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

2007－ＳＬＤＭ－129／2007－ＥＭＢ－4（11）
2007／3／16

SAに基づくFPGA配置アルゴリズムの領域分割による並列化

岡嶋 知宏† 有内 雄司† 久我 守弘†† 飯田 全広†† 末吉 敏則††

† †† 熊本大学大学院 自然科学研究科 〒860-8555 熊本市黒髪2-39-1
E-mail: †{okajima,ariuchi}@arch.cs.kumamoto-u.ac.jp, ††{kuga,iida,sueyoshi}@cs.kumamoto-u.ac.jp

あらまし 配置処理はFPGA自動設計フローの中で最も時間を費やす工程の一つである．近年では，FPGAの性能向上によって実装回路の大規模化が進み，自動設計に要する時間が急激に増加しているため，FPGA配置の高速化は重要な課題となっている．本稿では，最も広く用いられているSAに基づくFPGA配置を対象として，クラスタコンピュータ上で領域分割による並列化を適用する．本来，SAは逐次性の強いアルゴリズムであり，並列化には適していない．しかし，FPGAは一般に規則的構造を持つため，物理的領域に従って問題を分割することが可能である．評価の結果，領域分割による並列化が大規模回路の配置に適しており，大規模回路において線形に近い速度向上を数%のコスト劣化で達成可能であることが分かった．

キーワード FPGA配置，並列アルゴリズム，領域分割，クラスタコンピュータ

Parallelization with area partitioning for FPGA placement algorithm base on SA

Tomohiro OKAJIMA†, Yuji ARIUCHI†, Morihiro KUGA††,
Masahiro IIDA††, and Toshinori SUEYOSHI††

† †† Graduate School of Sience and Technology, Kumamoto University,
2-39-1 Kurokami, Kumamoto 860-8555, Japan
E-mail: †{okajima,ariuchi}@arch.cs.kumamoto-u.ac.jp, ††{kuga,iida,sueyoshi}@cs.kumamoto-u.ac.jp

Abstract Placement is one of the most time-consuming processes in automatically logic synthesis and layout for FPGAs. As FPGAs have improved circuit performance, the circuit scale that is implemented by FPGAs becomes larger. Then the computation time devoted to placement has grown dramatically. In this paper, we applied the parallel algorithm that based on area partitioning to FPGA placement using SA on cluster computer. Generally FPGAs have regular structure, therefore, area partitioning technique is effective. Experimental results show that parallelization with area partitioning is effective when the circuit size is large. For large circuits, it achieves nearly linear speed up without significant cost deterioration.

Key words FPGA placement, parallel algorithm, area partitioning, cluster computer

1. はじめに

FPGA（Field Programmable Gate Array）は，1985年にSRAMベースのLUT（Look Up Table）に基づく基本論理ブロックを2次元アレイ上に配したデバイスとして米国Xilinx社から発表されて以来，今や大規模PLD（Programmable Logic Device）の大部分を占めている．最近では，最新のプロセス技術が積極的に取り込まれており，動作周波数，集積度，そして，価格などの面で飛躍的に改善されてきている．そのため，グルーロジックや小規模回路のプロトタイプとしての利用だけでなく，コスト的に有利な少量生産のアプリケーション分野，さらに，ディジタル民生機器などの量産に採用する動きが広がりつつある．

ここで問題となっているのが，自動設計に要する時間の急激な増加である．自動設計の中でも，配置は最も時間を要する工程の一つであり，処理の高速化が望まれている．現在，SAに基づく配置手法が良好な解を生成するとして広く用いられているが，本来，SAは強い逐次性を内包しており，並列化に向いていない．しかしながら，一般にFPGAは規則的構造を有しており，物理的領域に従って問題を分割することによる悪影響は少ないと考えられる．そこで，本稿ではSAに基づくFPGA配置にクラスタコンピュータ上で領域分割による並列化を適用

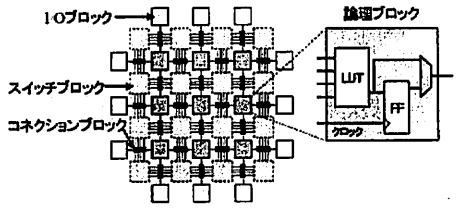


図 1 アイランドスタイル FPGA の概略図

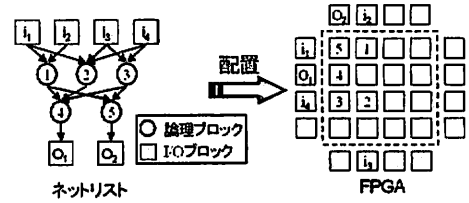


図 2 FPGA 配置の概念図

し，速度向上およびコスト劣化について評価する．

以下，2 章で FPGA 配置について述べ，3 章でその並列化について説明を行う．そして，4 章で本稿で採用した領域分割による並列配置の評価結果を示し，最後に 5 章をまとめとする．

2. FPGA 配置

本稿では，最も一般的であるアイランドスタイル FPGA を対象とし，その配置問題を扱っている．以下，対象 FPGA のアーキテクチャおよび FPGA 配置問題についての説明を行う．

2.1 対象とする FPGA アーキテクチャ

アイランドスタイル FPGA の概略図を図 1 に示す．論理ブロック，スイッチブロック，コネクションブロックをそれぞれ 2 次元配列状に配置し，論理ブロックを取り囲むように信号線が引かれている．さらに，その外周には外部と論理ブロック間のインタフェースとして I/O ブロックが配されている．論理ブロックは，順序回路や組合せ回路を実装できる基本ユニットであり，LUT (Look Up Table)，FF (Flip Flop)，そして，セレクタで構成される．スイッチブロック内には，信号線の交差部分上にパストランジスタが設置してあり，これらパストランジスタの ON/OFF を切り替えることで，信号の流れを制御可能である．コネクションブロックは，論理ブロックと信号線との間においてスイッチブロックと同様の働きをする．

つまり，FPGA は，論理機能を実現するプログラマブルな論理要素，ならびに論理要素間の接続を実現するプログラマブルな接続要素から構成される．従って，通常の LSI チップ設計がマスク生成を目的とするのに対して，FPGA の設計では論理要素ならびに接続要素の "状態" を決定することが目的となる．

2.2 FPGA 配置問題

図 2 に示すように，FPGA 配置問題は，FPGA を構成する要素（論理ブロックおよび I/O ブロック，以降合わせてブロックと呼ぶ）とそれら接続関係が記されたネットリストを入力として，その各論理要素を FPGA 上のどの位置に割当てるかを決定する問題である．配置は主として総配線長の最小化を目的として行われるが，配置のたびに配線を行うことは困難であるため，仮想配線長を見積もるコスト関数を定義し，その最小化を目的とするのが一般的である．

FPGA の論理要素は 2 次元アレイ状に規則正しく整列しているため，配置問題は 2 次割当て問題として定式化できる [1]．2 次割当て問題は NP 完全問題であることが知られており，実用規模の回路に対して厳密解法の適用は困難である．そのため，各要素間の配線をバネに見立て，それらの力学的エネルギーが

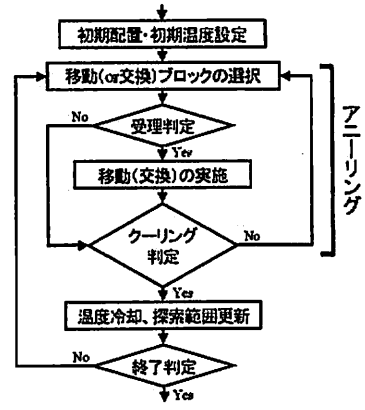


図 3 FPGA 配置フロー

最小となるように配置の調整を行う Force Directed アルゴリズム [2]，各論理要素の配置位置を段階的な領域分割によって決定する分割手法 [3]，そして，ランダムな配置から処理を開始し，SA (Simulated Annealing) に基づき配置を改善する手法 [4] などの様々な近似解法が提案されている．これらの中でも，SA に基づく配置手法はデバイスに依存する部分が少なく汎用的であり，そして，良好な解を生成するとして広く用いられている．本稿では，SA に基づく配置手法を取り扱う．

2.3 SA に基づく FPGA 配置手法

SA [5] は，局所探索法の一つである乱数降下法に，局所的最適解への収束を回避するために温度の概念を加えた手法である．SA に基づく FPGA 配置フローを図 3 に示す．

各処理についての説明を以下に行う．

初期配置・初期温度設定

一般に，初期配置はランダムに生成される．また，初期温度は，ほとんどすべての遷移が受理されるように十分高温に設定される．

移動（交換）ブロックの選択

まず，移動元のブロックをランダムに選択する．つづいて，移動先の位置を決定する．その際，FPGA の全域からではなく，移動元のブロックを中心とした一定範囲（以後，探索範囲と呼ぶ）の内部から選択される．探索範囲は FPGA 全域をスタートとし，基本的には，次第に減少させることで効率的な探索が行われる．

受 理 判 定

移動先の位置が他のブロックにより占有されている場合には交換，そうでなければ移動を行った際のコスト変化 ΔC を基に

受理判定は行われる．総配線長を見積もるためのコスト関数には，以下の式で表現されるものが広く用いられている(注1)．

$$C = \sum_{i=1}^{Nnum} q(numpins(i))\,[bb_x(i) + bb_y(i)] \qquad (1)$$

回路中の全てのネット i に関して，そのバウンディングボックスのマンハッタン距離 "$bb_x(i) + bb_x(i)$" にネットに属するブロック数 $numpins(i)$ に応じた重み q を加えた値を計算し，それらの総和としてコスト C が定義される．ここで，バウンディングボックスとは，ネットに属する全てのブロックを内包するような最小の矩形領域のことである．

選択した移動（交換）は，以下の確率で受理される．

$$Accepet(\Delta C, T) = \begin{cases} 1 & (\Delta C \leq 0) \\ e^{-\frac{\Delta C}{T}} & otherwise \end{cases} \qquad (2)$$

クーリング判定

移動（交換）ブロックの選択および受理判定は，一定期間繰り返される．この一連の処理をアニーリングと呼び，繰り返す期間のことをアニーリング周期と呼ぶ．ここでは，アニーリング周期を満たしたかどうかの判定を行い，条件を満たしていれば温度の冷却および探索範囲の更新を行い，満たしていない場合は移動（交換）ブロックの選択処理に戻る．

終 了 判 定

一定期間コスト変化が見られない場合，一定回数以上の試行を実施した場合などの判定条件により，処理の終了を判断する．終了条件を満たすまで，温度を冷却しつつアニーリングを繰り返す．

温度冷却のスケジューリング，探索範囲の更新方法，アニーリング周期，そして，終了判定などをどのように設定するかによって，解の質や計算時間が大きく変化する．

3. FPGA 配置の並列化

3.1 SA に基づく FPGA 配置の並列化手法

FPGA 配置の並列化手法として，定期同期型，受理時同期型，並列移動，そして，領域分割が挙げられる．以下に各手法を示す．

定期同期型

第一のアプローチとして，複数ノードで個別に配置を実行し，定期的に最良の解を選択する方式が考えられる．以下に処理手順を示す．

（1） 各ノードが異なる初期配置からスタート
（2） 各ノードで配置を実施
（3） 定期的に全体で同期を取り，最良の解を選択・分配

以降，終了条件を満たすまで (2)〜(3) を繰り返す．

この手法の問題点は，N ノードで実行した場合に，N 倍の探索能力が得られる訳ではないという点である．速度向上を得るためにアニーリング周期を $1/N$ として実行すると，単一ノードで実行した場合と比較して大幅にコストが劣化してしまうことが報告されている [6]．

(注1)：最近では，配線遅延や混雑さを加味したコスト関数も使用されている．

受理時同期型

第二のアプローチとして，単一の配置を高速化する手法がある．安直にこれを実行する手法として，受理時同期型が挙げられる．この手法は以下の手続きをとる．

（1） 全ノードが同一の初期配置でスタート
（2） 各ノードで異なる移動（交換）の評価
（3） いづれかのノードで受理が発生した時点で同期を取る
（4） 全ノードで同じ遷移を適用

以降，終了条件を満たすまで (2)〜(4) を繰り返す．

移動（交換）の受理確率が非常に低い場合には有効であるが，通常は処理に対して同期オーバーヘッドが大きく，この手法で速度向上を得ることは困難である．

並 列 移 動

受理時同期型では無作為に移動（交換）を実施しているため，受理発生のたびに同期が必要であった．そこで，並列移動では，異なるブロックペアであれば同時に移動（交換）の評価・実施が可能である点に着目し，並列に実行可能な移動（交換）をあらかじめ生成する．

この手法は以下の手続きをとる．

（1） 全ノードが同一の初期配置でスタート
（2） 並列に実行可能な移動（交換）を生成
（3） 各ノードで別々の移動（交換）を評価，実施
（4） 各ノードの評価結果を統合して状態遷移

以降，終了条件を満たすまで (2)〜(4) を繰り返す．

この手法では，複数の移動（交換）が同時に受理された場合に，それが不当である可能性を持つ．これは，同一ネット上の異なるブロックが移動された場合に，一方の移動による変化を他方からは知ることができないためであり，無視して処理を進めると大幅なコスト劣化に繋がる．移動（交換）生成時に異なるネットに属するブロックを選択することで対処可能ではあるが，このアプローチでは，共有メモリマシン上での実装でさえ速度向上が得られないことが報告されている [6]．

領 域 分 割

並列移動による問題点は，移動（交換）ブロック生成に対する制限が非常に厳しいことである．そこで，FPGA を物理的な領域に従って分割し，各ノードに個々の領域を割当てる手法が考えられる．以下に手順を示す．

（1） 全ノードが同一の初期状態でスタート
（2） 領域を分割し，各ノードに割当てる
（3） 各ノードが担当領域内部で移動（交換）を実施
（4） 定期的に全体で同期を取って結果を統合

以降，終了条件を満たすまで (2)〜(4) を繰り返す．

この手法では，並列移動と同様に，移動（交換）が不当に受理される可能性を持つが，異なる領域に跨るネット上でしか発生しない．そのため，この影響は並列移動と比較して非常に小さく，定期的に領域間で同期を取る程度で最終的なコスト劣化を抑制可能である．

領域分割による別の問題は，ブロックの移動可能な範囲が分割した領域内部に限定されることである．常に固定された領域では早期に局所解へと陥ってしまうため，領域の分割方法を切

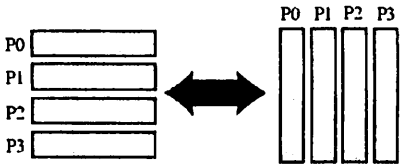


図 4 領域更新方法

り替え，FPGA 全域に渡る移動を許容する必要がある．切り替え方法は，任意位置に配されたブロックが任意の位置へ自由に移動できることを保証するようなものが望ましい．つまり，一度の切り替えによって移動可能な範囲が大きく変更されることが良いと考えられる．直感的な手法として，図 4 に示すように水平分割と垂直分割を交互に入れ替える方法がある．

以上のように，領域分割では，定期的に同期および領域の切り替えを行うことでコスト劣化を抑制しつつ並列実行が可能である．また，不当な遷移の少なさより，同期をそれほど頻繁に行わなくてもコスト劣化の抑制が可能であり，速度向上が期待される．そのため，本稿では，FPGA 配置の並列化手法として領域分割を採用している．

3.2 領域分割による並列化の実装

ここでは，本稿での領域分割による並列化の実装方法について説明する．有名な FPGA 向け配置配線ツールである VPR (Versatile Place and Route) [7] をベースとして，領域分割による並列化を実装した．

領域分割による FPGA 配置の処理フローを図 5 に示す．以下に各工程について説明を行う．

初期配置・初期温度設定

各ノードが個別にネットリストを読み込み，同一シードの擬似乱数を用いて初期配置，および，初期温度の設定を行う．VPR で使用される擬似乱数は初期シードにのみ依存するため，初期シードが同じであれば同一の初期状態が得られる．

アニーリング

各ノードが担当している領域内部でブロックの移動（交換）を行う．ただし，I/O ブロックは初期配置で固定し，論理ブロックのみの移動（交換）を行うものとした．

VPR において，探索範囲は FPGA 全域をスタートとし，移動（交換）の受理確率が 0.44 に近づくように制御される．領域分割を行うと，探索範囲の最大値が個々の領域サイズに制限されてしまう．しかし，処理の後半では探索範囲は非常に狭くなるため，これによる影響はそれほど大きくない．

領域更新判定

一温度の間に何回領域を切り替えるかをパラメータとして与え，アニーリング周期内で等間隔に切り替えを実行することとした．以後，切り替える回数を "領域更新回数"，切り替えまでの周期を "領域更新周期" と呼ぶ．領域更新周期分のアニーリングを行ったかの判定を行い，領域更新を実行するかどうか判断する．

領域更新

領域の分割方法には，水平分割と垂直分割を交互に繰り返す方式を採用している．領域更新時には，他のノードで実施された移動（交換）を反映する必要があるため，各ノードは担当領域内部の論理ブロックの位置情報をマルチキャストし，全ノードで同期を取る．

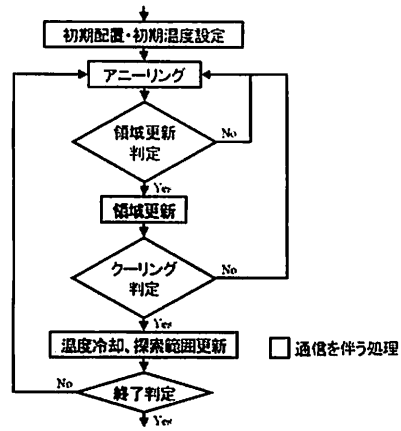


図 5 領域分割による FPGA 配置フロー

VPR では以下の式でアニーリング周期 M を与えている．

$$M = InnerNum \cdot (N_{block}^{4/3}) \quad (3)$$

N_{block} はブロックの総数であり，$InnerNum$ はデフォルトでは 10 に設定されている．従って，ブロック数が増加する程，アニーリング周期は指数関数的に増加し，領域更新時の通信量は処理に対して相対的に小さくなる．

クーリング判定

速度向上を得るために，アニーリング周期には，VPR のデフォルト値を実行ノード数で割った値を用いている．つまり，アニーリングの試行総数は同一である．

温度冷却，探索範囲更新

VPR では，温度冷却および探索範囲更新時に，前温度での移動（交換）の受理確率を用いる．全ノードが同一の遷移を辿るために，受理確率には全ノードの平均値を用いる．通信データ量は小さく，その頻度も少ないため，この通信による影響はほとんど発生しない．

終 了 判 定

VPR では，ネットあたりのコストに比べ温度が十分に低くなった時点で処理を終了する．そこで，全ノードの平均コストを求めた後に終了判定を行う．

4. 評　　価

4.1 評 価 方 法

領域分割による並列化の実装は，表 1 に示す環境で行っている．なお，並列化には，SCore [8] で提供される mpich-1.2.5 を用いた．また，使用したクラスタコンピュータは，SMP (Symmetric Multiple Processor) 構成で 1 台に 2 つのプロセッサを搭載しているが，今回は 1 台につき 1 プロセッサのみを使用し

表1 クラスタコンピュータの仕様

クラスタコンピュータ	HP ProLiant DL380 G3
台数	16台
CPU	Intel Xeon 2.8GHz × 2
OS	Fedora Core 3
並列プログラミング環境	SCore version 5.8.3

表2 ベンチマーク回路

分類	回路名
小規模回路	alu4, apex2, apex4, bigkey, des, diffeq, dsip, ex5p, misex3, s298, seq, tseng
中規模回路	elliptic, ex1010, frisc, pdc, spla
大規模回路	clma, s38417, s38584.1

ている(注2)．SAに基づくFPGA配置の領域分割による並列化では，ノード数（分割数）および領域更新回数によって速度向上率およびコスト劣化の傾向が変化する．そこで，表1に示す環境で領域分割による並列化の実装を行い，複数のノード数および領域更新回数でVPRとの比較を行っている．

評価回路には，表2に示す20個のMCNCベンチマーク回路[9]を使用しており，これらの回路は，T-VPack[7]により4LUT+FFのみで構成される論理ブロックへクラスタリングを行っている．以降では，論理ブロックの数が3,500以下のものを小規模回路，6,000以下のものを中規模回路，そして，それ以上のものを大規模回路と分類して議論する．

領域分割では，ノード数4，8，16の3パターンおよび，領域更新回数4，8，16，32の4パターンの全組合せである12パターンでの配置を各回路に対して行っている．20の評価回路全てについて初期シード1〜10の10パターンで測定し，評価にはその平均値をさらに回路規模毎に幾何平均した値を用いている．そして，比較として，“-place_algorithm bounding_box”，“-fix_pins random”，“-nodisp”，および，“-place_only”のオプションを指定したVPRを使用している．また，速度向上時のコスト劣化を比較するために，アニーリング周期を1/4，1/8，1/16とした場合の測定も行っている．

4.2 評価結果

配置速度の結果を図6に示す．横軸がノード数，縦軸がデフォルトのアニーリング周期を用いたVPRの実行時間を1とした時の速度向上率を示す．菱形はVPRの実行結果であり，比較のためにアニーリング周期を1/4，1/8，1/16としたものをノード数に対応させて表示している．また，実線はVPRの近似曲線である．

コスト増加量の結果を図7に示す．図6と同様に横軸はノード数であり，対応するVPRの値を菱形，そして，その近似曲線を実線で示している．縦軸はVPRのコストからの増加量を百分率で示している．つまり，デフォルトのアニーリング周期を用いたVPRを0とし，VPRのコストから何%分コストが増加しているかを示している．

(注2)：2プロセッサを用いた場合，ファイル読み込み時に競合が発生するというアルゴリズムとは無関係の問題が生じたため，1プロセッサのみの使用とした．

領域分割では，速度向上を得ると同時にコスト増加も発生している．VPRでもコスト劣化を許容するならば，アニーリング周期を短くすることで速度向上を得られる．従って，VPRで同一のコスト増加を許容するときの速度向上率を近似曲線を用いて計算し，その値で速度向上率を割ったものが，本質的な速度向上率といえる．図8には，その値を示している．

4.3 結果の考察

まず，図6より速度に関して以下のことが分かる．

- ノード数増加に伴って高速化
- 更新回数が多くなる程，速度向上率は低下
- 回路規模が大きくなる程，高速化の傾向が良い

大規模回路における速度向上率は特に良く，いくつかの点においては実行ノード数分以上の速度向上を示している．アニーリング周期を“1/ノード数”にしかしていないのに，それ以上の速度向上が得られているのは，温度冷却回数が少なくなったことが原因である．領域分割では，処理の初期部において探索範囲が制限され，その影響で受理率が高くなる傾向にある．そして，温度の冷却スケジューリングには受理率が関与し，受理率が高い区間では大幅に冷却されるようになっている．その結果として，温度冷却回数の減少し速度向上につながっているが，その分コスト劣化が生じていると考えられる．

次に，図7よりコストに関して以下のことが分かる．

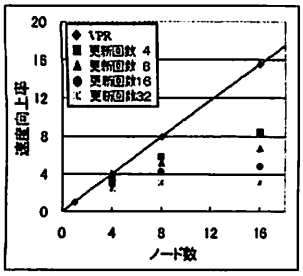
- ノード数増加に伴って劣化が大きくなる
- 領域更新回数が多くなるほどコスト劣化は低減
- 回路規模が大きくなるほどコストも全体的に増加

大きな特徴として，回路規模の増加に伴うコスト劣化の増大が，更新回数によって大きく異なる点が挙げられる．更新回数4の時は，回路規模が大きくなるにつれて大幅にコスト劣化が増加し，大規模回路においては1台で実行しているVPRとそれ程代わらないコスト劣化を生じている．それに対して，更新回数32では，大規模回路16ノードでさえ1.5%程度のコスト増加に抑えられている．つまり，回路規模が大きくなる程，更新回数を増加させた時の速度向上率低下と比較して，コスト劣化抑制の効果が大きいという特徴を持つ．これは，回路が大規模である程，領域分割による並列化の効果が得られるということを示している．回路規模が大きい程長い処理時間を費やし，高速化を必要とするため，この特徴は重要である．

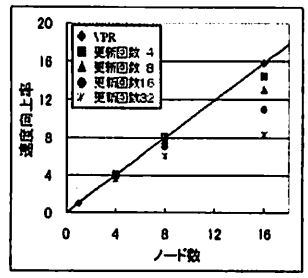
最後に，図8により，コスト増加率を加味した速度向上率について考察を行う．良好な結果を示している大規模回路のノード数16更新回数32の場合であっても，実質の速度向上率は4.5程度であり，理想的には16であることを考えると，小さな値である．しかし，この値はコスト増加率の僅かな変化で大きく変動する．領域分割方法や更新方法などを改良し，コスト増加率を限りなく0に近づけることが重要となる．

5. まとめ

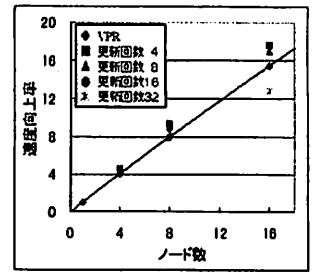
本稿では，FPGA配置の高速化を目的とし，領域分割による並列化の実装および評価を行った．その結果，大規模な回路であれば線形の速度向上を数%のコスト増加で実現可能であることを示した．さらに，回路規模が大きくなる程良い傾向を持つ



小規模回路

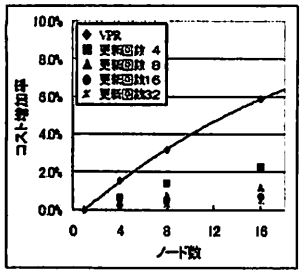


中規模回路

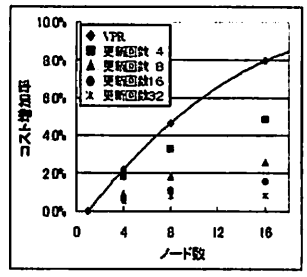


大規模回路

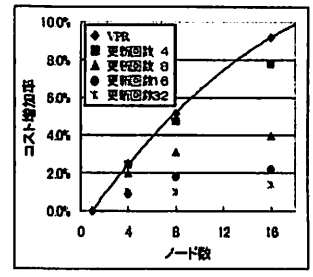
図 6　速度向上率



小規模回路

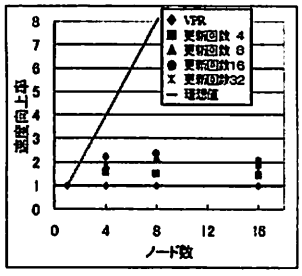


中規模回路

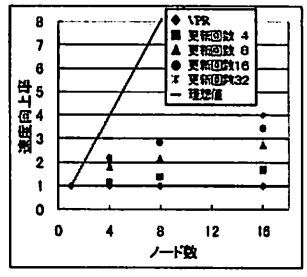


大規模回路

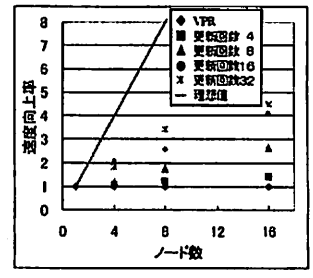
図 7　コスト増加率



小規模回路



中規模回路



大規模回路

図 8　コスト増加を加味した速度向上率

という特徴を持つため，領域分割が FPGA 配置に適した並列化手法であることが分かった．しかし，コスト増加率を加味した速度向上率は満足できるものではなく，コスト劣化抑制のために領域分割方法などの検討が必要である．

また，本稿では配置時のコストで評価しているが，実際には配線後の結果が重要である．現在，複数の配線長を持つアーキテクチャを想定して配線を行うと，配置時のコストと配線後のクリティカルパス遅延間の相関が低いことが分かっている．単純なコスト関数を採用していることが原因と考えており，今後遅延や混雑さを加味したコスト関数に対応する必要がある．

文　献

[1] M. Hanan, and J.M. Kurtzberg, "A review of the placement and quadratic assignment problems," SIAM Review, vol.14, pp.324–342, 1972.

[2] H. Li, W.K. Mak, and S. Katkoori, "Force-directed performance-driven placement algorithm for fpgas," isvlsi, vol.00, pp.193–198, 2004.

[3] M. Hutton, K. Adibsamii, and A. Leaver, "Timing-driven placement for hierarchical programmable logic devices," FPGA '01: Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays, pp.3–11, 2001.

[4] P. Banerjee, "Accelerators for fpga placement," The 4th Annual Inter Research Institure Student Seminar in Computer Science, 2005.

[5] S. Kirkpatrick, J.C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," Science, vol.220, no.4598, pp.498–516, 1983.

[6] M. Haldar, A. Nayak, A. Choudhary, and P. Banerjee, "Parallel algorithms for fpga placement," GLSVLSI '00: Proceedings of the 10th Great Lakes symposium on VLSI, pp.86–94, 2000.

[7] "Vpr and t-vpack: Versatile packing, placement and routing for fpgas package ver 4.30," http://www.eecg.toronto.edu/vaughn/vpr/vpr.html, 1997.

[8] "PC Cluster Consortium," http://www.pccluster.org/.

[9] S. Yang, "Logic synthesis and optimization benchmarks user guide version," Tech. Report, Microelectronics Center of North Carolina, 1991.