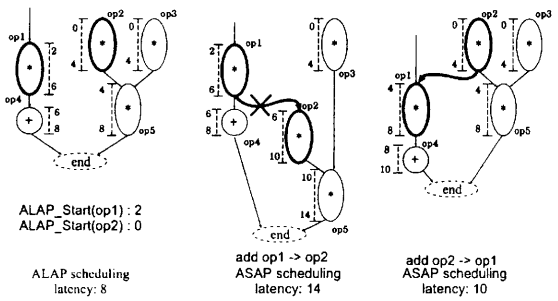

図 9 Example of one direction

スケジューリングを行ったあとの左端の DFG から，op1 の実行開始時間 (2) は，op2 の実行開始時間 (6) より遅いということがわかる．したがって，one direction を適用した場合，op2 から op1 にのみリソースエッジが引かれる．op1 から op2 に対してのリソースエッジは引かれない．

4. 実　　験

新規フィルタ one direction の効果を検証するため，以下に示す条件で 4 つの DFG からデータパス回路を合成した．合成に要する時間，出力された回路の性能を調べ，新規フィルタを用いない場合と比較した．

4.1 条　　件

以下の 4 つに対する DFG を合成の対象とした．

（1） Differential Equation Solver（DIFFEQ）:乗算 6 個，加減算 4 個，比較演算 1 個

（2） 4th-order LMS Filter (LMS):乗算 9 個，加算 8 個

（3） AR-Lattice Filter: 乗算 16 個，加算 12 個

（4） 逆離散コサイン変換（IDCT）：乗算 16 個，加算 30 個

ライブラリデータには Mercury 付属のものを用いた．設計制約は特に与えていない．プログラムは C++で記述し，コンパイラは gcc ver. 3.3.1 を使用した．FreeBSD，Linux，Cygwin (Windows 2000 上)，および SunOS 5.8 上で動作を確認した．

Remove Redundant，Avoid Implied Edge の二つのフィルタは，すべての実験で適用されている．また，現実的な時間内に合成を終了させるために DIFFEQ 以外には階層化探索もあわせて適用した．ブロックの大きさは 4th-order LMS filter は 9，AR-Lattice filter は 7，IDCT は 3 とした．階層化探索のブロックの大きさは，処理を現実的な時間内に終わらせることができる範囲内でできるだけ大きな値を選んだ．

表 1，2 が合成に要する探索空間と合成時間の実験結果である．表の左側 4 列が one direction フィルタを用いた場合，右側 4 列が用いない場合である．項目は左から順に，one direction フィルタとともに用いられた既存のフィルタ，リソースエッジの付加によって作られた解候補群の数，合成に要した時間，最終的に出力された解の個数となっている．新規フィルタを利用しない残り 4 列も同様に，使用フィルタ，リソースエッジの付加によって作られた候補群の数，合成に要した時間，出力された解の個数である．表中の N/A は，その条件下では合成に時間がかかりすぎるために数値が得られなかった項目を表す．

フィルタの欄の略語の対応は

od 新規フィルタ（only One Direction）

rr Remove Redundant

ai Avoid Implied Edge

ml Minimal Latency

pe Prune Equal Result

oo Maximally Shared (prune on Only One)

となっている．

諸事情から，実験環境（CPU，メモリ容量等）の統一が完全にはできなかった．同一合成対象では環境は同一だが，合成対象が違う場合は環境が異なっている場合があるため，合成時間は参考値としていただきたい．

表 3，4 は one direction フィルタを使用した場合とそうでない場合の解の質を比較するために作成した．解は複数個出力されるが，そのすべてが異なる面積，速度を持っているわけではなく，構成は異なるが面積と速度が同等な解がいくつも存在する．そのため，解の面積・速度が違うものだけを取った．合成に用いたフィルタが行，合成された回路の速度（レイテンシ）が列のインデックスで，各欄は対応する行のフィルタによって得られた解の中で，対応する列の速度を持ったものの面積である．該当する速度を持った解が存在しなかった場合は空欄としている．

4.2 考　　察

提案フィルタを用いることによって合成対象のすべてにおいて探索空間を削減することができた．

Minimal Latency と提案フィルタはともに遅延の増加を回避

表 2 Experimental results: AR-Lattice Filter, IDCT

AR-Lattice Filter

one direction フィルタ使用				既存フィルタのみ			
filter	探索数	時間 (s)	解	filter	探索数	時間 (s)	解
ml pe	5896	5	3	ml pe	12258	10	84
ml	101134	84	3	ml	172033	151	64
pe oo	667722	544	33	pe oo	12743196	15513	2116
pe	667722	543	33	pe	12743196	15593	2116
oo	37219338	32367	33	oo	N/A	N/A	N/A
none	37219338	41415	33	none	N/A	N/A	N/A

IDCT

one direction フィルタ使用				既存フィルタのみ			
filter	探索数	時間 (s)	解	filter	探索数	時間 (s)	解
ml pe	4476	7	1	ml pe	4515	8	1
pe oo	47342	84	5	pe oo	1976239	3727	151
pe	47342	84	5	pe	1976239	5601	151
ml	873534	1453	1	ml	1773579	2962	1
oo	21151758	35976	8	oo	N/A	N/A	N/A

表 3 Circuit area comparisons: Differential Equation Solver, 4th-order LMS

Differential Equation Solver

	latency				
Filter	1560	1740	1869	2040	3180
none	1144457	934657		832857	623057
od	1144457	934657		832857	623057
pe	1144457	934657		832857	623057
od pe	1144457	934657		832857	623057
ml	1144457		1042657		

4th-order LMS

	latency						
Filter	2640	2940	3600	3900	4020	4380	5520
none	1783151	1579551	1159951		1058151		848351
od	1783151	1579551	1159951		1058151		848351
pe	1783151	1579551	1261751	1159951		1058151	848351
od pe	1783151	1579551	1261751	1159951		1058151	848351
ml	1783151	1579551					

表 4 Circuit area comparisons: AR-Lattice, IDCT

AR-Lattice

	latency		
Filter	2940	3420	3900
pe	2202608	2100808	1891008
od pe	2202608		2100808
ml	2202608		

IDCT

	latency			
Filter	1980	2340	2640	2460
pe	5711354	3931154	3829354	3829354
od pe	5711354	3931154	3829354	3829354
ml	5711354			

するフィルタであるため，これらを組み合わせた場合の削減率はもっとも良い場合で 1/3 程度となっている．提案フィルタと Maximally Shared の併用はほとんどの場合探索数を削減できず，処理時間が増加する場合すらあるが，Maximally Shared は提案フィルタ以外のフィルタと併用した場合でも，ほとんどの場合において探索数を削減できない．Maximally Shared の削減対象はある種類のリソースの数が一つになっている状態からの分岐であり，削減できる探索空間がもともと小さい上に，階層化探索によってリソースの数が一つとなる解候補を含む探索空間が既に除去されているためと思われる．提案フィルタは，単体での削減量はそれほど高くはないが Maximally Shared より強力で，また他のフィルタと組み合わせても効果のあるフィルタであると言える．

表 3，4 は各フィルタを利用した際の解の質を表しているが，提案フィルタを用いた場合の解の質の低下は AR-Lattice のみで起こり，他の場合では質の低下は見られなかった．また，提案フィルタは速度優先から面積優先まで解候補の幅を残している．Minimal Latency は強力なフィルタだが，もっとも速度の速い解しか得られない．結論として，提案フィルタは解の幅を狭めることなく他のフィルタとの併用でさらに探索空間を削減でき，Mercury の合成時間の短縮に有効であると言える．

5. おわりに

本稿では，分岐限定法を用いた非同期回路のデータパス合成ツール Mercury に対し，新たな解空間削減フィルタを提案した．実験から，他のフィルタと併用した場合においても得られる解候補の幅を狭めず，また解の質も落とさずに探索空間と合成時間を効果的に削減することができた．

現在，合成時間の更なる短縮のために，解空間の分割探索手法の改良や，探索順の変更を試行・実装中である．

文　献

[1] Brandon M. Bachman, "Architectural-level Synthesis of Asynchronous Systems," Master's thesis, University of Utah, 1998.

[2] Brandon M. Bachman, Hao Zheng, and Chris Myers, "Architectural Synthesis of Timed Asynchronous Systems," In Proc. International Conf. Computer Design (ICCD), pages 354-363, 1999

[3] Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu, and Steve Y-L Lin, "HIGH-LEVEL SYNTHESIS Introduction to Chip and System Design", Kluwer Academic Publishers, 1992.

[4] http://www.specc.org/

[5] http://www.systemc.org/