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

2008－SLDM－133（16）
2008 / 1 / 17

FPGA向け動作合成におけるメモリバインディングとスケジューリングアルゴリズムについて

佐川 由己† 貞方 毅† 松永 裕介††

† 九州大学 大学院 システム情報科学府 情報工学専攻
†† 九州大学 大学院 システム情報科学研究院 情報工学部門

あらまし FPGA(Field Programmable Gate Array) ではメモリのサイズや数、ポート数が決まっている。そのためFPGA 向け動作合成では複数の配列を同じメモリへバインディングしなければいけない場合がある。同じメモリへバインディングされた複数の配列に対する配列アクセス (参照や書き込み) は、メモリのポート数を超えて同じステップにスケジューリングする事ができない。そのためメモリバインディング次第で配列アクセスの並列度が変化する。配列アクセスが頻繁に起こるアプリケーションの合成では、演算の並列度が高くても配列アクセスの並列度が低いと最大ステップ数が多くなってしまう。本論文ではメモリサイズ、メモリ数、メモリのポート数制約下で配列アクセスの並列度を高めるようにメモリバインディングとスケジューリングを行なうヒューリスティックスを提案する。目的は各DFG(Data Flow Graph) の最大ステップ数の総和を最小化する事である。既存手法として、Simulated Annealing(以下、SA) を用いた手法が提案されている。SA を用いた手法の問題点として解を出すまでに時間がかかる事が挙げられる。提案手法と SA を用いた手法を比較した実験では、多くの場合ほぼ同じステップ数となる解を見つける事ができている。総ステップ数の悪化は最悪な場合で 20 %程度である。また最高で約 2000 倍、平均で約 1500 倍、高速に解を出す事ができている。

キーワード 動作合成 メモリバインディング スケジューリング FPGA

Memory Binding and Scheduling in High Level Synthesis for FPGAs

Yuki SAGAWA†, Tsuyoshi SADAKATA†, and Yusuke MATSUNAGA††

† Graduate School of Information Science and Electrical Engineering, Kyushu University
†† Faculty School of Information Science and Electrical Engineering, Kyushu University

Abstract In High Level Synthesis for FPGAs,arrays in behavioral description may be bound to the same memory block since the number of memory block is fixed for FPGAs. The number of access to such arrays at one step is limited to the number of memory port. If arrays that are accessed frequently are bound to the same memory block,array accessess will conflict with each other and the conflict will affect the number of steps. Therefore, memory binding and scheduling should be considered simultaneously. In this paper,we propose a heuristic algorithm that deals with memory binding and scheduling simultaneously under the memory size,the number of memory,and the number of memory port constraints subject to minimize the sum of maximum step for all Data Flow Graphs. Experimental results show that the proposed algorithm can find as a good solution as the approach using Simulated Annealing in many cases.

Key words High-Level Synthesis Memory Binding Scheduling FPGA

1. まえがき

半導体の微細化技術の進歩に伴い、大規模かつ複雑な機能をもつ LSI(Large Scale IC) の設計が可能になっている。設計規模の増大は、設計コストの増大や設計期間の長期化を招いている。一方、携帯電話やデジタルカメラのように、LSI を用いる民生デジタル製品の市場での競争は激化しており、製品の市場投入の短期化の要求が高まっている。

設計期間の短縮と増大する設計コストを抑制するために、FPGA(Field Programmable Gate Array) の利用と動作合成の導入が有効であると考えられる。FPGA とは、チップ製造後にチップが実現する機能を変更可能なデバイスの事である。新た

にマスクを作る必要がないため製造に関わる時間を短縮できる。また動作合成とは、動作記述と呼ばれる従来の RTL(Register Transfer Level) よりも抽象的なレベルでの回路の記述から、与えられた制約の下で求める回路の RTL 記述を自動的に合成する手法である。抽象的な記述を用いる事で記述ミスの抑制が期待できる。また制約を変える事で同じ動作記述から複数の RTL 記述を得る事ができるため、使用変更への柔軟な対応が可能である。さらには設計資産の再利用の点でも設計の短期化が期待できる。以上の点から設計期間の短縮と設計コストの抑制のために FPGA の利用と動作合成の導入を組み合わせる事が有効であると考えられる。

FPGA 向けの動作合成において考えるべき点としてスケジューリングとメモリバインディングが挙げられる。スケジューリングとは動作記述中の演算や配列の参照や書き込みが行なわれるステップを決定する処理である。ステップは処理のタイミングを表したもので、RTL におけるクロックサイクルに対応する。また動作記述中の配列をメモリへ割り当てる処理をメモリバインディングと呼ぶ。FPGA ではメモリサイズ、メモリ数、各メモリのポート数が決まっているため、メモリバインディングを行なう際にはメモリの制約を考慮しなければいけない。このときメモリ制約を満たすために複数の配列を一つのメモリへ割り当てなければならない場合がある。同じメモリへ割り当てられた配列に対するアクセス (参照や書き込み) を、同じステップにスケジューリングする場合はメモリのポート数を超えてはいけない。そのためメモリバインディングの結果次第で配列アクセスの並列度が変化する。配列アクセスが頻繁に起こるアプリケーションの合成では、演算の並列度が高くても配列アクセスの並列度が低いと最大ステップ数が多くなる可能性がある。そのためステップ数最小化を目的とした動作合成を考える場合、並列度を高めるメモリバインディングとスケジューリングを考える必要がある。図 1 にメモリバインディングにより配列アクセスの並列度が変化する例を示す。演算器数の制約は加算器が 2 つである。メモリは読み出しポート、書き込みポートがそれぞれ 1 つである。図 1 の (a) のバインディングでは配列アクセスが並列に行なえないため、最大ステップ数が 5 である。一方、図 1 の (b) のバインディングでは配列アクセスが並列に行なわれるため最大ステップ数が 4 となる。

メモリバインディングとスケジューリングを同時に考えた既存手法として、Simulated Annealing [9](以下、SA) を用いた手法が提案されている [1]。SA を用いた手法の問題点として良い解を出すまでに多くの時間を要する事が挙げられる。本論文ではスケジューリング結果から得られる情報を利用して、並列度を高めるメモリバインディングとスケジューリングを行うヒューリスティックスを提案する。SA を用いた手法の一部を実装し、提案手法と比較した実験では、SA を用いた手法に比べて多くの場合においてほぼ同じ結果を得られた。解の悪化は最悪な場合でも約 20 %程度であった。また最高で約 2000 倍、平均で約 1500 倍高速に解を出す事ができる事が分かった。

本論文の構成は以下である。2 章で動作合成の概要とメモリバインディング、スケジューリングについて述べる。3 章で問題の定義と提案手法の説明を行なう。4 章で関連研究の紹介を行なう。5 章で既存手法の一部と提案手法とナイーブな手法を比較した実験について述べる。6 章でまとめる。

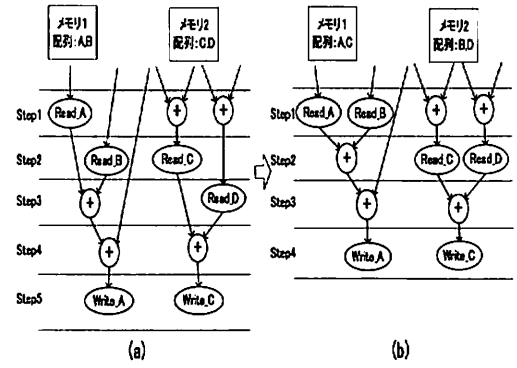

図 1　メモリバインディングの例

2. 準　　備

2.1 動作合成の概要

ここでは動作合成の概要 [7] について説明をする。動作合成は実現したい処理の順序を記述した動作記述から、与えられた制約を満たす RTL 記述を自動的に合成する設計自動化技術である。制約は使用するリソース (演算器やメモリ) の種類や数、面積、回路の動作にかかる総クロックサイクル数等である。合成される RTL 記述で用いられるリソースはあらかじめライブラリとして用意されており、動作合成では各処理にたいしてその処理を実現可能なリソースをライブラリから割り当てる。

動作合成の内部処理は以下の通りである。まず動作記述から動作合成における内部データ構造である CDFG(Control Data Flow Graph) に変換する。CDFG の各ノードは DFG(Data Flow Graph) に対応しており、エッジは処理の順序を表している。DFG は有向非循環グラフであり、各ノードが演算や配列アクセスを表し、各エッジがノード間のデータの依存関係を表している。配列アクセスを表すノードにはどの配列へのアクセスであるかが関連付けられている。得られた CDFG に対して以下の 4 つの部分問題が解かれる。1 つ目はライブラリから RTL において用いるリソースの種類や数を決定するモジュール選択問題である。2 つ目は各処理を実現するリソースを決定するアロケーション問題である。3 つ目は各処理が行われるタイミングを決定するスケジューリング問題である。4 つ目は各処理や変数、配列を実際のリソースのインスタンスに割り当てるバインディング問題である。バインディング後、演算器と記憶ユニット間のインターコネクトが決定される。最後に上記の処理の結果に基づいて、データパスとコントローラを生成する。

2.2 メモリバインディングとスケジューリング

バインディング問題の一つであるメモリバインディングとスケジューリングについて説明する。メモリバインディングとスケジューリングは互いに影響を及ぼしあう [1]。そのためスケ

ジューリングではメモリバインディング結果を考慮しなければならない。

まずメモリバインディングの説明を行なう。メモリバインディングではCDFG上の配列をメモリのインスタンス(以下では単にメモリと呼ぶ)に割り当てる。メモリと配列はパラメータとしてそれぞれビット幅、ワード数、サイズを持つ。サイズはビット幅とワード数の積である。メモリバインディングでは、ビット幅制約とワード数制約を満たさなければいけない。ビット幅制約とは、割り当てるられる配列のビット幅はメモリのビット幅以下でなければいけないという制約である。ワード数制約とは、割り当てられている配列のワード数の総和はメモリのワード数以下でなければいけないという制約である。

次にスケジューリングについて説明する。スケジューリングではDFG上の各演算ノード、配列アクセスノードをステップに割り当てる。ステップとは処理のタイミングを表したもので、RTLにおけるクロックサイクルに対応するものである。既存手法としてステップ数制約下でリソース数最小を目指す手法[10]やリソース数制約下でステップ数最小を目指す手法が提案されている[3]。以下では既存手法であるリストスケジューリング[3]について説明する。リストスケジューリングではステップ毎に割り当て可能なノードを全て列挙しリストへ入れ、優先度を基にノードをソートする。優先度が高い順に演算器数制約を満たす範囲でリスト内のノードを現在のステップへ割り当てる。割り当てる事ができなかったリスト内のノードは次ステップのリストへ入れられる。ステップ1から順に割り当て可能なノードの列挙、優先度を用いたソート、ステップへの割り当てを繰り返し、リストが空になると終了する。スケジューリングが終了すると、各DFGにおいてクリティカルパスが決定される。クリティカルパスとは、DFGのノードの内ステップ数が最大であるノードから入力側の一つのノードを順にたどって行く事で得られるノードの集合である。

メモリバインディングを同時に考える場合、スケジューリング前にメモリバインディングが決定しているとするとメモリのポート数制約を満たす必要がある。メモリのポート数制約とは、各ステップにおいてあるメモリ m に割り当てられている配列アクセスはメモリ m のポート数以上割り当てられてはいけないという制約である。メモリバインディングが決定している場合は、上記のリストスケジューリングはメモリのポート数制約を考慮する様に変更される。

3. 配列アクセスの並列度を高めるメモリバインディングとスケジューリング手法の提案

3.1 問題の定義

本論文で考える問題を定義する。メモリのサイズ、数、ポート数、演算器数制約下でメモリバインディングとスケジューリングを行い、各DFGの最大ステップ数の総和を最小化する問題である。前提としてモジュール選択、アロケーションは済んでいるとする。入力はCDFG、出力はメモリバインディング済み、スケジューリング済みのCDFGである。

簡単化のためにいくつかの仮定を置く。各メモリは全てビット幅、サイズ、ポート数が等しいとし、CDFG上の各配列とメモリのビット幅は全て等しいとする。配列のビット幅、ワード数、サイズはCDFGから得られる。メモリのサイズは制約で与えられるため、メモリのワード数はサイズを配列のビット幅で割ったものである。メモリのポートは全て読み書きポートとする。また各メモリ、各配列のワード数は2の指数乗とする。またCDFG上の各演算、各配列アクセスは1ステップで行なわれるとする。配列を分割する事は考えない。

3.2 提案手法

提案手法について説明する。メモリバインディングとスケジューリングを同時に考える場合、アプローチとしてメモリバインディングとスケジューリングを反復的に行なう事が考えられる。反復的に行なう場合、メモリバインディングとスケジューリングのどちらの処理を先に行なうかにより二通りのアプローチが考えられる。スケジューリングを先に行なう場合、メモリバインディングでは配列アクセスの並列度を満たすように配列をメモリに割り当てる必要がある。メモリバインディングを先に行なう場合、スケジューリングにおいてメモリのポート数制約を満たすように配列アクセスをスケジューリングする必要がある。提案手法では後者のアプローチを取っている。

提案手法は初期メモリバインディング後、リストスケジューリング[3]とスケジューリング情報を利用した再メモリバインディングを反復的に行ないコストを更新していく手法である。再メモリバインディングでは配列アクセスの並列度を高めるために二つの処理を行なっている。一つ目はスケジューリング後に得られるクリティカルパス(以下、CP)を利用した配列の列挙である。二つ目はリストスケジューリング時に用いる優先度を利用した割り当て先のメモリの決定もしくは交換する配列の決定である。本論文ではALAP(As Late As Possible)スケジューリング[4]におけるステップ数が多いほど優先度を高く設定している。入力はCDFG、出力はメモリバインディング済み、スケジューリング済みのCDFGである。メモリサイズ、メモリ数、メモリポート数、各演算器の数が制約として与えられる。コストは各DFGの最大ステップ数の総和である。

以下に提案手法のフローを説明する。まず初期メモリバインディングを行なう。初期メモリバインディングの結果を基にスケジューリングを行ない、コストを計算する。スケジューリング結果を基に、ヒューリスティックな手法を用いて再メモリバインディングを行なう。再メモリバインディング結果を基にスケジューリングを行いコストを計算する。この時コスト更新の有無を調べる。再メモリバインディングとスケジューリングを、コスト非更新回数が指定した回数に達するまで反復する。指定した回数に達した場合、その時点までに得られたコストが最小となるメモリバインディングとスケジューリング結果を出力する。各反復においては山登り探索を適用し、コスト更新の有無に関わらず反復を行なっている。また各反復において前回のメモリバインディング結果をタブーリスト[8]に保持しており、再メモリバインディングの結果が前回と等しくなった場合、メモリ割り当てを変更している。図2に提案手法の擬似コードを示す。図2中の S,N,P,R はそれぞれ制約として与えられるメモリ

```
Proposed(CDFG(V, E),S,N,P,R,k){
  nu = 0;
  A =getArray(CDFG);
  M =prepareMemories(A,S,N,P);
  B =InitialBinding(A,M);
  ListScheduling(CDFG,B,R);
  C =getCost(CDFG);
  C_min = C;
  B_min = B;
  while(nu < k){
    B =Re-Binding(CDFG,B,M);
    ListScheduling(CDFG,B,R);
    C =getCost(CDFG);
    if(C < C_min){
        C_min = C;
        B_min = B;
        nu = 0;
    }
    else
      nu+ = 1;
    tabulist = tabuUpdate(B);
  }
  ListScheduling(CDFG,B_min,R);
  Return CDFG;
}
```

図 2　提案手法の擬似コード

```
Re-Binding(CDFG(V, E),B,M,A,tabulist){
  At = enumArray(CDFG)
  for(a_i ∈ At){
    m_1 = selectMostPrioMem(a_i, B)
    a_1 = selectMostPrioArray(a_i, B)
    m_2 = selectNextPrioMem(a_i, B)
    a_2 = selectNextPrioArray(a_i, B)
    B = ChangeMostPrioMemOrArray(B, a_i, m_1, a_1)
  }
  if(checktabu(B,tabulist))
    B = ChangeNextPrioMemOrArray(B, a_last, m_2, a_2)
  Return B;
}
```

図 3　再メモリバインディングの擬似コード

のサイズ、数、ポート数、各演算器の数を表す。各関数については省略する。以下では各処理の詳細を述べる。

3.3　初期メモリバインディング

ここでは、制約を満たすメモリバインディングが存在する事を判定する方法と、存在すると判定された場合の初期メモリバインディングについて説明する。まず与えられた制約下で可能なメモリバインディングが存在するかを判定する。決められたサイズの A 個 の配列を、決められたサイズの M 個 のメモリに割り当てる事が出来るかどうかを決定する問題は、ビンパッキング問題において A 個の品物を M 個のビンに割り当てる事ができるかどうかを決定する問題として定式化できる。上記の問題は NP 完全である事が知られている [6]。NP 完全な問題に対しては多項式で厳密に解くアルゴリズムが存在しないだろうと考えられている。そのため本論文ではヒューリスティックな手法を用いて判定する。まず CDFG 上の各配列をワード数が多い順にソートする。次にソートした順に用意したメモリの中で残りのワード数が最も多いメモリに割り当てていく。割り当てる配列のワード数がメモリのワード数より多ければ、与えられた制約下ではメモリバインディングは不可能であると判定する。全配列を割り当てる事が出来た場合、その割り当て結果を初期メモリバインディングとする。上記の手法はヒューリスティックな手法であるので、可能なメモリバインディングが存在する場合に存在しないと判定してしまう可能性がある。

3.4　再メモリバインディング

再メモリバインディングにおいて行なう二つの処理について説明する。メモリの数を m、配列の数を n とするとメモリバインディングの組み合わせは m^n 考えられる。全ての組み合わせを列挙する場合、$O(m^n)$ の計算量となり実用的な時間では解けない。そのため再メモリバインディングでは、二つの処理で探索する組み合わせを制限している。再メモリバインディングではリストスケジューリングの結果から得られる情報を基に二つの処理を順に行なう。始めに CP の情報を利用してメモリ割り当てを変更させる配列の列挙を行なう。これらの配列を変更対象配列と呼ぶ。配列列挙の処理でメモリ割り当てを変更する配列を n 以下に制限する。次に変更対象配列に対して列挙した順に、メモリ割り当ての変更もしくは配列の交換を行なう。この時リストスケジューリングにおける優先度を利用して変更先のメモリの決定もしくは交換する配列の決定を行なう。メモリもしくは配列の決定の処理で、割当先のメモリもしくは交換先の配列を一つに制限する。変更対象配列全てに対してメモリの変更もしくは配列の交換が終了すると、新たなメモリバインディング結果が得られる。図 3 に再メモリバインディングの擬似コードを示す。以下では二つの処理について説明する。

3.4.1　変更対象配列の列挙

メモリ割り当てを変更する対象となる配列の列挙方法について説明する。提案手法の目的は各 DFG の最大ステップ数の総和を最小化する事である。各 DFG の最大ステップ数を削減するためには、各 DFG における CP 上のノードの最大ステップ数を減らせば良い。CP 上の配列アクセスの中には優先度が高い他の配列アクセスの存在によって、より遅いステップにスケジューリングするように判定された配列アクセスが存在する場合がある。CP 上のノードの最大ステップ数を減らすためには、上記の配列アクセスと優先度が高い他の配列アクセスを同ステップに並列に割り当てる事が考えられる。これらの配列アクセスを並列化対象配列アクセスと呼ぶ。並列に割り当てられなかった原因は、優先度の高い配列アクセスに関連づけられた配列が同じメモリに割り当てられていたためである。これらの配列を優先配列と呼ぶ。優先配列に対するアクセスと並列化対象配列アクセスを並列に割り当てるためには、それぞれの配列アクセスに関連づけられた配列の内、少なくともどちらか一方のメモリ割り当てを変更する必要がある。優先配列のメモリ割り当てを変更する事を考えた場合、CP 上の優先配列へのアクセスが次ステップへ追いやられる可能性がある。そのため並列化対象配列アクセスに関連付けられた配列を変更対象配列として

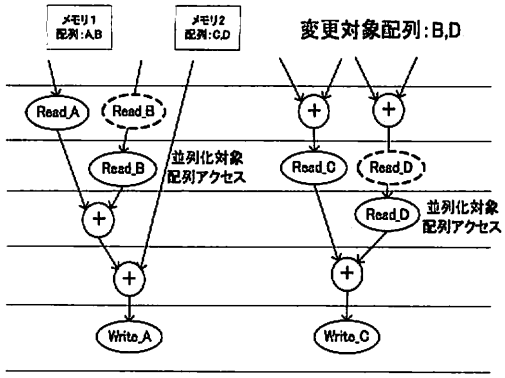

図 4　変更対象配列の例

列挙する。図 4 に図 1 の DFG における並列化対象配列アクセスと変更対象配列の例を示す。この例では全てのパスが CP である。点線で描かれたノードが割り当てられているステップが、並列化対象アクセスが最後に次ステップ以降に追いやられたステップである。図 4 の例では、変更対象配列は B,D である。

3.4.2　割り当てるメモリまたは交換する配列の決定

変更対象配列に対して列挙した順に、割り当てるメモリの変更もしくは配列の交換を行なう。割り当て先のメモリもしくは交換する配列を決定する際には、スケジューリング時に得られる情報を用いる。並列化対象配列アクセスがスケジューリング時に次ステップに追いやられた原因は、優先配列と同じメモリに割り当てられたためである。よって割り当てられるメモリの変更を行なう際に変更対象配列が優先配列と同じメモリに割り当てられた場合、再スケジューリング時にも同じ様に次ステップに追いやられてしまう。そのためスケジューリング時に優先配列を記憶しておき、バインディングの変更時には優先配列を考慮して変更先のメモリや交換する配列を決定する。割り当てるメモリまたは交換する配列の決定は以下の優先順位で行なう。

第一に割り当てられている優先配列の数がメモリのポート数より少ないメモリを選択する。第二に二つの配列を交換することで二つの配列が割り当てられていたそれぞれのメモリにおいて、割り当てられている優先配列がメモリのポート数より少なくなる配列を選択する。第三に二つの配列を交換することで二つの配列が割り当てられていたメモリの内一方のメモリにおいて、割り当てられている優先配列がメモリのポート数より少なくなる配列を選択する。第四に割り当てられている配列の数が最も少ないメモリを選択する。

以上の優先順位を基に選択されたメモリへ割り当てを変更する、もしくは選択された配列に対して二つの配列の割り当て先のメモリを交換する。ただしメモリのワード数制約を違反する場合は優先順位がより低いものを選択する。

4.　関連研究

ここでは関連研究を紹介する。メモリアロケーション、メモリバインディング、スケジューリングを同時に考えた手法として SA を用いた手法が提案されている [1]。メモリアロケーションとは配列が割り当てられるメモリの種類を決定する処理である。目的は、ユーザが指定したコストを最小化する事である。前提としてモジュール選択と演算器、レジスタアロケーションは済んでいるとしている。配列とメモリを、ビット幅を横方向、ワード数を縦方向とした 2 次元のタイルとして考えており、横方向の配列のグループ化と縦方向の配列のグループ化を行なっている。近傍操作は配列の横方向もしくは縦方向のグループ化、グループ化の解除、配列アクセスが割り当てられているステップの変更、配列グループが割り当てられるメモリの種類の変更である。スケジューリングでは、リストスケジューリング [3] を行なっている。[1] の手法においてメモリの種類を一種類、横方向のグループ化を行なわないと仮定しコストを各 DFG の最大ステップ数の総和とすると、本論文で定義した問題と同じ問題と言える。SA を用いた手法における問題点として、良い解を得るまでに多くの時間を要する事が挙げられる。

5.　実　　験

5.1　実験概要

提案手法の有効性を確かめる実験を行なった。提案手法とナイーブな手法と既存手法である SA を用いた手法の一部を実装し、ベンチマークプログラムに対してメモリ数制約を 1 から 27 まで変化させた際の、各 DFG の最大ステップ数の総和と実行時間の比較を行なった。

SA を用いた手法はビット幅の共有を行なわず、コストを総ステップ数とするように変更を加えている。近傍操作はある配列を別のメモリへ移動させる事とある二つの配列を交換する事としている。近傍操作によりバインディング結果がメモリのサイズ制約を超えた場合にはペナルティをつけており、温度が高いうちは受理されるが温度が下がるにつれて受理される確率が下がっていく。初期温度は 10 とし、1 万回の近傍操作を行なう毎に 0.9 倍に温度を下げている。終了条件は温度が 0.05 となった場合である。

ナイーブな手法では、全配列に対して全メモリへの移動または他の全配列との交換を行い、スケジューリングを行なっている。スケジューリング後にコストを計算し、最もコストが良かったメモリの移動もしくは配列の交換を行なっている。コストの更新がなくなると終了する。

ベンチマークプログラムは Strassen のアルゴリズム [5] である。Strassen のアルゴリズムは行列積を効率よく計算するアルゴリズムであり、内部で 27 個の配列を用いる。各配列の要素数は 64 である。各メモリはビット幅 32bit、ワード数 512、読み書きポート数を 2 としている。これらの値は xilinx 社の vertex シリーズを参考に設定した [2]。演算の並列度が十分に高い場合の配列アクセスの並列度が与える影響を調べるために、演算器制約は各演算器の数を十分に多く与えている。実験は Intel Xeon 5140(2.33GHz、Dual Core)、メインメモリ 4GB の環境で行なった。

5.2　結果と考察

実験の結果を表 1 に示す。#mem はメモリの数、Naive は

表1 実験結果

	Naive		Proposed		SA	
#mem	#Step	time[sec]]	#Step	time[sec]]	#Step	time[sec]]
4	50	89.12	51	8.11	48	16296
5	45	140.46	48	5.51	41	10711
6	45	146.97	38	11.58	37	12050
7	45	153.96	37	5.24	35	8292
8	45	158.95	35	5.09	33	9866
9	45	165.01	33	5.1	33	7354
10	45	171.18	36	4.02	30	8859
11	45	177.16	30	5.31	30	6628
12	45	183.38	30	4.45	30	8166
13	45	189.49	30	4.42	29	6128
14	45	195.68	29	6.46	28	7510
27	45	275.65	28	5.03	28	5610

ナイーブなアルゴリズム、Proposed は提案手法、SA は Simulated Annealing を用いた手法を示している。#Step は各 DFG の最大ステップ数の総和、Time は実行時間を示している。メモリ数 1〜3 の場合は、いずれの手法においても解が無いと判定されたため結果は省略する。また 14〜26 まではステップ数が同じであったため省略する。

実験の結果から提案手法は SA を用いた手法に比べて最高で約 2000 倍、平均で 1 500 倍、高速に解を見つける事ができている事が分かった。ステップ数の悪化は最悪で 20 %である。また提案手法はナイーブな手法に比べて、全ての場合において 10 倍ほど高速に解を見つける事ができ、多くの場合よりステップ数が少なくなるメモリバインディングを行なっている事が分かった。ただし、メモリ数が少ない場合にはナイーブな手法に比べてステップ数が多くなっている。この原因は、タブーリストのサイズが小さいためだと考えられる。再メモリバインディング時に前回と同じメモリバインディング結果にならないようにタブーリストに前回のメモリバインディング結果を保持しているが、前々回と同じメモリバインディング結果になっている場合がある。そのため 3 つのメモリバインディング結果を反復してしまい、他のメモリバインディングを探索できていないと考えられる。タブーリストのサイズを大きくする事で、3 つのメモリバインディング結果を反復する事は防ぐ事ができる。タブーリストのサイズは現状の実装では 1 としているが、大きくする事でより良い結果が期待できる。

6. ま と め

本論文では、配列アクセスの並列度を高める事でステップ数最小化を目指すメモリバインディングとスケジューリングを行なう手法を提案した。提案手法はメモリバインディングとスケジューリングを反復する手法である。メモリバインディングでは前回のスケジューリング時の情報を利用して、割り当てを変更する配列の列挙と割当先のメモリの決定を行なう。

提案手法の有効性を調べる実験を行なった結果、SA を用いた手法に比べて最高で約 2000 倍、平均で約 1500 倍、高速に解を出す事ができた。総ステップ数の悪化は最悪で 20 %であった。ナイーブな手法に比べると多くの場合において総ステップ数が少なくなった。しかし、メモリ数が少ない場合には総ステップ数が多くなっていた。この原因はタブーリストのサイズが小さいためだと考えられる。よってタブーリストのサイズを大きくする事で解の改善が期待できる。ただしサイズを大きくするとタブーリストのチェックにより時間がかかるため、どの程度のサイズにするかは検討する必要がある。

本手法では簡単化のために各配列のビット幅は全て等しいと仮定している。各配列のビット幅が異なる場合、メモリ制約を満たすためにワード数方向だけでなくビット幅方向の共有も考えなければいけない場合が考えられる。また本手法では配列の分割を考慮していない。サイズの大きな配列が現れる動作記述に対して本手法ではメモリの制約を満たせないと判定される。この時配列を分割する事で制約を満たせる様になる場合が考えられる。また配列を分割し別々のメモリへ割り当てる事で、分割された二つの配列へ同時にアクセスできるようになり配列アクセスの並列度を上げる事ができる場合がある。以上の点は今後検討する。

謝辞 本論文を執筆するにあたって、SA の実装に協力して頂いた九州大学の小玉翔氏に感謝します。

文 献

[1] Herman Schmit, Donald E.Thomas, Array Mapping in Behavioral Synthesis, in the Proceedings of the eighth International Symposium on System Synthesis(ISSS'95), pp.90-95, 1995.

[2] http://japan.xilinx.com/support/documentation/user_guides/j_ug070.pdf.

[3] R.Jain, A.Mujumdar, A.Sharma, H.Wang, Empirical Evaluation of Some High-Level Synthesis Scheduling Heuristics, in the Proceedings of the 28th Design Automation Conference, pp.210-215, 1991.

[4] Daniel Gajski, Allen Wu, Nikil Dutt, Steve Lin, HIGH-LEVEL SYNTHESIS:Introduction to Chip and System Design, Kluwer Academic Publishers, Norwell, Massachusetts, 1992.

[5] Strassen, Volker, Gaussian Elimination is not Optimal, Numer.Math, vol.13, no.4, pp.354-356, 1969.

[6] M.R.Garey, D.S.Johnson, Complexity results for multiprocessor scheduling under resource constraints, SIAM Journal on Computing, vol.4, pp.397-411, 1975.

[7] Giovanni De Micheli, SYNTHESIS AND OPTIMIZATION OF DIGITAL CIRCUITS, McGraw-Hill.Inc, Princeton, 1994.

[8] F.Glover, M.Laguna, Tabu Search, Kluwer Academic Publishers, Norwell, Massachusetts, 1997.

[9] S.Kirkpatrick, P.D.Gellat, M.P.Vecchi, Optimization by simulated annealing, Science, vol.220, no.4598, pp.671-680, 1983.

[10] P.G.Paulin, J.P.Knight, Force-Directed Scheduling for the Behavioral Synthesis of ASIC's, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.8, no.6, pp.661-679, June, 1989.