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

2007－ＳＬＤＭ－128（６）
2007／1／17

LSIフロアプランニングに対する遺伝的アルゴリズムとタブー探索に基づく並列アルゴリズムとそのPCクラスタ上での実現

島津 尊是† 若林 真一† 永山 忍†

†広島市立大学情報科学部 〒731-3194 広島市安佐南区大塚東 3-4-1
E-mail: {wakaba,s_naga}@ce.hiroshima-cu.ac.jp

あらまし 本研究では，LSIフロアプランニングに対する遺伝的アルゴリズム (GA) とタブー探索 (TS) を組み合わせた並列アルゴリズムを提案する．提案手法は3段階に分れている．第1段階では，GAを用いて解空間の広域探索を実現する．効率のよい探索を実現するため，目的関数は簡単化したものを用いる．第2段階では，第1段階で得られた優良解集合に対してGAに基づく探索を行うが，解の詳細な評価を行うため，目的関数は第1段階より詳しいものを用いる．最期に第3段階では第2段階で得られた優良解集合に対してTSに基づいてさらなる解の改善を行う．計算時間の短縮のために，GAとTSに並列処理を導入し，手法全体を並列アルゴリズム化した．さらに，提案並列アルゴリズムをPCクラスタ上に実現し，計算機実験による評価を行った．

キーワード 遺伝的アルゴリズム，タブー探索，LSIフロアプランニング，並列アルゴリズム

A Parallel Algorithm Based on Genetic Algorithm and Tabu Search for LSI Floorplanning and Its Implementation on a PC Cluster

Takayoshi SHIMAZU†, Shin'ichi WAKABAYASHI†, and Shinobu NAGAYAMA†

† Faculty of Information Sciences, Hiroshima City University
3-4-1, Ozuka-higashi, Asaminami-ku, Hiroshima 731-3194, Japan
E-mail: {wakaba,s_naga}@ce.hiroshima-cu.ac.jp

Abstract This paper proposes a parallel floorplanning algorithm for VLSI floorplanning, which was based on genetic algorithm (GA) and tabu search (TS). The proposed method consists of three phases. In the first phase, solutions were globally searched by GA on each PC independently. In the second pahse, from good solutions found in the first phase, further search was performed based on GA in each PC, with interchanging solutions among PCs. In the last phase, TS was executed in each PC to improve the best solutions found in the second phase to get a final solution. The proposed method was implemented with the MPI (Message Passing Interface) library on a PC cluster. Experimental results show the effectiveness of the proposed method.

Key words genetic algorithm, tabu search, LSI floorplanning, parallel algorithm

1. はじめに

LSI設計においてフロアプラン設計は回路の性能に大きな影響を与える重要な設計工程である．しかし，この工程はモジュール数の増加に伴い，通常の解探索手法を用いたのでは計算時間が爆発的に増加してしまう．本研究では解の探索と改良を3段階に分け，PCクラスタ上の各PCが遺伝的アルゴリズムとタブー探索を組み合わせたフロアプランニング手法を並列に実行することで，解空間を大域的に探索し，よりよい解を求める手法を提案する．

2. 定　　義

2.1 フロアプランニング問題

LSIフロアプランニング問題とは，矩形（チップ）内に与えられた複数の矩形（モジュール）を互いに重なることなく面積が最小化するように配置する問題である．ただし，LSI設計においては総配線長や配線混雑度も性能に影響を与えるため，問題の目的関数としてチップ面積だけでなく，総配線長の最小化，配線混雑度の最小化，分散化等を考慮する必要がある．

本研究で扱うフロアプランニング問題は幅と高さの情報が与えられた m 個のモジュールと n 本のネットで構成されるネッ

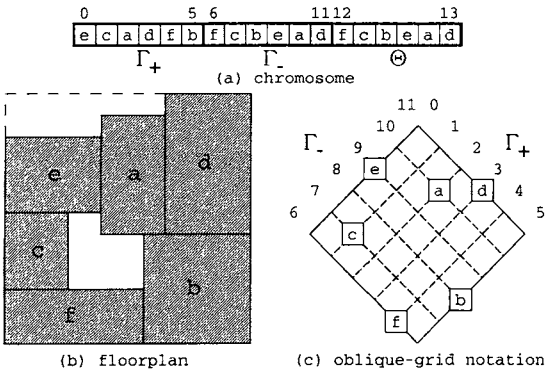

図1 個体表現

トリストを入力とし，互いに重なりなく平面上に配置した各モジュールの位置を出力とする．得られた配置において，全てのモジュールを囲む最小矩形の面積をチップ面積とし，配置されたモジュールから各ネットの配線長を計算し，総配線長を求める．得られたモジュール配置はチップ面積と総配線長の重み付き和を目的関数として評価する．

2.2 遺伝的アルゴリズムとタブー探索

遺伝的アルゴリズム（*genetic algorithm, GA*）は，生物界において染色体の交叉や突然変異によって新しい世代が作られ，弱者が淘汰され強者だけが生き残っていくという生物の進化のメカニズムを最適化問題の解法に応用したものである[4]．*GA* では逐次改良のように1つの許容解(以下では個体と呼ぶ)を少しずつ改良していくのではなく，複数の個体を個体の評価値に関係なく組み合わせて新しい個体を生成し，その中から評価関数に基づいて個体を取捨選択していくところに特徴がある．これにより，逐次改良では個体の改良において局所解に陥りやすく，求めた局所解が最良解かどうかは初期解に依存してしまうのに対して，*GA* では一部の個体が局所解に陥ったとしても別の個体と組み合わせることで，局所解から抜け出す新たな個体を生成することができる．

一方、タブー探索（*tabu search, TS*）は局所的探索手法を一般化したアルゴリズムである[3]．TS では現在の解の局所的近傍を探索し，近傍内の最良解へと遷移する．近傍内の最良解が現在の解より優れているかどうかに関係なく遷移し，さらに遷移した先の近傍内の最良解へと遷移していく．同一解への探索の繰り返しを避けるために遷移した解をタブーリストと呼ばれるリストへ書き込み，タブー解への遷移を禁止する．

2.3 シーケンスペア

GA では個体は記号の1次元系列として扱われるため，幅，高さの2次元情報であるフロアプランは1次元系列にコード化する必要がある．本研究ではモジュールの2次元配置を表すのにシーケンスペア（*Sequence-Pair, SP*）を用いる[5]．

シーケンスペアは配置の対象となるモジュール名で構成される1対の系列 (Γ_+, Γ_-) によりモジュールの配置を表現する方法である[5]．各モジュールは (Γ_+, Γ_-) のそれぞれの系列に1つずつ含まれ，この (Γ_+, Γ_-) の中のモジュールの並びにより，任意の2つのモジュールの相対位置関係が示される．

図1(a)に a〜f の6個のモジュールに対するシーケンスペアの一例を，図1(b)にこのシーケンスペアで表現されるモジュール配置を示す．図1(b)において破線の矩形が求められたチップ面積であり，その中の斜線部分がモジュールが配置されている部分である．空白部分はモジュールの配置されていない無駄なスペースとなる．また，モジュール間の相対位置の情報はシーケンスペアから図1(c)のようにオブリークグリッド上で表すことができる．

本研究では，Γ_+，Γ_- に各モジュールの向きを表す Γ_θ を加えた $(\Gamma_+, \Gamma_-, \Gamma_\theta)$ を個体表現として用いる．モジュール i の向き Γ_θ^i は 90° ずつ回転させた場合とそれぞれの表裏の場合により $\Gamma_\theta^i \in \{0, ..., 7\}$ の8種類とする．

3. 提案手法

本研究では，著者らが[6],[8]で提案した GA に基づくフロアプランニング手法を拡張し，TS を導入し，さらに並列アルゴリズム化した上で，MPI(*Message Passing Interface, MPI*)を用いてマスタ・スレーブ型のマルチコンピュータシステム上に実現する．MPI はメッセージ通信プログラムに必要なプロセッサ間の通信に関する関数群の仕様の規格である[7]．スレーブは2種類の GA と TS のいずれかを3つのフェーズに分けて実行する．各プロセスはこれらのフェーズをソフトウェアパイプライン的に実行することで徐々に解の最適化を行っていく．以下に各フェーズの概要を示す．

3.1 概　　要

(1) フェーズ 1(GA1)

各プロセスがランダムに生成した個体集合を初期解として，遺伝的アルゴリズムにより解空間の探索を行う．ただし，フェーズ1では解の厳密な評価よりも計算時間を優先し，幅広い解空間を短時間で探索を行っていく．高速実行を実現するため，目的関数は簡単化した関数を用いる．ただし，後のフェーズでの目的関数(制約条件)との整合性を考慮して，目的関数(制約)を構成する項目は省略しない．

目的関数

チップ面積と各ネットの総配線長の最小化と，さらにこれら2つの値の重みつき和の最小化を目的関数とする．ただし，高速実行を実現するために各評価値の計算は簡単化する．

制約

チップ面積のアスペクト比の考慮，モジュール配置位置に関する制約とパス遅延に関する制約を考慮する．フェーズ1では，制約は目的関数に組み込み，評価の一部として利用する．

各プロセスの動作

フェーズ1ではマスタプロセス，スレーブプロセス関係なく全てのプロセスが独立して GA を実行する．各プロセスは通信(解の交換)を行わず，それぞれがランダムに生成した初期解集合を基に GA による解の探索を行う．

規定世代 $GA1_{max}$ に達すると全プロセスで解を共有し，フェーズ2へ移行する．

(2) フェーズ 2(GA2)

フェーズ 1 で各プロセスが求めた解集合を 1 つにまとめ，その中の優良解を初期個体集合として再び各プロセスが GA を実行する．得られた解はフェーズ 1 よりも詳細に評価する．

目的関数

GA1 同様，チップ面積，各ネットの総配線長，両者の重みつき和の 3 種類を目的関数とする．各解はフェーズ 1 よりも詳細に評価する．

制約

フェーズ 1 同様，チップ面積のアスペクト比の考慮，モジュール配置位置に関する制約とパス遅延に関する制約を考慮し，目的関数に組み込んで評価の一部として利用する．

各プロセスの動作

マスタプロセスは各スレーブプロセスが生成した優良解を 1 つの解集合として保持する．常に最新の優良解集合とするため，マスタプロセスは各スレーブプロセスから定期的に優良解を回収する．マスタプロセスはスレーブプロセスからの優良解の要求に応じて，この集合から定数個の解を送信する．

スレーブプロセスはフェーズ 1 終了時に全プロセスで共有した解の中の優良解を初期解集合として，さらに GA による探索を行う．ただし，フェーズ 1 と異なり，解集合の状況 (一定期間解の更新がない等の解集合が収束した状態) の状況によっては他プロセスの優良解を組み込む．

フェーズ 1 同様，規定世代 ($GA2_{max} = GA1_{max}/2$) に達するとスレーブプロセスは終了時の優良解をマスタプロセスへ送信し，フェーズ 3 へ移行する．

(3) フェーズ 3(TS)

フェーズ 2 の優良解を入力として，各プロセスが異なる入力に対してタブー探索に基づいた局所改良を行う．フェーズ 1，フェーズ 2 とは異なり，制約を満足した上で，目的関数を最小とする解を求める事を目的とする．

目的関数

チップ面積，各ネットの総配線長の最小化を目的とする．各評価値の計算は計算時間よりも正確さを重視して計算する．

制約

フェーズ 1，フェーズ 2 同様，チップ面積のアスペクト比の考慮，モジュールの配置位置に関する制約とパス遅延に関する制約を考慮するが，フェーズ 1，フェーズ 2 とは異なり制約を満たした上で目的関数を最小とする解を求める．

各プロセスの動作

マスタプロセスは各スレーブプロセスから回収した解集合の中からタブー探索の対象となる解を選択する．対象は面積，配線長，重みつき和のそれぞれの最良解と，プロセス数に応じて異なる解を選択する．対象は各スレーブプロセスへ面積，配線長，重みつき和のどの評価値を改良するかの指示と共に送信する．

最後に，各スレーブプロセスからタブー探索によって改良された解を回収し，最終的な解を選択する．

スレーブプロセスはマスタプロセスから受信した解に対してタブー探索を実行する．改良は解と同時に受信した指示された評価値について行う．結果はマスタプロセスへ送信する．

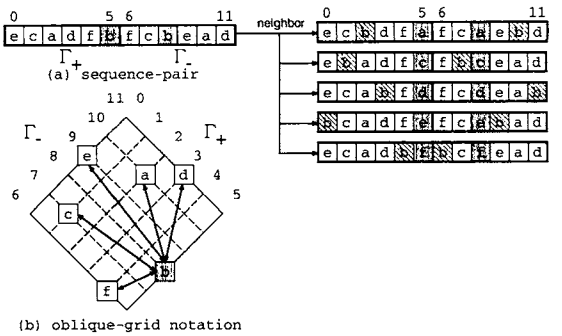

図2 近 傍 例

近傍の定義

タブー探索の近傍は「対象となるモジュールを他のモジュールと Γ_+，Γ_- 両方の出現位置を同時に交換したシーケンスペア」とした．シーケンスペアの Γ_+，Γ_- を同時に交換するため，近傍となるシーケンスペアは対象となるモジュールと他のモジュールを交換したものとなる．

図 2 にシーケンスペア $[\Gamma_+, \Gamma_-] = [(e,c,a,d,f,b), (f,c,b,e,a,d)]$ について対象となるモジュールを **b** とした場合の近傍を示す．近傍がモジュール同士の単純な交換のため，モジュール **b** と他のモジュール (a,c,d,e,f) を交換した 5 個のシーケンスペアが近傍となっている．

さらに全モジュールを対象として選択してこの操作を行い，繰り返し実行することで解の改良を行う．

3.2 解の評価

本研究では，世代 T における i 番目の個体を x_i^T(図 4) とすると，その重みつき和 $g(x_i^T)$ はシーケンスペアより求められた面積と配線長の重み付き和を用いる．

まず，個体が表すモジュール配置の面積 A_i についてはまずシーケンスペアから各モジュールの相対位置関係を求める．次に全てのモジュールの相対位置関係から水平，垂直制約グラフを作成する．例えば，図 3 は図 1 のフロアプランに対する制約グラフである．さらに制約グラフの各辺の重みをモジュールの幅，高さとして始点から終点までの最長経路を求める．最長経路上の重みの総和をチップの幅 W，高さ H として，その積から個体の面積を求める [9].

$$A_i = W \times H$$

これは各フェーズで共通である．

次に各ネットの配線長の評価は各フェーズによって異なり，フェーズが進むにつれて詳細に行っていく．

フェーズ 1 ではモジュールの中心位置を仮想端子とした半周長 $L1_n$ を配線長として近似する．フェーズ 2 ではモジュールの端子位置を考慮した半周長 $L2_n$ を配線長として近似する．フェーズ 3 では今回はフェーズ 2 と同様の値を利用するが，将来的にはスタイナ木から求めることを考えている [1].

従って総配線長 N_i は

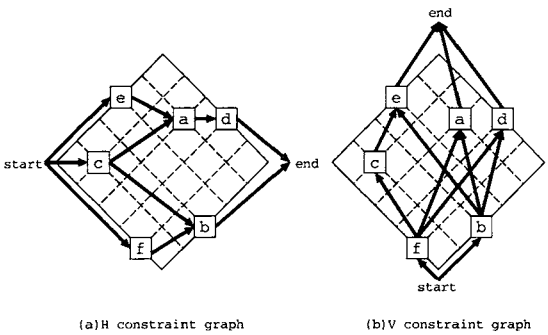

図3　水平，垂直制約グラフ

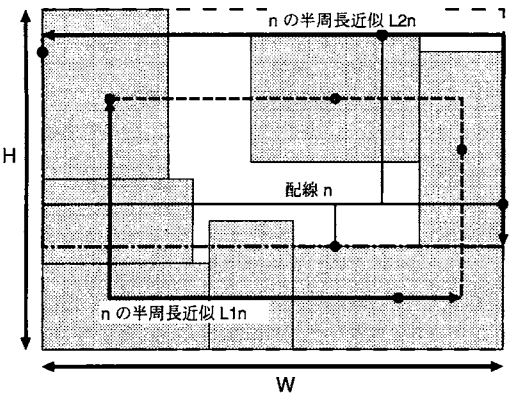

図4　フロアプランの評価

$$N_i = \sum_{i=1}^{n} L1_i, \quad \text{または} \quad N_i = \sum_{i=1}^{n} L2_i$$

から計算する．

さらに，チップ面積 A_i と総配線長 N_i から重みつき和 $g(x_i^T)$ を以下のように計算する．

$$g(x_i^T) = A_i + k \times N_i$$

k はチップ面積と総配線長の次元を合わせるための定数である．

3.3　GA の基本操作

GA で用いる基本操作 (交叉，突然変異，淘汰) は基本的には GA1，GA2 とも同じものを利用する．ただし，交叉手法に関しては GA1 ではシーケンスペアを大幅に変更し，広域探索に向いた共通トポロジ保存交叉 (*Common Topology Preserving Crossover, CTPX*) [6] を利用する．GA2 では CTPX とシーケンスペアの一部を変更し，局所探索に向いた配置依存部分交換交叉 (*Placement – based Partially Exchanging Crossover, PPEX*) [6] を組み合わせて利用する．突然変異手法には局所改良手法 *MM* [6] を利用し，淘汰手法にはパレート選択 [2] を用いる．

CTPX は両方の親に共通するモジュールの相対位置関係をできる限り多く子に残す交叉手法である．モジュールの相対位置関係は Γ_+，Γ_- の組み合わせにより決定するため，Γ_+，Γ_- に

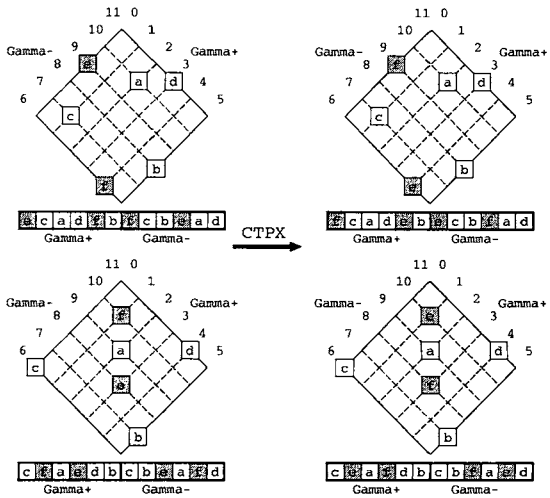

図5　CTPX の例

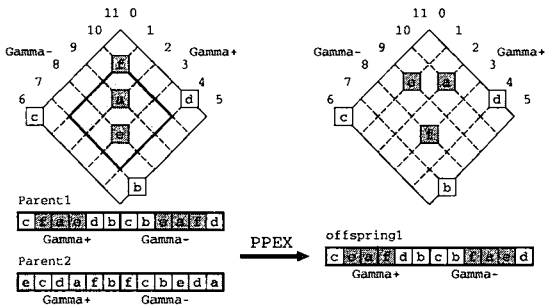

図6　PPEX の例

共通する部分系列を見つける必要がある．最長共通部分系列 (*longest common subsequence, LCS*) に含まれるモジュールは両方の親に同じ順序で含まれているため，配置上では同じ相対位置関係となる．この相対位置関係を親に共通する特徴とみなし，*LCS* に含まれるモジュールはそのまま子に受け継がせる．*LCS* に含まれないモジュールに関してはもう一方の親の出現順序に配置することで，両方の親の特徴を子に受け継がせている．

CTPX の例をシーケンスペアとオブリークグリッドを用いて図5に示す．図5では a, b, c, d の4個のモジュールは *LCS* に含まれるためにそのまま子に受け継がれ，残りの e, f のモジュールはもう片方の親の出現順序に置き換えることで新しい個体を生成している．

一方，PPEX は CTPX のようにシーケンスペア全体を変更するのではなく，オブリークグリッド上のランダムに選択したグリッドを中心に窓領域を設定し，この窓に含まれたモジュールのみを対象として交叉を行う．窓に含まれたモジュールを互いの親の出現順序で配置することで，両方の親の特徴を受け継いでいる．

PPEX の例を図6に示す．図6では parent1 に対して窓領域を決定し，窓領域内のモジュール a, e, f を parent2 での出現順序で配置し，offspring1 を生成している．

表 1　各プロセス数での面積 [mm^2](デッドスペース [%])，総配線長 [mm] とその重みつき和及び計算時間 [sec]

			プロセス数 1			プロセス数 5			プロセス数 14		
			面積	配線長	和	面積	配線長	和	面積	配線長	和
GA1+ GA2	面積最良	最良	**19.65(10.60)**	2926.11	34.28	**19.49(9.86)**	2883.54	33.91	**19.60(10.34)**	2787.18	33.53
		平均	**20.28**	2807.23	34.32	**19.79**	2892.91	34.26	**19.74**	2839.81	33.94
		最悪	**20.79(15.48)**	2804.60	34.81	**20.01(12.20)**	3045.17	35.24	**19.97(12.04)**	2874.43	34.35
	配線長最良	最良	21.09(16.70)	**2547.10**	33.83	20.41(13.91)	**2494.07**	32.88	20.79(15.50)	**2529.27**	33.44
		平均	21.97	**2703.56**	35.48	20.89	**2592.56**	33.85	20.70	**2567.52**	33.54
		最悪	21.13(16.85)	**2764.88**	34.96	20.75(15.34)	**2667.67**	34.09	20.83(15.66)	**2607.87**	33.87
	和最良	最良	20.37(13.73)	2585.69	**33.30**	20.06(12.40)	2512.24	**32.62**	19.98(12.07)	2546.83	**32.71**
		平均	20.37	2755.06	**34.15**	20.20	2607.74	**33.31**	20.02	2593.05	**32.98**
		最悪	20.83(15.67)	2761.56	**34.64**	20.93(16.05)	2578.66	**33.82**	20.16(12.86)	2592.19	**33.12**
	計算時間		3126.74			3341.60			3590.58		
GA1+ GA2+ TS	最良	平均	**18.94**	3885.64	38.36	**18.83**	3863.46	38.15	**18.76**	3897.00	38.25
		最悪	**19.11(8.06)**	3880.98	38.51	**18.98(7.45)**	3941.58	38.69	**18.80(6.53)**	3989.54	38.75
	配線長最良	最良	25.79(31.87)	**2062.82**	36.10	23.77(26.07)	**2104.78**	34.29	24.90(29.43)	**2098.70**	35.39
		平均	25.03	**2116.83**	35.61	24.50	**2123.79**	35.12	24.16	**2110.93**	34.72
		最悪	24.34(27.83)	**2137.62**	35.03	24.30(27.71)	**2139.96**	35.00	23.89(26.46)	**2124.90**	34.52
	和最良	最良	19.45(9.66)	2434.45	**31.62**	19.35(9.22)	2423.39	**31.47**	19.05(7.76)	2469.76	**31.40**
		平均	19.55	2469.48	**31.90**	19.43	2465.47	**31.76**	19.34	2456.32	**31.62**
		最悪	19.70(10.81)	2495.99	**32.18**	19.33(9.10)	2532.01	**31.99**	19.31(9.04)	2513.30	**31.88**
	計算時間		9727.15			5558.88			5946.87		

4. 実　　験

4.1 実験環境

提案手法を C 言語と MPI を用いて実装し，評価実験を行った．実験は CPU が Pentium4 3.4GHz，主記憶 1GB の PC1 台，主記憶 3GB の PC6 台と CPU3.2GHz，主記憶 2GB の PC7 台の計 14 台の PC を 1Gbit/sec のイーサネットで接続することで実現した PC クラスタ上で実験を行った．

GA の主なパラメータは GA1 の終了世代数 $GA1_{max}$ を 100000，GA2 の終了世代数 $GA2_{max}$ を $GA1_{max}$ の半分である 50000，GA の母集団の個体数を 25，交叉・突然変異から生成する個体数を 100，重みつき和の総配線長に対する重み k を 0.005 とした．GA2 ではマスタの優良解の保持数を 100，スレーブのマスタへの優良解の定期送信間隔を 200 世代，解集合の収束条件を目的関数である面積，総配線長，両者の重みつき和の 3 種類の最良解の更新が 300 世代ないこと，とした．スレーブの GA に関わるパラメータは GA1 に関しては乱数のシード以外全て共通，GA2 に関しては交叉手法である PPEX と CTPX の選択確率をプロセスごとに異なるものにしている．

評価値計算では制約はないものとし，チップ面積は GA1，GA2 共通に計算し，総配線長は GA1 ではモジュールの中心を端子位置と仮定した半周長近似から計算し，GA2，TS ではモジュールの向き，端子位置を考慮した上で半周長近似から計算した．

4.2 実験結果と考察

実験データは GSRC ベンチマークデータ [10] のモジュール数 200，ネット数 1274 を利用した．実験結果を表 1 に示す．ただし，表には求めた 5 回の試行で求めたパレート解の内の面積最良，総配線長最良，重みつき和最良のそれぞれの解の最良解と平均値を示している．

プロセス数 1 の実験結果は比較対象として，GA1，GA2 共に通信処理を行わず逐次処理した結果である．フェーズ 3 では面積・配線長・重みつき和それぞれの最良解へのタブー探索の適用を逐次的に実行するため，複数プロセスの場合の約 3 倍の計算時間がかかっている．

プロセス数 5 のフェーズ 3 では面積・配線長・重みつき和それぞれの最良解についてタブー探索を実行し，プロセス数 14 では各評価値の上位 4 種類の解にタブー探索を適用している．

まず，逐次処理となるプロセス数 1 の結果との比較から，並列化により面積，配線長，重みつき和のどの評価も並列化した方が優れていることがわかる．これは単純にプロセス数 5 ではプロセス数 1 の場合の 5 倍の解の評価を行い，解の探索範囲が広がっていることが挙げられる．

計算時間についてもプロセス数 1 とプロセス数 5 では同程度である．これは並列処理の通信にかかるオーバヘッドはあるものの，通信回数が少なくマスタプロセスが処理できる範囲でしか各スレーブからのデータの送信が起きないためだと考えられる．

ただしさらにプロセス数を増やした 14 台の場合では評価はプロセス数 5 の結果とほとんど変わらない．これはプロセス数 1 の 14 倍の数の解を評価しているとはいえ，各プロセスの GA2 の初期解が共有した解集合の優良解からランダムに定数個選択した解集合とまったく同じではないが似かよっており，その結果プロセス数 5 の場合と比べて大幅な改良は行われなかったと予想される．従って，今後各スレーブプロセスの GA のパラメータを差別化することで，プロセス数の増加に比例して解

表2 プロセス数5におけるGA単体での面積 [mm^2](デッドスペース [%]), 総配線長 [mm] とその重みつき和及び計算時間 [sec]

		面積	配線長	和
面積最良	最良	**19.44(9.64)**	2646.64	32.68
	平均	**19.68**	2592.40	32.64
	最悪	**19.93(11.83)**	2571.87	32.79
配線長最良	最良	20.43(13.98)	**2432.99**	32.59
	平均	21.01	**2529.37**	33.66
	最悪	21.25(17.31)	**2602.57**	34.26
和最良	最良	19.79(11.23)	2456.69	**32.08**
	平均	19.74	2562.39	**32.55**
	最悪	19.67(10.68)	2642.40	**32.88**
計算時間		5082.20		

表3 プロセス数5におけるTS単体での面積 [mm^2](デッドスペース [%]), 総配線長 [mm] とその重みつき和及び計算時間 [sec]

		面積	配線長	和
面積最良	最良	**19.90(11.70)**	3804.86	38.92
	平均	**20.55**	3865.23	39.88
	最悪	**21.15(16.91)**	3805.52	40.17
配線長最良	最良	29.05(39.51)	**2155.76**	39.82
	平均	29.55	**2198.48**	40.54
	最悪	31.57(44.35)	**2231.40**	42.73
和最良	最良	21.03(16.47)	2504.86	**33.56**
	平均	20.95	2560.61	**33.92**
	最悪	21.55(18.49)	2587.82	**34.49**
計算時間		5540.14		

の改良も行うことができるではないかと考えている.

計算時間についてはプロセス数が少ない場合と比較して多少増加している. これはプロセス数5の場合ではGA2での通信回数が少ないため, 各スレーブプロセスからの優良解の送信をマスタプロセスがスレーブプロセスが待ちの状態にならない範囲で処理できるが, プロセス数が14と増えるとスレーブプロセスからの優良解の送信が一斉に起きてしまい, マスタプロセスが処理しきれずボトルネックの状態になり, マスタの処理が終わるまでスレーブプロセスが処理を中断するような状況に陥っていると考えられる. 対策としては各スレーブプロセスからの優良解の送信のタイミングをずらし, マスタプロセスが処理できる範囲に抑えることを考えている.

次にフェーズ3であるTSではGAの結果と比較すると, どの評価値もプロセス数に関係なく優れていることがわかる. これはGAでの解の改良が複数のシーケンスペアの比較から行う改良であるため, モジュールの交換といった局所的な操作を行わずに改良していくためだと考えられる.

比較対象として表2に示すフェーズ3による近傍探索を行わず, GAの世代数を増やしGA+TSと同程度の計算時間になるように設定した場合との実験結果を示す. この結果からフェーズ1+2となるGAではある程度収束した状態であり, フェーズ3の近傍探索が有効であると考えられる.

逆に, 提案手法のフェーズ3であるTS単体で, 提案手法全体(フェーズ1~3)と同程度の個数の解を生成・評価した場合の実験結果を表3に示す. この結果よりTS単体ではGAとTSを組み合わせた場合より優れた解を求める事はできないことがわかり, 両者を組み合わせることが有効であると考えられる.

ただし, プロセス数1の場合とプロセス数5の場合ではGAでのフェーズの結果と異なり, 両者の解はほとんど同じものとなっている. これは, TSにおいては近傍が単純なモジュールの交換であり, かなりの回数の繰り返しによって最終的な配置がGAの解を反映しつつも, 似かよってしまうのではないかと予想される.

また, プロセス数14の場合でもプロセス数が少ない場合に比べ多くの解に対してTSを実行しているが, 最終的に得られる解は同程度の解となっている. これは近傍探索手法が等しいため, 対象が多くても似たような解しか得られないためと考えられる. よって今後異なる近傍を定義し, 1つの解に対して複数の異なる近傍を探索することで別の局所解を発見し, 現在の局所近傍解との比較からよりよい解を求める必要がある.

5. おわりに

本稿では, LSIフロアプランニング問題に対して遺伝的アルゴリズムとタブー探索法を組み合わせた並列フロアプランニング手法を提案した. 提案手法をPCクラスタ上に実装して実験的評価を行い, 提案手法の有効性を示した. 今後の課題として, 各スレーブのパラメータを個別に設定することで解空間のより有効な探索を実現すること, フェーズ3(TS)について異なる定義の近傍の実装とその実験的評価, 各種のフロアプラン制約の実装等が挙げられる.

文　献

[1] C.Chu, Y.-C.Wong: "Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design," *Proc.ISPD 2005*, pp.28-35, 2005.

[2] Kalyanmoy Deb, Samir Arawal, Amrit Pratap, T. Meyarivan: "A fast elitist non-dodminated sorting genetic algorithm for multi-objective optimization : NSGA-II," *Parallel Problem Solving from Nature–PPSN VI, Proc. 6th International Conference 2000*, pp.849-858, 2000.

[3] Fred Glover, Manuel Laguna: *Tabu Search*, Kluwer Academic Publishers, 1997.

[4] David E. Goldberg: *Genetic Algorithm in Search, Optimnization and Machine Learning*, Addison-Wesley, 1989.

[5] H. Murata, S. Nakatake, K. Fujiyoshi, and Y. Kajitani: "VLSI module placement based on rectangle-packing by the sequence-pair," *IEEE Trans. on Computer-Aided Design of Integrated Circuit and Systems*, Vol. 15, No.12, pp. 1518-1524, Dec. 1996.

[6] 中矢真吾, 小出哲士, 若林真一: "適応的遺伝的アルゴリズムに基づくVLSIフロアプランニングの一手法", 情報処理学会論文誌, Vol.43, No.5, pp.1361-1371, 2002.

[7] P. S. Pacheco 著 (秋葉博訳): MPI 並列プログラミング, 培風館, 2001.

[8] Takayoshi Shimazu, Shin'ichi Wakabayashi, Shinobu Nagayama: "A parallel implementation of a genetic algorithm-based floorplanninng method on PC clusters," *Proc.SASIMI2006*, pp.404-411, 2006.

[9] Xioping Tang, Ruiqi Tian, D.F.Wong: "Fast evaluation of sequence pair in block placement by longest common subsequence computation, " *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.20, No.12, pp.1406-1413, 2001.

[10] http://www.cse.ucsc.edu/research/surf/GSRC/bench1.html