NAIST-IS-DD0461017

博士論文

遺伝的アルゴリズムのハードウェア化のための アーキテクチャと回路規模予測モデル

橘 達弘

2007年2月1日

奈良先端科学技術大学院大学
情報科学研究科 情報処理学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士 (工学) 授与の要件として提出した博士論文である。

橘 達弘

審査委員：

伊藤 実 教授　　（主指導教員）
藤原 秀雄 教授　　（副指導教員）
安本 慶一 助教授　（副指導教員）

遺伝的アルゴリズムのハードウェア化のための アーキテクチャと回路規模予測モデル*

橘 達弘

内容梗概

組合せ最適化問題を解くための手法として，単一目的遺伝的アルゴリズム（Single Objective Genetic Algorithm，SOGA），SOGA を多目的最適化問題に拡張した多目的遺伝的アルゴリズム（Multi-Objective Genetic Algorithm，MOGA）がある．一般に，GA は計算量が大きいため，高速に実行させるための手法が求められ，その 1 つとしてハードウェア化に関する研究がされている．また，組込みシステムへの搭載を目的に，Field Programmable Gate Array (FPGA) 上に GA チップとして実装できることが望まれている．GA を FPGA 上に実装するためには，ハードウェア記述言語（Hardware Description Language，HDL）などで記述された GA を論理合成し，GA のゲートレベル論理回路（以下，GA 回路）を作成する．そして，この GA 回路を FPGA 上に実装する．組込みシステムへ GA チップを搭載するためには，必要となる性能，消費電力，問題サイズなどの要求を満たす GA 回路が必要となる．そこで，本論文では，FPGA 上に適切な SOGA 回路を実装する手法とその設計を支援するための手法を提案する．また，MOGA のハードウェア実装に適したアーキテクチャおよび実装手法を提案する．

SOGA を FPGA 上に実装する際には，(1) 作成した SOGA 回路記述が実装先の FPGA 上に実装可能であるかどうかが論理合成を行うまで不明である，(2) 与えられた問題やパラメタが異なるたびに再設計を行う必要があるという問題がある．問題 (1) を解決するためには，SOGA 回路の回路規模を論理合成を行うことな

* 奈良先端科学技術大学院大学 情報科学研究科 情報処理学専攻 博士論文, NAIST-IS-DD0461017, 2007 年 2 月 1 日.

く予測する手法が，問題（2）を解決するためには，様々な SOGA 回路を作成するための設計手法（アーキテクチャ）がそれぞれ必要である．そのため，本研究では，SOGA 回路を設計するためのアーキテクチャ（SOGA アーキテクチャ）とこのアーキテクチャにより作成した SOGA 回路の回路規模を予測するための方法（回路規模予測モデル）を考案した．SOGA アーキテクチャは，世代交代モデルとしてハードウェア化に適した Minimal Generation Gap（MGG）モデルを採用している．提案アーキテクチャでは，SOGA 回路は，基本となるモジュールを組合せて構成され，モジュールを組替えることで様々な問題に適用可能であり，並列化が容易であるという特徴を持つ．また，回路規模予測モデルを用い，HDL の 1 つである VHDL で記述された SOGA 回路記述を出力するツールも作成した．このツールは与えられたパラメタ（実装するデバイス，対象問題，個体数，並列度など）から適切な SOGA 回路記述を出力する．

次に，MOGA 回路のハードウェア実装に適したアーキテクチャ（MOGA アーキテクチャ）を，SOGA アーキテクチャを拡張することで実現した．MOGA では，複数の解を同時に探索するため，多様性を保つ機構が必要である．しかし，既存のニッチ法やランク戦略などの多様性確保のための機構をパイプライン処理で実装することは困難である．そこで，パイプライン処理に適した多様性を維持するための新しい手法を設計し，MOGA アーキテクチャに採用した．

提案アーキテクチャによる GA 回路の性能を確認するため，それぞれソフトウェアで実装した一般的な SOGA および MOGA との比較を行った結果，提案手法で実装された回路は，最大約 3,270 倍優れた性能を達成することを確認した．最後に，回路規模予測モデルの有効性を確認するために，提案アーキテクチャにより作成した SOGA 回路の回路規模の予測値と実測値を比較した．その結果，回路規模予測モデルによる予測値と実測値の誤差は最大 3%に収まることを確認した．

キーワード

遺伝的アルゴリズム，多目的遺伝的アルゴリズム，FPGA，実装手法，回路規模予測モデル

Architecture for Hardware Implementation of Genetic Algorithm and Prediction Model for Circuit Size*

Tatsuhiro Tachibana

Abstract

Single Objective Genetic Algorithms (SOGAs) are approximation methods to solve combinatorial optimization problem. Multi-Objective Genetic Algorithms (MOGAs) are enhancement of SOGAs to solve multi-objective optimization problem. Since GAs require high computation power, there is a demand to execute GAs at high speed. For this purpose, some research efforts have been made to implement GAs on hardware platform. Also, in order to use GAs in embedded system, it is required to implement GAs on Field Programmable Gate Array (FPGA) as GA chips. To do so, GAs must be described in Hardware Description Language (HDL) so that gate level logic circuits (called GA circuits) are synthesized using synthesis tools. In order to implement GAs on FPGA, GA circuits need to satisfy constraints (performance, power consumption, problem size, etc). So, in this thesis, we propose a method to implement SOGA on FPGA adequately and the tools to support the design of SOGA circuits. We also propose an architecture and hardware implementation method for MOGAs.

In order to implement SOGAs on FPGA, there are the following two problems. Problem (1) is that it is not known whether generated SOGA circuit can fit on the target FPGA device until synthesizing it. Problem (2) is that it is necessary to redesign SOGA circuits when different problem and different parameters are given. In order to solve problem (1), it is necessary to predict the SOGA circuit size without synthesizing it. In order to solve problem (2), we need an architecture which is suitable to

* Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0461017, February 1, 2007.

implement various SOGA circuits. Thus, we developed an architecture to design various circuits (called SOGA architecture) and a method to predict SOGA circuit size (called prediction model). In proposed architecture for SOGA, Minimal Generation Gap (MGG) model which is suitable for hardware implementation is adopted. SOGA circuits based on proposed architecture consists of several basic modules so that it can be applied to various problems by changing modules and designers can easily increase the number of concurrent pipelines. Based on our architecture and prediction model, we have developed a tool to generate GA circuits written in VHDL (which is one of HDL). This tool outputs register transfer level VHDL description of SOGA circuit from given parameters (target device, problem type, the number of solution candidates, the number of concurrent pipelines, etc).

Next, we propose an architecture for hardware implementation of MOGA (called MOGA architecture). MOGA architecture is enhancement of SOGA architecture. MOGAs require a mechanism to preserve diversity of individuals. However it is difficult to implement existing methods (which are niching methods, ranking strategy, etc) as pipelined circuits. Thus, in order to keep diversity of solution candidates, we developed a new mechanism which can be implemented as pipelined circuits and adopted it in MOGA architecture.

For evaluation, we compared performance of GA circuits developed with our methods with software SOGA and software MOGA, respectively. As a result, we confirmed that our GA circuits have up to 3,270 times higher search efficiency than software SOGAs. Finally, we compared the predicted circuit size with actual circuit size to evaluate accuracy of our prediction model. As a result, we confirmed that our prediction model can predict the actual circuit sizes within 3% error.

Keywords:

genetic algorithm, multi-objective genetic algorithm, FPGA, hardware implementation method, prediction model for circuit size

目 次

図 目 次

表 目 次

1. 緒論

本論文は，筆者が奈良先端科学技術大学院大学情報科学研究科に在学中に行った遺伝的アルゴリズム（Genetic Algorithm，GA）のハードウェア化に関する研究をまとめたものである [18, 19, 27, 20].

遺伝的アルゴリズム (Genetic Algorithm，以下 GA)[13] は組合せ最適化問題の近似解を得る手法である．GA は，生物の進化を模倣した最適化手法で，解候補を個体上の染色体に持たせ，複数の個体を用いた多点探索を行う．本論文では単一目的の組合せ最適化問題を解く GA を単一目的遺伝的アルゴリズム（Single Objective Genetic Algorithm，以下 SOGA）と呼ぶ.

SOGA はほとんどの単一目的の組合せ最適化問題に適用でき実装も容易であるため，様々なアプリケーションで用いられている．しかし，SOGA は専用のアルゴリズムと比べて計算量が大きいため，リソースの限られた機器での使用は制限があった.

SOGA の携帯端末上でのアプリケーションとして，巡回セールスマン問題 (以下 TSP) に時間制約などを加え，観光向けのパーソナルナビゲーションに応用したもの [22] や，SOGA を用いた，高速かつ高品質な画像の圧縮 [10] 等がある．また，ネットワークにおいてビデオなどのマルチメディアデータを実時間配信するための最適コストのマルチキャスト配送木を高速に算出する手法 [12] も提案されており，ルータ上でこのようなアルゴリズムを実行することができれば便利である．これらのアプリケーションを，携帯端末，ルータ，あるいは FAX/AV 機器などに実装するためには，SOGA を高速，低消費電力，かつ安価に実現するための実装技術が必要である．また，SOGA を実行する大規模な並列計算機システムを実現する試み [5] が行われているが，このようなシステムの拡張性を高めるためには，各処理ユニットの高速化，小型化，低消費電力化が必須である．これらの目的のため，SOGA のハードウェア実装であるハードウェア GA に関する研究がなされている [25, 31, 30, 15, 4, 27].

また，組込みシステムへの搭載には，Field Programmable Gate Array (FPGA) 上に GA チップとして実装できることが望まれている．GA を FPGA 上に実装するためには，ハードウェア記述言語（Hardware Description Language，HDL）など

で記述されたGAを論理合成し，GAのゲートレベル論理回路（以下，GA回路）を作成する．そして，このGA回路をFPGA上に実装する．組込みシステムへGAチップを搭載するためには，必要となる性能，消費電力，問題サイズなどの要求を満たすGA回路が必要となる．本論文では，まず，FPGA上に適切なSOGA回路のための設計手法（SOGAアーキテクチャ）とその実装手法とその設計を支援するための手法を提案する．

また，GAには，SOGAを多目的最適化問題（Multi-Objective Optimization Problem）に適用できるように拡張した多目的遺伝的アルゴリズム（Multi-Objective Genetic Algorithm，以下MOGA) もある．多目的最適化問題とは，複数の異なる目的に対し，パレート最適解と呼ばれる複数の最適解の集合を探索する問題である．多目的最適化問題は，飛行機，エンジン等の設計の際に，相反する要求を満たす必要があるパラメタを求める問題として用いられる [28, 29, 24]．MOGAは，複数のパレート最適解を同時に探索する必要があるため，SOGAよりさらに計算量が大きい．そこで，SOGAと同様にMOGAをハードウェア化することで高速化を行なうことが考えられる．しかし，MOGAをハードウェア化した研究は筆者らの知る限り存在しない．本論文では，SOGAアーキテクチャを拡張することで，MOGAのハードウェア実装に適したアーキテクチャ（MOGAアーキテクチャ）および実装手法を提案する．

本論文では，2章でSOGAをFPGA上に実装する回路とその設計を支援するための手法について述べる．SOGAをFPGA上に実装する際には,（1）作成したSOGA回路記述が実装先のFPGA上に実装可能であるかどうかが論理合成を行うまで不明である,（2）与えられた問題やパラメタが異なるたびに設計を行う必要があるという問題がある．問題（1）を解決するためには，SOGA回路の回路規模を論理合成を行うことなく予測する手法が，問題（2）を解決するためには，様々なSOGA回路を作成するための設計手法（アーキテクチャ）がそれぞれ必要である．

これらの問題を解決するため，本研究では，SOGA回路を設計するためのアーキテクチャ（SOGAアーキテクチャ）と本アーキテクチャに従い作成したSOGA回路の回路規模を予測する手法を提案する．本論文で提案するアーキテクチャは，

GA オペレータだけではなく，評価関数も含めてハードウェアで実装することで，計算量の大きい評価関数の処理時間も短縮し，効率の良い実装を可能にしている．提案アーキテクチャは，基本となるモジュールの組合せで構成され，モジュールを組替えることで，様々な問題に適用が可能である．また，並列化の拡張性に優れた並列型 GA の一種である島モデル型 GA(Island GA，IGA) を構成することが可能であり，使用可能な回路規模が大きい場合に，これを十分に活かして高い探索性能を得ることが可能である．また，並列度が高くなってもクリティカルパスが長くなりにくい特徴がある．

次に問題（1）を解決し，指定した FPGA 上への柔軟な実装を可能にするために，SOGA で用いる個体数，並列度等から，FPGA 上に実現される回路規模を予測する手法を提案する．これにより，回路を合成することなく，性能および最終的な回路規模を予測することが可能になる．性能とコストのトレードオフを考慮し，指定した FPGA 上にその回路規模の範囲内で，並列 SOGA 回路を実装することができ，ラピッドプロトタイピングに役立つ．提案手法による SOGA 回路の実装を容易にするため，回路規模の予測結果から，実装可能なデバイス及び並列度を探索する実装判定ツール，と実装判定ツールを用いて得られたパラメタ値から HDL の 1 つである VHDL で記述された SOGA 回路記述を導出するツールを実装した．本論文では，提案したアーキテクチャを用いて，問題サイズの異なるナップサック問題 (Knapsack Problem) と TSP に対するハードウェア GA を実装することで本アーキテクチャの問題サイズに対する汎用性を示す．また，提案手法の有効性を示すために，それぞれの回路規模予測モデルを導出し，実際に合成された回路と予測を比較を行った．結果，回路規模予測モデルによる予測値と実測値の誤差は最大約 3%以内に収まることを確認した．また，提案アーキテクチャを用いて作成した回路が，ナップサック問題，TSP ともに，優れた性能を示し，Pentium4 （2.4GHz）の最大約 1/80 の低消費電力で実行可能であることをゲートレベルのシミュレーションにより確認した．

次に 3 章において，MOGA 回路のハードウェア実装に適したアーキテクチャおよび実装手法を提案する．MOGA 回路のハードウェア実装に適したアーキテクチャ（MOGA アーキテクチャ）を，SOGA アーキテクチャを拡張することで実

現した．MOGA では複数のパレート最適解を効率良く探索するために，解の多様性を維持するニッチ法 [16] やランク戦略 [7, 21] 等の機構が必要となる．しかし，ニッチ法やランク戦略は全ての解候補（個体）間で比較を行うためにソートなどの繰り返しを伴う操作を必要とし，これらの既存の機構をパイプライン処理で実装することは困難である．そこで，パイプライン処理に適した多様性を維持するための新しい手法を設計し，MOGA アーキテクチャに採用した．また，提案アーキテクチャでは解探索能力の向上と回路のスケーラビリティの確保のために，島モデル型 GA を拡張し，島ごとに異なる部分のパレート最適解を探索させる MOGA に適した並列化手法を採用する．提案アーキテクチャによる MOGA 回路の有効性を示すために，ソフトウェア上に実装した代表的な MOGA の 1 つである NSGA-II [7] を Pentium4 (2.4GHz) 上で動作させたものとの性能比較を行った．提案アーキテクチャによる MOGA 回路は VHDL で記述し，Quartus II を用いて論理合成を行った．対象問題には多目的ナップサック問題を適用した．実験の結果より，提案手法が NSGA-II より最大約 3,270 倍優れた解探索能力を持つことを確認した．

2. SOGA を FPGA 上に実装する回路とその設計手法

2.1 はじめに

本章では，与えられた問題とその問題サイズ，実装先 FPGA の書き込み可能な回路規模から適切な回路を構築するための手法を提案する．そのために，まず様々な SOGA のアプリケーションをハードウェアで実装するためのアーキテクチャを提案する．

次に，指定した FPGA 上への柔軟な実装を可能にするために，SOGA で用いる個体数，並列度等から，FPGA 上に実現される回路規模を予測する手法を提案する．これにより，回路を合成することなく，性能および最終的な回路規模を予測することが可能になる．性能とコストのトレードオフを考慮し，指定した FPGA 上にその回路規模の範囲内で，並列 SOGA を実装することができ，ラピッドプロトタイピングに役立つ．提案手法による SOGA 回路の実装を容易にするため，回路規模の予測結果から，実装可能なデバイス及び並列度を探索する実装判定ツール，と実装判定ツールを用いて得られたパラメタ値から VHDL で記述された SOGA 回路を導出するツールを実装した．

2.1.1 関連研究

SOGA のハードウェアの実装に関して，いくつかの研究がなされている．[30] では H^3 エンジンと呼ばれる遺伝的アルゴリズム専用ハードウェアを実装している．[25] では, 交叉法に適応選択を組み込んだ SOGA をハードウェア化している．[31] では, 不連続閉曲線抽出手法を対象としたハードウェア GA を適用している．[9] では TSP を対象にハードウェア GA を実装している．[15] では，比較的容易な問題である Set Coverage Problem に適用している．その他，[4] を含む既存の手法の多くは，それぞれの問題に特化したアーキテクチャを使用している．そのため，対象外の問題を適用する際の符号化した染色体をどのように扱い，どの様に送信するかの仕様が定義されておらず，また，交叉，突然変異などの遺伝的操作をハードウェア化した際の入力と出力の仕様も定義されていない．結果として特定のアーキテクチャを他の問題にそのまま適用することは困難である．

様々な用途に応じたSOGA回路を情報機器や小型の電子装置上に実装するためには，汎用的なSOGA回路のアーキテクチャが必要である．また，実装するFPGAの種類やSOGAの対象となる問題ごとに発生する様々な制約条件(性能，コスト，消費電力)を満足する回路を作成するための手法が必要となる．このような，設計の補助を行なう手法として，[11]では，リアルタイムの組込み型システムの効率的な開発のための手法が提案されている．しかし，この手法は，SOGA回路の開発に特化するものではない．

2.2 GA のハードウェア化の基本アイデア

この節では，SOGA について簡潔に説明を行ない，その後，本論文で用いた SOGA の回路化手法の概要について述べる．また，提案手法で採用する，ハードウェア化に適していると思われる既存の幾つかの SOGA アーキテクチャについて述べる．

2.2.1 単一目的遺伝的アルゴリズム

SOGA では，与えられた問題の探索空間内の一点 (解候補) を表現する**染色体** (chromosome) と**適応度**(どれだけ最適解に近いかを表す数値，fitness) を含む**個体**を複数用いて近似解を求める．

SOGA の動作の概要は以下のようになる．(1) 多数のランダムな個体を作成する．(2) 各個体の適応度を計算する．(3) 良い適応度を持つ個体群を選択し，残りを破棄する．(4) 選択された個体のペアを両親 (parent) とし，両親の性質を混ぜ合わせて新しい個体 (offspring) を作成する (**交叉**)．(5) 新しい個体の染色体に対しある確率で突然変異を起こす．(6) 終了条件を満たせば，終了し，満さなければステップ (2) に戻る．

SOGA では，これらの操作を繰り返し適用し準最適解を探索する．また，終了条件は，個体の評価回数が一定回数に達するか，または，一定回数評価を行った際に，新たに良い解が見つけられなかった場合などが用いられる．本論文では，個体の評価回数が一定回数に到達した場合，SOGA を終了するものとする．

染色体を構成している要素を**遺伝子**，その遺伝子の書かれている染色体内の位置を**遺伝子座**と呼ぶ．

2.2.2 SOGA の回路化手法の概要

提案手法の目的は，与えられた問題とその問題サイズ，実装先 FPGA の書き込み可能な回路規模から適切な回路を合成することである．FPGA は，ロジックエレメント数と利用できるメモリブロックが限られているため，これらを考慮して回路を合成する必要がある．目的を達成するために，汎用的なアーキテクチャを

提案する．提案するアーキテクチャは以下の特徴を持つ．(1) ハードウェア化に適しており様々な問題に適用可能である．(2) 利用可能な回路規模内で，できるだけ SOGA 回路の並列度を増やすことができる．(3) 与えられた問題とそのサイズから回路規模の予測を行う．

(1) を実現するために，Minimal Generation Gap(MGG) モデル [12] を基礎とした世代交代モデルを採用し，モジュール単位での設計を行なう．この MGG モデルはハードウェア化に適しており，必要となるメモリとレジスタを削減することができる．世代交代モデルについては，2.2.3 節で説明する．また，モジュール単位での設計を行なうことで様々な問題に適用が可能となる．

(2) を実現するために，複数の並行に実行されるパイプライン間での情報の同期頻度を少なくする必要がある．並列実行を行なうアーキテクチャとして島モデル型 GA(Island GA，IGA) を採用する．並列実行については，2.2.4 節で説明する．

(3) を実現するために，問題の種類，問題のサイズ，解候補の数，そして，並列度数より回路規模を予測する手法を開発した．詳細は，2.4 節に示す．

2.2.3 提案手法における世代交代モデル（SOGA）

従来の世代交代モデルでは，選択操作の際に親個体と同数の新しく生成された子個体を必要とする（図 1 参照）．このような従来の世代交代モデルを用いる一般的な GA を本論文では simple GA （SGA）と呼ぶ．この従来の世代交代モデルを用いてハードウェア化すると，新しく生成された個体群を全て保持する必要があるため回路規模が増大する．そのため，文献 [15]，[4] では，survival-based, steady-state GA と Compact Genetic Algorithm を用いることで回路規模とメモリ量の増加を抑制している．しかし，survival-based，steady-state GA は，常に個体群の中で適応度が最も悪い個体を把握する必要がある．特にメモリを用いて個体群を保持する場合，最も適応度が悪い個体を探索するのに時間がかかり，パイプライン処理が困難となる．Compact Genetic Algorithm は，個体群を保持せず，代りに染色体の集合を確率分布として保持する手法である．この手法は，各遺伝子を 0 もしくは 1 で表現する bit string 形式の符号化手法のみに使用でき，それ以外の符号化手法を用いる問題では適用が困難である．

また，従来の世代交代モデルでは新たな個体群が生成されるまで選択操作を行うことができないため効率的なパイプライン処理が困難である．この問題を解決するため，提案アーキテクチャでは，MGG モデル [14] を基としたハードウェア化への実装に適した世代交代モデルを採用する（図 2 参照）．

この世代交代モデルでは，交叉・突然変異操作の際，個体群中より選択された 2 つの親個体より 1 つの子個体のみ作成する．次に生成された子個体を評価する．新しく生成した染色体 1 つと親の染色体 2 つの中から，2 つの最良個体を選択する．本モデルは，文献 [23] で採用されているモデルと同様であるが，2 つの個体に交叉，突然変異を適用し， 複数の染色体を生成し，親の染色体と生成された染色体間で選択を行なう点で異なる．提案アーキテクチャでは，3 つの個体のみ保持すれば良いため回路規模を抑えることができ，パイプライン処理による実装が容易である．

提案アーキテクチャでは，親個体を子個体で入れ換え（上書き）の際，個体群を保持しているメモリ上への書き込みが必要である．またパイプライン処理のため，絶えず親個体を読み出して送り続ける必要がある．しかし機構の簡単化のため，メモリ上において個体の上書きと読み出しの処理を同時に行うことができない．そこで提案手法では，子個体による上書きが行われる際には，メモリ上からではなく上書きに用いた個体をメモリから読み出された子個体として扱う．このときメモリには上書きのみを行う．上書きが行われない場合には，メモリ上からランダムな個体を読み出す．

2.2.4 島モデル型 GA の概要

GA の並列化には様々な手法が存在する．本論文では，並列 GA の 1 つである島モデル型 GA(Island GA，IGA)[5] を採用する．IGA は，解候補である個体群を複数の集合に分割する．これらの分割された個体群の集合を，IGA では島と見なし，これらの島で解候補を独自に進化させる．個体群全体での情報の共有を行なうために，**移民** (migration) と呼ばれる操作を行なう．移民は，一定間隔ごとに島間で個体を移動させる操作である．本論文では，個体を移動させる間隔を移民率と呼ぶ．IGA は，SGA より多様性が確保されやすいため，局所解に陥りにくく，

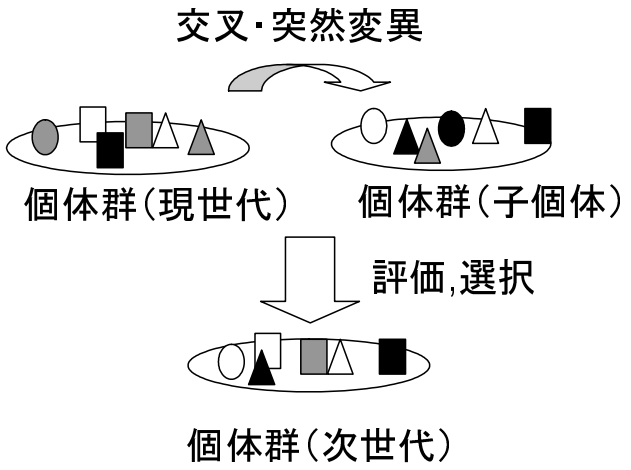

図 1 従来の世代交代モデル

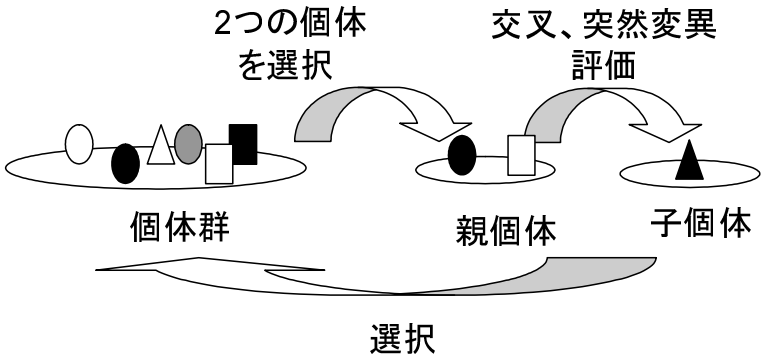

図 2 提案手法の世代交代モデル

性能が改善される．回路の並列化において，クリティカルパスが長くなることで，最大動作周波数が減少することがある．IGA は，隣り合う島間で個体の移民を行なえばよいため，クリティカルパスがそれほど長くならず，並列化による最大動作周波数の減少が抑えられる．

2.3 共通アーキテクチャ

本節では様々な問題に適用可能であり，回路規模が予測可能で，並列化が容易である SOGA 回路向けアーキテクチャについて述べる．提案アーキテクチャによる SOGA 回路は，遺伝的操作に対応した複数のモジュールから構成される．提案アーキテクチャでは，モジュール間でのパイプライン化に適した情報の伝達手順や各モジュールの入出力仕様を定義する．これを行なうことで，モジュール単位で様々な種類の交叉，突然変異及び，評価関数をハードウェアで実装することができ，それら組替えることで様々な問題に適用可能となっている．また，同時に提案アーキテクチャは問題に依存せず，かつ，拡張性に優れた並列化手法も定義している．

ここでは，最小単位の SOGA 回路の構築手法である基本アーキテクチャと並列化を行なう手法である並列アーキテクチャについて述べ，最後に各モジュールの処理時間が一定で無い場合の対処について述べる．

2.3.1 基本アーキテクチャ

基本アーキテクチャは，図 3 に示す 4 つのサブモジュール：管理モジュール (management module)，交叉モジュール (crossover module)，突然変異モジュール (mutation module)，評価モジュール (evaluation module) から構成される．管理モジュールは，個体の染色体と適応度を保持する．交叉，突然変異，評価の各モジュールでは，それぞれ対応する遺伝的操作を行う．選択は，管理モジュールと交叉モジュールで行われる．

また，GA では個体数，交叉率，突然変異率，移民率などのパラメタにより解探索能力が変化することが知られている [17]．本論文では，これらのパラメタのうち個体数のみは，設計の際に与えられた値を使用する．それ以外のパラメタは，それぞれの値を回路の動作時に外部より与えられ，それをレジスタで保持し，用いる．各モジュールで保持されたパラメタは，動的に外部より与えられ，変化させることができる．

各個体の染色体は，n ビットの 2 進数で表される．モジュール間のバス幅は一

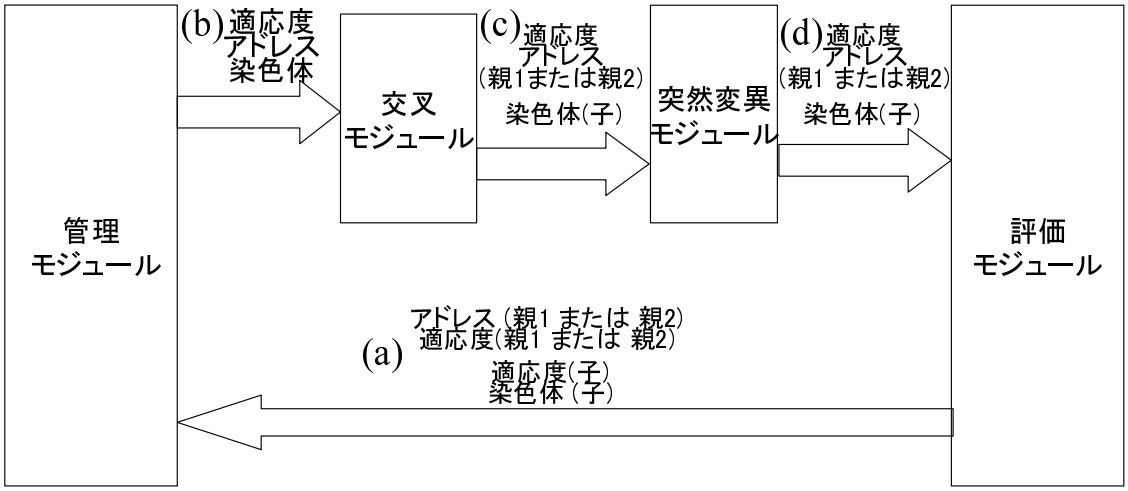

図 3 基本アーキテクチャ（SOGA）

定であり，m ビットとする．n, m の値はパラメタとして与えることができる. 各モジュールは，原則として各クロックごとに m ビットのデータを受け取り，c クロック後に m ビットの演算結果を出力できるように設計される (ここで c は定数). また，データ処理中も，各クロックごとに新たなデータを受け取れるよう設計されている．すなわち，$n \geq m$ の場合，$\lceil \frac{n}{m} \rceil$ クロックが一染色体辺りのスループットとなる．各モジュールは，順番にデータを受信しつつパイプライン的にデータを逐次処理する.

以下，各モジュールの詳細について述べる.

管理モジュール

管理モジュールは個体群を保持するメモリを含む．このモジュールはメモリより個体を読み出し，それらを 1 つずつ交叉モジュールへ送信する（図 3 （b））. 同時に，重複個体排除モジュールから情報を受信する（図 3 （a））．その際，このモジュールは評価モジュールから受信した情報に従い個体の更新を行う.

交叉モジュール

交叉モジュールは，受信した親個体と，一つ前に受信した親個体を交叉し，一つの子個体を生成する．一つ前に受信した親個体 (アドレス，適応度，染色体) を

格納するための領域を持つ．交叉モジュールでは，選択操作の一部も行う．２つの親の適応度を比べ，悪いほうの個体のアドレスと適応度を突然変異モジュールへ出力する (図 3 (c)).

突然変異モジュール

突然変異モジュールは，交叉モジュールより受信した子の染色体に，与えられた確率で突然変異を適用し，評価モジュールへ送信する．また，交叉モジュールより受信した適応度が悪い方の親個体のアドレスおよび適応度を評価モジュールへ出力する (図 3 (d)).

評価モジュール

評価モジュールでは，突然変異モジュールより受信した子の染色体を評価し，各目的に対する適応度を求める．そして，評価モジュールより受信した情報に子の適応度を加えた情報を管理モジュールへ送信する（図 3 （a））.

2.3.2 並列アーキテクチャ

利用可能な回路規模に応じて性能を改善できるようにするために，基本アーキテクチャを並列化する．基本アーキテクチャにおけるモジュールの組み合わせを 1 つの島と見なし，複数の島からなる IGA を構成する．この際，複数の島の間で個体の交換が行えるようにするため，図 4 に示すように，各パイプラインの管理モジュールと交叉モジュールの間に移民モジュール (immigration module) を導入する．移民モジュールは，図に示すように 2 つの島の管理モジュールと接続される．通常，交叉モジュールは自身が属する島の管理モジュールから個体を読み込むが，決められた周期で他方の島の管理モジュールから個体を読み込む．移民モジュールに接続する管理モジュールの数は，島の数によらず，常に 2 であるため，並列数の増加はクリティカルパスの長さに影響しない.

2.3.3 各モジュールの処理が一定クロックで無い場合の対処

適用する問題 (例えば TSP，Job Shop Scheduling Problem 等) によっては，各遺伝演算子モジュールで行われる処理が複雑になり，処理に必要なクロック数が一

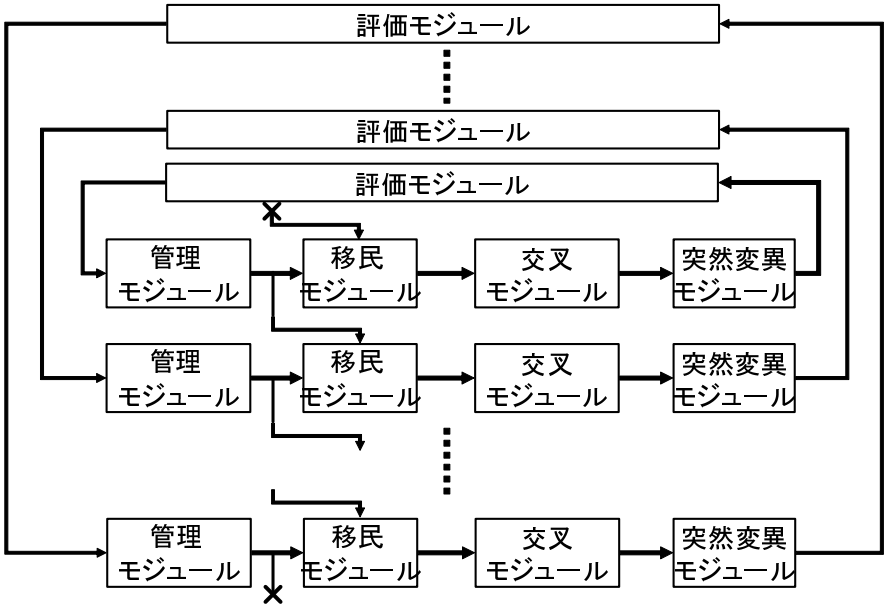

図 4 並列アーキテクチャ（SOGA）

定とはならない.

このようなモジュールが存在すると，パイプラインのストールによって他のモジュールの性能を活かしきることができない．処理クロック数が一定ではないモジュールが存在する場合，以下の２つの手法により性能の低下を防ぐ.

処理クロック数が一定でないモジュールを複数配置し，その前後のモジュールと，これらのモジュールのいずれの間でも送受信が行えるように回路を構成する．処理クロック数が一定でない複数のモジュールを並行に動作させ，処理が終わったモジュールから順に次のモジュールにデータを送るようにすることで，処理時間を平均化する．本論文の評価実験では，この手法は回路規模の増大を招くため使用していない.

もう一つの方法は，処理時間が一定でないモジュールと，その下流のモジュールの間にバッファを配置し，下流のモジュールはバッファから読み込むようにする．処理時間が一定でないモジュールが早く処理を終えると，バッファに書き込み，次の染色体の処理を開始できる．このようにして，処理時間を平均化する.

2.4 回路作成支援ツール

設計した回路を FPGA 上へ実装する際に，その回路の規模と使用メモリ量をもとに，実装可能な FPGA デバイスを決定する必要がある．本節では，回路規模を予測するモデルについて述べた上で回路作成支援ツールについて述べる．

2.4.1 回路規模予測モデル

提案アーキテクチャに従い設計した SOGA 回路は，複数のモジュールから構成される．各モジュールの回路規模は個体数の対数と問題サイズの一次関数で予測できることが予備実験の結果から分かっている．また提案アーキテクチャではモジュール間の通信のための制御回路は単純であるため，予測するべき回路全体の回路規模は，使用されるモジュール群の回路規模の総和として近似可能だと考えられる．これらより，回路規模予測モデルを構築する．各モジュールの回路規模を予測するための一次関数の係数は，各モジュールごとに個体数の対数と問題サイズを変化させ，実際に論理合成を行って得た回路規模から，重回帰分析により得る．本論文では，各パラメタを 3 通りに変化させ，モジュールごとに $3 \times 3 = 9$ 通りの論理合成を行なった結果から係数を求めることとする．全体の回路規模は，回路全体に含まれる各モジュールの数に，各モジュールの予測回路規模を掛けたものの総和により予測する．

また，各モジュールで用いられる memory block を合計することで，メモリの面から実装可能デバイスが判明する．この使用メモリ量の予測と回路規模予測モデルを組み合わせることで実装可能デバイスが判明する．

実験結果は 2.5 節で示す．

2.4.2 回路作成支援ツール

指定した FPGA デバイスに対して，最適な回路のパラメタ値を探索し，回路の RT レベル VHDL 記述を自動的に生成するツールを作成した．ツールは，問題サイズ，個体数，実装先 FPGA などのパラメタから，実装可能である並列度を返す

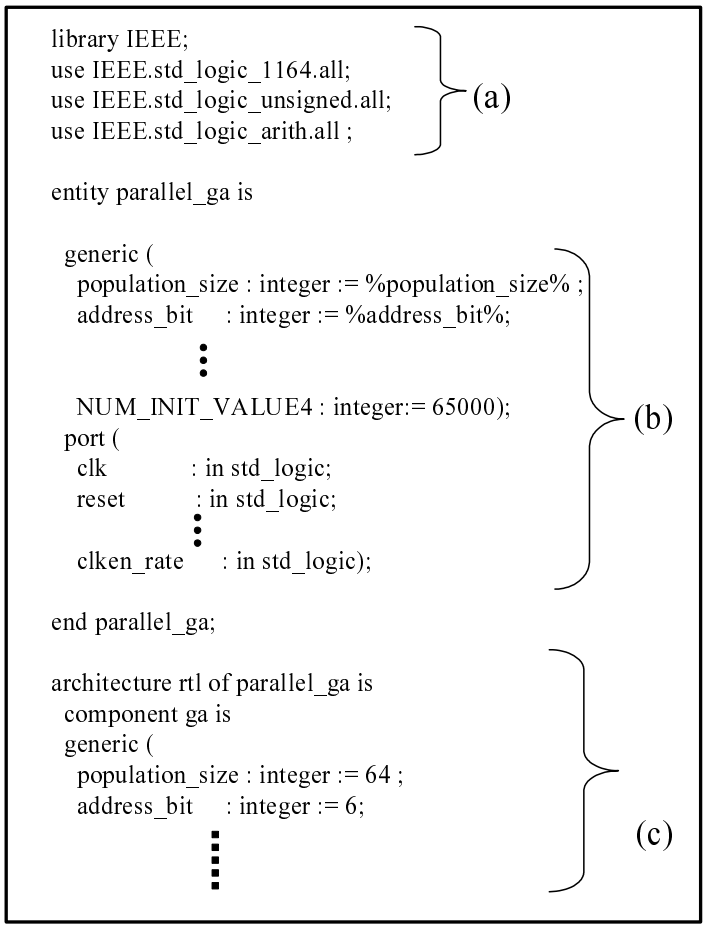

図 5 テンプレートファイル

実装判定部と，実装判定部により得られたパラメタ値から，自動的に VHDL で記述した島モデル型 GA 回路を出力する回路作成部からなる．

実装判定部は，前に述べた回路規模予測モデルを用いて判定を行う．

回路作成部は，各モジュールを VHDL で記述したファイルの集合であるライブラリと，回路を自動生成するために必要なテンプレートファイル (図 5) とコンポーネントファイル (図 6) から構成される．テンプレートファイルには library 宣言 (図 5 (a)) と entity 宣言 (図 5 (b))，architecture(図 5 (c)) の一部が記述されている．コンポーネントファイルは，回路とインターフェースモジュールのコンポーネン

```
interface%num% : interface
 generic map(
    fitness_bit => fitness_bit ,
    num_evaluate_bit => num_evaluate_bit)
 port map( clk => clk ,
    din_num_evaluate1 => line_interface%num_1e%_evaluate ,
    din_num_evaluate2 => line_ga%num_2%_num_evaluate ,
    din_best_fitness1 => line_interface%num_1f%_best_fitness ,
    din_best_fitness2 => line_ga%num_2%_best_fitness,
    dout_num_evaluate => line_interface%num_e%_evaluate,
    dout_best_fitness => line_interface%num_f%_best_fitness);
```

図 6 コンポーネントファイル

トインスタンス文が記述されている．回路作成ツールは，テンプレートファイルとコンポーネントファイルを用いて SOGA 回路を記述する．その際，パラメタに相当するテンプレートファイル内部の%ラベル名%を指定した値に置き換える．このパラメタには，個体数，問題サイズ，並列度がある．コンポーネントファイルでも同様の操作を行なう．問題に対応したライブラリとテンプレートファイル，コンポーネントファイルを用意することで，その問題に対応した島モデル型 GA 回路を出力することができる．

また，これらのツールの実行時間は，2.5 節のアプリケーションに適用した場合，いずれも 1 秒以内となり，実用上十分高速であるといえる．

2.5 実験結果・評価

本節では，提案アーキテクチャに従った，ナップサック問題と TSP に対する GA 回路の実装について述べる．

次に 2.4.2 節で述べたツールと Altera 社の Quartus II を用いた実験結果について述べる．ターゲットデバイスとして，Altera 社の Cyclone を選び，メモリは，FPGA の内部メモリを使用する．

2.5.1 ナップサック問題への適用

ナップサック問題は，以下のように定義される．荷物の集合 S とナップサックの大きさが，入力として与えられる．但し，各荷物には容積と価値が与えられている．ナップサック問題は，荷物の容積の総和がナップサックの大きさを超えず，価値の総和を最大化するような，S の部分集合を求める問題である．

本論文でのナップサック問題に対する実装では，品物の数を s とするとき，染色体の長さを s ビットの長さとする．各ビットが各品物に対応する遺伝子座になる．ある遺伝子座の遺伝子が “1” の時は，対応する品物をナップサックに入れ，“0” である場合は入れないことを示す．s ビットの染色体全てが，1 クロックごとに各モジュールに入出力されるとする．

交叉モジュール

交叉法として，一様交叉を用いる．すなわち，子個体の染色体は，各遺伝子ごとに親個体のどちらか (ランダムに決定) の対応する遺伝子をコピーすることで生成する．

突然変異モジュール

突然変異モジュールは，各遺伝子座ごとに，突然変異率で与えられた確率で，ビットを反転する．

評価モジュール

適応度は以下のように定義される．

- ナップサックに入っている品物の容積の総和がナップサックの容量を越えている場合，適応度は 0.

- 上記以外の場合，適応度はナップサックに入っている品物の価値の総和.

評価モジュールは，各品物の価値と容積をそれぞれレジスタに保持する．これらのレジスタの内容は，計算開始時に外部より与えられる．容積と利益の総和の計算は，クリティカルパスの延長による最大動作周波数の低下を避けるために，以下のようにして計算する．1 クロック目に，品物を 2 つずつペアにしてそれぞれのペアの和を並列に計算する．2 クロック目に，1 クロック目で得られた和をペアにして，それぞれの和を並列に計算する．以上を繰り返し，全ての品物の総和を求める.

2.5.2 巡回セールスマン問題への適用

TSP は，完全グラフ $G(V, E)$(ここで，V は頂点の集合，E は辺の集合) が入力として与えられる．グラフの頂点を都市と呼ぶ．各辺には正の距離が与えられる．TSP は，すべての都市を 1 度ずつ訪れる巡回経路の中で，経路に含まれる辺の距離の合計が最小のものを求める問題である.

TSP では，染色体は，巡回する各都市の番号を各遺伝子として，全ての都市番号を連結したものとする．例えば，51 都市の TSP を扱う場合，各都市を表現するために 6 ビット必要になる．従って，染色体を表現するために，$51 \times 6 = 306$ ビット必要になる．本実装では，適応度を 16 ビットで表現する.

306 ビットの染色体を 1 クロックで送受信すると，回路規模が大きくなってしまうので，染色体 1 つの送受信は，複数クロックかけて行なう．本実装では，各遺伝子 (都市) のビット数を m，訪れる都市数を n とすると，1 クロックあたり m ビット，それぞれの染色体を，n クロックで送受信する.

交叉モジュール

交叉法として，部分写像交叉 (PMX)[13] を採用した．PMX は，致死個体を発生させない交叉法であり，ハードウェア化が比較的容易である．まず 2 つの異なる遺伝子座を乱数により決定する．2 つの間の遺伝子は，親 1 からコピーし，それ以外の部分は親 2 からコピーすることで染色体を生成する．染色体中に 2 回以上現われる都市については，一度も出現しない都市に置換する.

突然変異モジュール

突然変異には 2 都市交換を用いる．すなわち，染色体内部の乱数で選択した 2 つの遺伝子座の遺伝子を入れ替える．

評価モジュール

染色体の適応度は，染色体に記された順序に従い全ての都市を巡回したときの経路の距離の総和によって計算される．評価モジュールに，各都市間の距離のテーブルを保持させる．このテーブルは，回路が動作する前に予め初期値として与えられる．各都市間の距離は，8 ビットの整数で保持する．

評価モジュールが一つの遺伝子を受信すると，受信した遺伝子と 1 つ前に受信した遺伝子間の距離を距離テーブルより取得し，それを適応度に加算する．

2.5.3 アーキテクチャ性能評価

提案手法により生成される回路の性能の妥当性を調べるため，生成した回路の解探索能力と処理速度を調べた．処理速度の見積もりを行うには，提案手法による回路の最大動作周波数を知る必要があり，そのためには，論理合成ツールが扱う FPGA デバイスに回路を合成できなければならない．ここでは，実際に提案手法のハードウェア化した際の回路規模と実装先デバイスの回路規模より，性能比較には，問題サイズが 64 のナップサック問題と eil51 と呼ばれる都市数 51 の TSP を採用した．

比較対象として SGA を用いる．TSP では，交叉法に PMX と枝交換交叉 (EXX)[26] を採用した SGA を用いた．これはソフトウェア GA では，複雑であるが解探索能力の高い EXX の実装が妥当と思われるためである．

ソフトウェア GA は，Pentium 4 2.4 GHz と 256MB のメモリを持つ PC 上で実行した．またプログラムのコンパイルには gcc のバージョン 2.95.4 を用いコンパイルオプションとして `-O3` を用いた．交叉率，突然変異率，移民率などのパラメタは予備実験を行い，最も優れた値を使用した．

まず，提案手法の特徴を確認するため，評価回数あたりの性能比較を行った．図 7，8 に実験結果を示す．図の MGG (1) は提案手法を並列度 1 で実行した場合を示し，SGA(PMX) は，PMX を用いた場合の性能を示している．この結果より，提案手法は，ナップサック問題と TSP の両者ともに SGA と比較して，探索の初

期段階においてより優れた性能を示すことが分かる．しかし，図 8 より，TSP では SGA と比較して探索の中盤から終盤において性能が逆転していることより，提案手法は SGA と比較して早期収束に陥いりやすく，探索回数を増やした際の解の質はやや劣ることがわかった．しかしながら，解の質の差は軽微であり，ハードウェア化の際には MGG モデルを採用することは妥当であると考えられる．

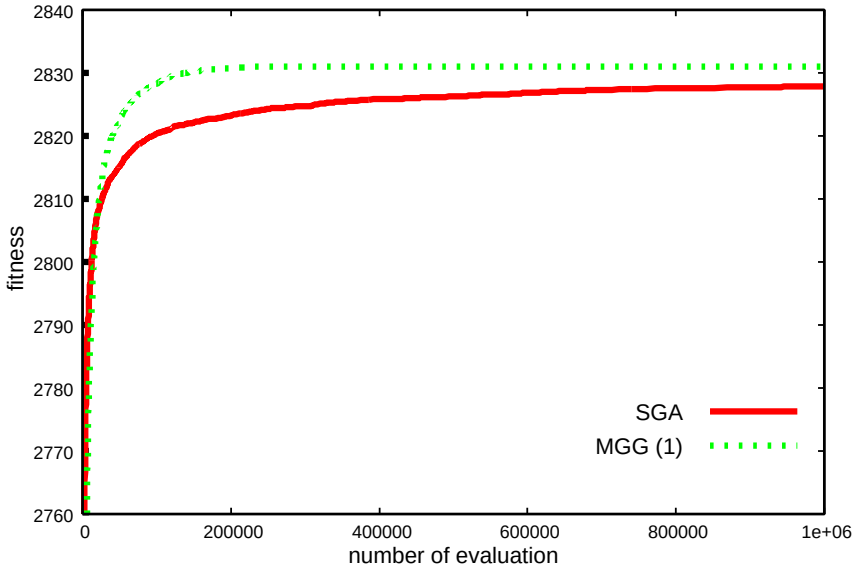

図 7 評価回数辺りの性能変化（ナップサック問題）

次に，単位時間あたりにどれだけ良い解が得られるかを計測した．提案アーキテクチャの処理時間は，Quartus II でのシミュレーションにより算出した．ターゲット FPGA デバイスとして，Altera 社の Cyclone シリーズ [1] を用いた．

それぞれの結果を図 9，10 に示す．ただし，図 9 は，適応度が高いほど良い解を，図 10 では低いほど良い解を表している．各図において MGG は提案アーキテクチャの性能を示し，カッコ内の数字はその並列度を示している．図 10 の SGA(PMX) と SGA(EXX) は，それぞれの交叉法を用いた場合でのソフトウェア GA の性能を示している．また，MGG (8)，MGG (16) は回路規模の点で 1 つの FPGA 上に実装することは不可能である．そのため計算速度の算出には，MGG (4) を実装した複数の FPGA を用いるものとして計算した．また，提案アーキテクチャに従い設

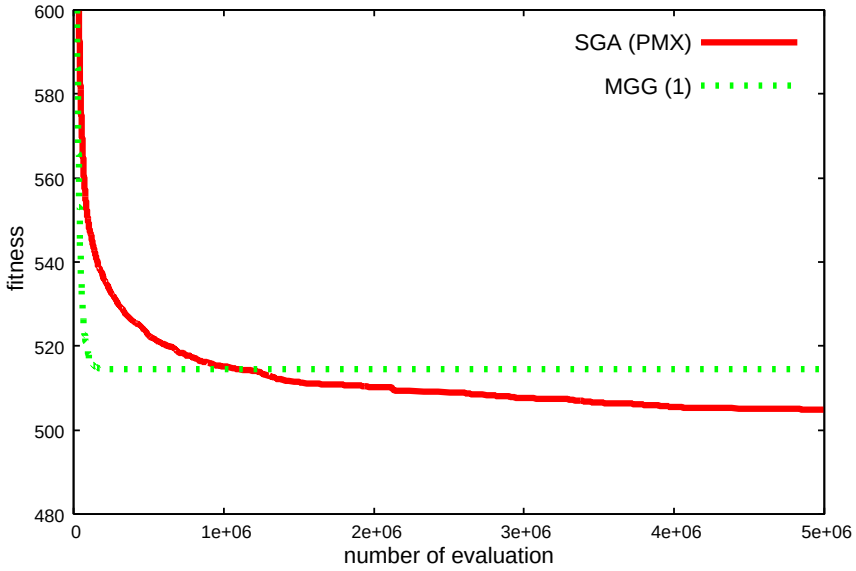

図 8 評価回数辺りの性能変化（TSP）

計した回路の最大動作周波数 (Maximum Operation Frequency，MOF) を表 1 に示す．また図 9 では，ソフトウェア GA を用いたところ収束まで約 1 秒かかり，結果が図に入らなかったため省いている．

図 9，10 より，提案手法による回路は解が収束するまでの時間が十分に短く，また，並列度数が高い場合，より高度な交叉アルゴリズムを用いたソフトウェア実装に匹敵する解探索能力を達成できることがわかる．特にナップサック問題では，適応度が 2829 に達するまでにかかる時間は MGG (1)，MGG (2)，MGG (3) においてそれぞれ，1022 (μs)，628 (μs)，581 (μs) かかる．それに対し，SGA は $261\times10^4(\mu s)$ かかることより，提案手法は SGA より最大約 4490 倍優れていることがわかる．このことより，提案手法は，回路規模に応じて並列度数を上げることで，GA を高速化し，性能を向上させることができることがわかる．図 10 における SGA(EXX) と比較すると，MGG (16) とほぼ同等の性能を示していることから提案手法において複雑な交叉法をモジュール化することにより，さらなる性能向上が可能になると考えられる．

次に，基本アーキテクチャの 1 個体当たりの処理時間を表 2 に示す．表 2 より，

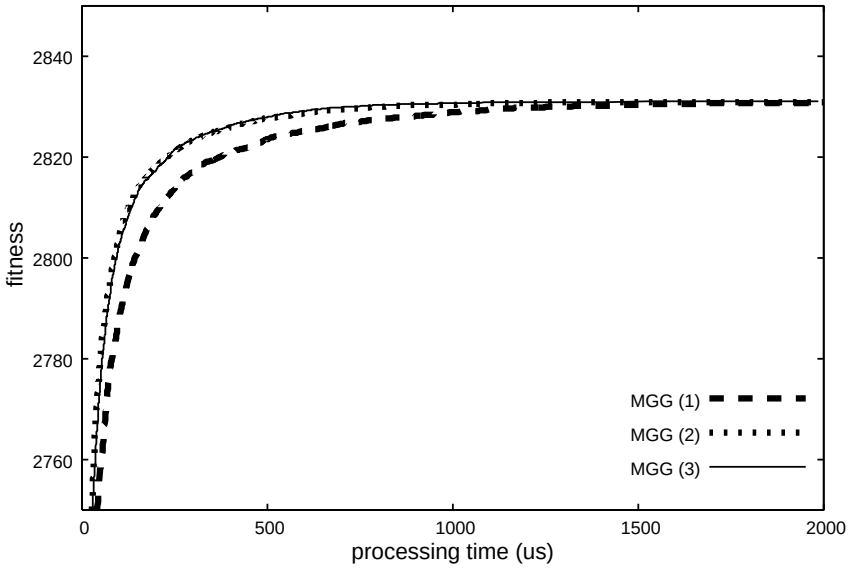

図 9 ナップサック問題の解探索能力

提案手法により作成した回路は問題サイズが増加するにつれ最大動作周波数が低下するものの，ソフトウェアで実装した SGA より計算速度の点で十分優れた性能を示している．

提案する並列アーキテクチャの利点を明確にするため，提案アーキテクチャにより作成した GA 回路と単純な並列化を行った GA 回路 (単純並列化 GA 回路，simple parallel circuit) の性能の比較を行った．単純並列化 GA 回路は，提案アーキテクチャによる並列回路とは異なり全てのパイプライン間でデータの共有を行なう回路である．対象となる問題には，問題サイズ 32 のナップサック問題を使用した．表 3 はそれぞれの回路の並列度ごとの最大動作周波数の比較を示したものである．結果より，提案手法による島モデル型 GA 回路の最大動作周波数の変化は単純並列化 GA 回路に比べ十分低く抑えられていることが分かる．

また，消費電力の結果を表 4 に示す．消費電力は問題サイズの増加に伴い増加しているが，Pentium 4(2.4GHz) の TDP (Thermal Design Power) の約 1/80 であり，低消費電力を実現できていることがわかる．

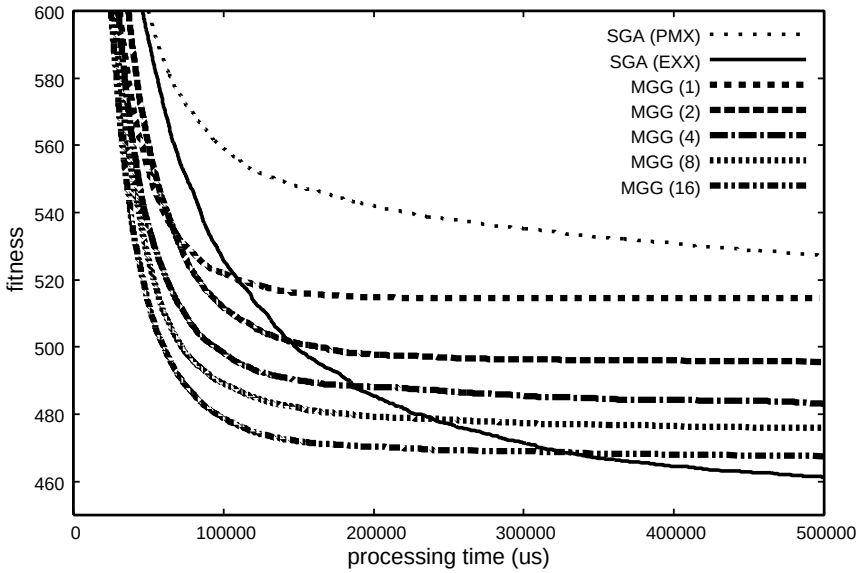

図 10 TSP(eil51) の解探索能力

表 1 提案アーキテクチャの最大動作周波数

Problem	number of pipeline	MOF
Knapsack Problem	1	108 (MHz)
	2	105 (MHz)
	3	102 (MHz)
TSP	1	135 (MHz)
	2	109 (MHz)
	4	110 (MHz)

表 2 一個体にかかる処理時間

Problem	Problem Size	Basic Architecture		SGA
		Processing time	MOF	
Knapsack Problem	16	7.09 (ns)	141 (MHz)	0.508 (μs)
	32	7.25 (ns)	138 (MHz)	0.765 (μs)
	64	9.26 (ns)	108 (MHz)	1.36 (μs)
	128	8.47 (ns)	118 (MHz)	2.24 (μs)
TSP	51	1.28 (μs)	135 (MHz)	2.07 (μs)
	76	2.06 (μs)	120 (MHz)	2.25 (μs)
	101	2.99 (μs)	113 (MHz)	3.32 (μs)

表 3 比較対象の手法の最大動作周波数

concurrent pipeline	our parallel circuit (MHz)	simple parallel circuit (MHz)
2	106	139
3	119	107
4	109	93
5	114	96

表 4 消費電力

Problem	Device	Total Power
-	Pentium4(2.4GHz) [3]	57.8 (W)
Knapsack Problem	FPGA($s = 16$)	293 (mW)
	FPGA($s = 32$)	362 (mW)
	FPGA($s = 64$)	427 (mW)
	FPGA($s = 128$)	700 (mW)
TSP	FPGA($s = 51$)	476 (mW)
	FPGA($s = 76$)	511 (mW)
	FPGA($s = 101$)	611 (mW)

2.5.4 回路規模予測モデル

2.4 節で提案した回路規模予測モデルの正確さを評価するために，並列度の異なる複数の回路を構築し，その実測値と予測値との比較を行なった．

それぞれの回路における予測値と実測値の比較を図 11，12 に示す．ここで，括弧内に記された数字は並列度を示している．回路規模の予測値と実測値の予測誤差は最大 3%であり，提案モデルが実用上十分正確であることが確認された．

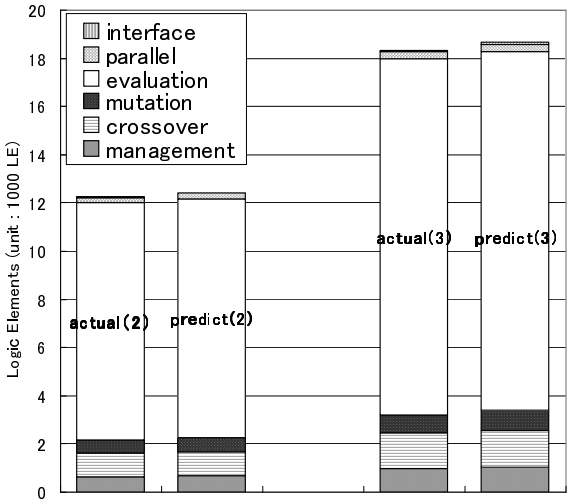

図 11 回路規模予測モデル (ナップサック問題，64)

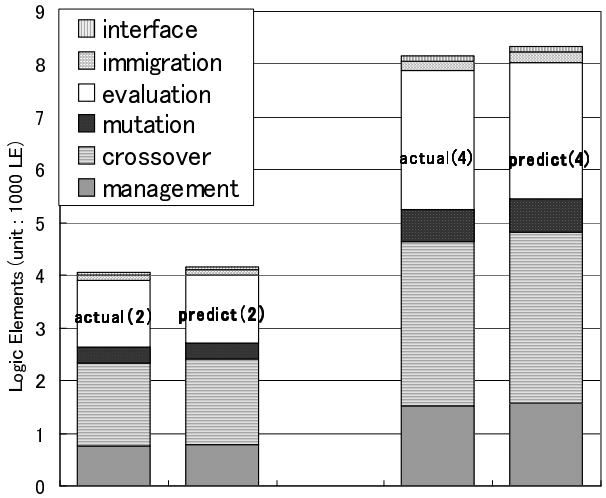

図 12 回路規模予測モデル (TSP，eil51)

2.6 まとめ

本章では，FPGA 上への SOGA の柔軟な実装方法を提案した．提案手法により，デバイスに適した回路を作成することができる．実験結果より，設計した回路は，消費電力が Pentium 4 2.4GHz の約 1/80 で動作することを確認した．また，回路規模の予測値と実測値の誤差が 3%程度であることを確認した．

3. MOGAのハードウェア実装に適したアーキテクチャとその実装手法

3.1 はじめに

本章では，MOGA をハードウェア回路として実装するためのアーキテクチャ（MOGA アーキテクチャ）を提案する．また，提案アーキテクチャでは解探索能力の向上と回路のスケーラビリティのために，島モデル型 GA を拡張し，島ごとに異なる部分のパレート最適解を探索させる MOGA に適した並列実行または並列化手法を採用する．

3.1.1 関連研究

本節では，MOGA のハードウェア化に関する関連研究について述べる．SOGA のハードウェア化については，2.1.1 節で述べた．しかし，これらの研究は SOGA をターゲットとしており，著者らの知るかぎりにおいて MOGA をハードウェア化した研究は見あたらない．

また，ソフトウェアで実装した MOGA の研究には以下の例がある．文献 [7] では，Deb らが Non-Dominated Sorting Genetic Algorithm-II (NSGA-II) を，文献 [21] では，Zitzler らによる Strength Pareto Evolutionary Algorithm-II (SPEA-II) を提案している．これらの手法は，それぞれ独自のランク戦略を用い，優れた性能を示している．しかし，これらは機構が複雑であるためハードウェア化には適さない．

3.2 提案手法

SOGA 回路を用いて多目的最適化問題を解くことは困難である．それは，SOGA は 1 つの最適個体を求めるのに対し，MOGA では多様なパレート最適解を探索することが要求されるからである．本節では 2 章で提案した SOGA アーキテクチャ [27] を拡張し，MOGA をパイプライン処理でハードウェア化するためのアーキテクチャを提案する．しかし，MOGA のハードウェア化には，MOGA ではパレート最適解を効率良く探索するため，多様性を維持するための手法であるニッチ法やランク戦略が用いられる．しかし，これらをパイプライン処理で実装することは困難であるという問題がある．

この問題を解決するため，パイプライン処理に適した多様性を維持するための手法を考案し，実装を行った．本論文では，考案した機構を重複個体排除機構と呼ぶ．また，解探索の性能を向上させるため，MOGA に適した並列化手法も提案する．

本節では，3.2.1 節で MOGA と MOGA のハードウェア化の際の問題点とそれらについて述べる．そして，3.2.2 節で，提案アーキテクチャの世代交代モデル，3.2.3 節で重複個体排除機構，3.2.4 節で MOGA に適した並列化手法について述べる．

3.2.1 多目的遺伝的アルゴリズム

MOGA とは多目的最適化問題を解くために GA を拡張した手法である．多目的最適化問題では，同時に複数の目的を最適化する必要があるため，1 つの目的に対する適応度の比較では解の優劣を決めることができない．

多目的最適化問題では，解候補である個体の優劣は以下のように定義される．2 つの解候補である個体 *x*，*y* があると仮定する．ここでは目的の集合 P の要素である目的 p に対する目的関数値が，それぞれ $f_p(x)$，$f_p(y)$ で表わされるものとする．目的関数値は，その個体がその目的においてどれだけ良いかを表す値である．ここでは，目的関数値が大きいほど良いとする．この時，以下に示す式が成立すると，個体 x は個体 y を**支配する**といい，個体 x は個体 y より優れているも

のとする．

$$\exists_{p\in P}(f_p(x) > f_p(y)) \wedge \forall_{p\in P}(f_p(x) \geq f_p(y))$$

パレート最適解は解空間において他のどの解にも支配されない解を指す．一般にパレート最適解は解空間中の点の集合として表現される．多目的最適化問題は全てのパレート最適解の集合（パレートフロント）を見つける問題である．

また，MOGA は，多目的最適化問題を解くために，解候補である**個体**を複数用いて近似解を求める．個体は，**染色体** (chromosome) と各目的の目的関数値 (適応度) から成る．染色体は，与えられた問題の探索空間内の一点 (解候補) を表す．MOGA で一般に従来からよく用いられる世代交代モデルは，以下の操作からなる（図 1 参照）.（i）多数のランダムな個体を作成する.（ii）各個体の適応度を計算する (**評価操作**).（iii）各個体の支配・被支配関係を調べ，支配されていない個体を優先的に残し，残りの個体を破棄する (**選択操作**).（iv）個体群から 2 つの個体を取り出しペアを作成する．これらを両親 (parent) とし，両親の性質を混ぜ合わせて新しい個体を作成する (**交叉操作**).（v）新しい個体の染色体に対しある確率で突然変異を起こす (**突然変異操作**).（vi）ステップ（ii）に戻る．MOGA はこれらの操作を指定した時間の間くりかえし適用することでパレート最適解を探索する．MOGA と SOGA のアルゴリズムの最大の違いは選択操作にある．MOGA では，ランク戦略などの手法を用いて選択操作を行なう，

次に，NSGA-II[7] で用いられている次世代の個体群を決定するための手法であるランク戦略と混雑度（crowding distance）について述べる．

NSGA-II では，個体群に属する個体を以下の手順に従いランクをつける．

1. 個体群 P に属する各々の個体 x_i のエンティティとして，個体 x_i を支配する個体数 n_i と個体 x_i が支配する個体の集合 S_i を求める．

2. $n_i = 0$ である全ての個体 x_i の集合を $F_t, t = 1$ とする．

3. F_t に属する全ての個体 x_i のエンティティである S_i に属している個体を x_j とする．この個体 x_j のエンティティである n_j より 1 を引く．

4. $n_j = 0$ となる個体 x_j の集合を H とする．

5. $P = P - F_t, t = t + 1$.

6. 集合 H を F_t として，$|P| = 0$ であれば終了．そうでないのであれば 3 に戻る．

NSGA-II では集合 F_i の属している個体をランク i とする．そして，ランク 1 に属する個体が最も優れた個体であるとし，ランクの異なる個体間での個体の優劣を決定する．

また，NSGA-II では，同ランクに属する個体間の優劣を決定するために．混雑度を導入している．混雑度は，図 13 に示す様に，ある個体 x_i の周囲にある同ランクに属する個体 x_k，x_j によって描かれる四角形を構成する辺 E_{f1}，E_{f2} の平均値を用いる．この個体 x_k，x_j を用いて描かれる四角形の内部には個体 x_i 以外の同ランクの個体を含まない．ここでは，混雑度が大きければ大きいほど優れた個体であると見なす.

NSGA-II では，個体群と新たに生成された個体群にこれらの手法を適用し，全ての個体の優劣を決定する．そして，MOGA のパラメタの 1 つである個体数が N である時，上位 N 個の個体を次世代の個体群として使用する．

これらの手法では，全ての個体のランクと混雑度を決定するためには，繰替えし個体群の比較を行う必要がある．そのため，パイプライン処理によるハードウェア化は困難である．そこで本論文では，新たに多様性を維持しつつ，複数のパレート最適化問題を探索するための手法を提案する．

3.2.2 提案手法における世代交代モデル（MOGA）

本節では MGG における世代交代モデルについて述べる．2.2.3 節で述べた世代交代モデルとは選択操作のみが異なる．この世代交代モデルでは，交叉・突然変異操作の際，個体群中より選択された 2 つの親個体より 1 つの子個体のみ作成す

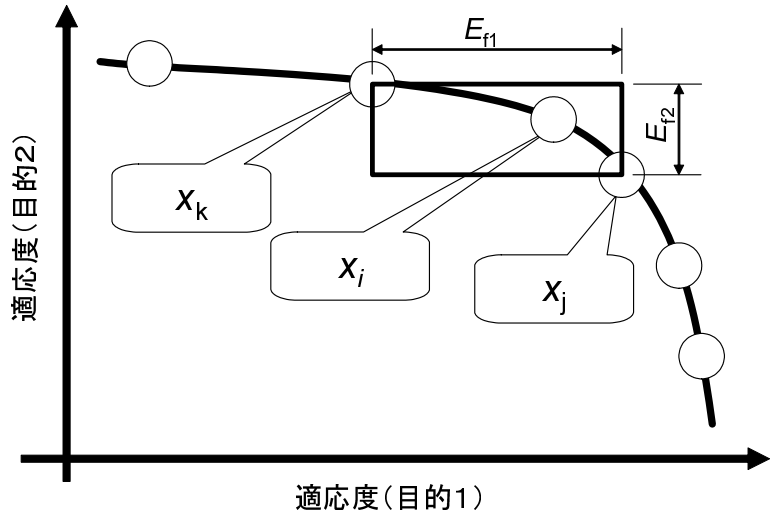

図 13 混雑度（crowding distance）

る．次に生成された子個体を評価する．MOGA における選択操作では，まず，2 つの親個体を比較し支配されている親個体（被支配個体）を選ぶ．そして，選ばれた親個体と新しく生成された子個体を比較し，子個体が親個体を支配している場合には，比較した親個体と子個体を入れ替える（図 2 参照）．

3.2.3 重複個体排除機構

従来の手法であるニッチ法やランク戦略は，個体群中で全ての個体間で比較を行うためにソートなどの繰り替えしを伴う操作を必要とし，パイプライン処理での実装は困難である．しかし，個体数が少なく，これらの多様性を維持するための手法を採用しない場合，重複個体が発生し，多様性が失われる．そこで，提案アーキテクチャでは個体群の多様性を維持するため，重複個体の排除を基本とした手法を採用する．本節ではパイプライン処理での実装が容易である多様性を維持するための機構，重複個体排除機構を提案する．この機構は選択機構における結果を反映して，親個体を子個体に入れ替える機能を持つ．

提案機構は，パイプライン処理で実装するため，子個体と全ての親個体の逐次

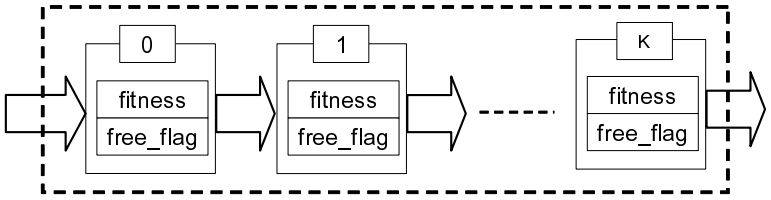

図 14 重複個体排除機構

比較を行う．この時，子個体と親個体が同一であり，子個体に印がついてない場合，子個体に印をつけ，子個体に印がついている場合，親個体に印をつける．また，子個体が比較した個体が支配していれば，その個体に印をつける．

比較の際，提案機構は，入力された子個体を (1) MGG モデルによる個体の置き換えを行なう子個体と (2)MGG モデルによる置き換えを行わない子個体とで処理が異なる．(1）の子個体は，比較した個体が置き換えるべき親個体（被支配個体）であれば置き換える．(2）の子個体は，自身に印が無い場合，(i) 比較した個体を支配している場合と（ii）比較した個体に印があり，かつ，自身が比較した個体に支配されていない場合，置き換えを行う．それぞれ，親個体との置き換えを行った子個体は，印をつける．機構の簡易化のため，個体が同一かであるかのチェックは適応度の比較により行う．

重複個体排除機構を以下の様に実装する（図 14 参照）．k 個のサブモジュールを用意する．ここで k は MOGA で保持する個体数であり，MOGA は親個体の集合 $Y = \{y_1, ..., y_k\}$ を保持している．各サブモジュールは 3 つのパラメタを持っている．それぞれ，サブモジュールの識別番号 i，個体群中の個体 y_i の適応度 $fitness_i^y$，個体 y_i が個体群中で重複するかどうかを示すフラグ $free\_flag_i^y$ である．各サブモジュールはパイプライン処理される．i 番目のサブモジュール S_i は，$i-1$ 番目のサブモジュール S_{i-1} から情報を受信する．ここでは，各サブモジュールが受信する新しい子個体の集合を $X = \{x_1, ..., x_k\}$ とする．サブモジュール S_i は以下の情報を受信する．子個体 x_i の適応度（$fitness_i^x$）と染色体（$chromosomes_i^x$），アドレス（$overwrite\_address_i^x$），選択操作による子個体と親個体を入れ替えるかどうかを表すフラグ（$selected\_flag_i^x$），個体群中に重複個体を発見したかを示す

フラグ（$found\_flag_i^x$）である．ここではアドレス（$overwrite\_address_i^x$）の初期値は親個体のアドレスであり，各フラグの初期値は false である．

サブモジュール S_i のアルゴリズムを以下に示す．

1. 入力として、サブモジュールは，サブモジュール S_{i-1} から以下のパラメータセットを受け取る。
   $fitness_i^x$, $chromosome_i^x$, $selected\_flag_i^x$,
   $overwrite\_address_i^x$, $found\_flag_i^x$.
2. If $found\_flag_i^x$ == false then goto 5.
3. If 個体 x_i が個体 y_i を支配，または 重複個体 then $free\_flag_i^y \leftarrow$ true.
4. Goto 11.
5. If ($selected\_flag_i^x == true$ and $i == overwrite\_address_i^x$) == false then goto 7.
6. $fitness_i^y \leftarrow fitness_i^x$,
   $free\_flag_i^y \leftarrow$ false,
   $found\_flag_i^x \leftarrow$ true,
   goto 11.
7. If 個体 y_i と個体 x_i が重複個体 then
   If $found\_flag_i^x$ == false then
   $found\_flag_i^x \leftarrow$ true,
   $selected\_flag_i^x \leftarrow$ false,
   goto 11.
   else
   $removed_flag_i^y \leftarrow$ true,
   goto 11.
   else goto 8.
8. If 個体 x_i が個体 y_i が支配 then
   $free\_flag_i^y \leftarrow$ ture.
9. If ($free\_flag_i^y$ == false or $selected\_flag_i^x$ == true) then goto 11.
10. $fitness_i^y \leftarrow fitness_i^x$,
    $free\_flag_i^y \leftarrow$ false,

$found\_flag_i^x \leftarrow$ true,
$selected\_flag_i^x \leftarrow$ true,
$overwrite\_address_i^x \leftarrow i$.

11. 出力として、次のサブモジュール S_{i+1} に子個体 x_i のパラメータセットを渡す。

（2）から（4）は重複個体または親個体が支配されていたことによる上書きのためのフラグたてを行なう.（5）から（6）は，MGG による個体の置き換えを行う（7）は親個体と子個体が重複個体であるときの操作を行う.（8）から（10）は親個体に印がついているときの子個体の上書きを行う.

3.2.4 MOGA に適した並列化手法

2.2.4 節の SOGA アーキテクチャでは，回路の並列化による最大動作周波数の減少を防ぐために島モデル型 GA（Island GA，IGA）を基にした並列化手法を採用した．この手法では隣り合う 2 つの島間のみで，個体のやりとりを行うことで，機構を簡易化し，クリティカルパスの長さを抑え，最大動作周波数の減少を抑えている．しかし，この手法を MOGA に適用した場合，全ての島が解空間中に存在するパレート最適解の同一部分のみ探索する恐れがある．また，多目的最適化問題を効率良く解くための手法として島ごとに異なる目的を優先して探索する手法も提案されている [6].

これらより，島ごとに解空間中の異なる部分のパレート最適解を探索することで効率良く解探索を行う手法を採用する.

提案手法による並列 MOGA 回路は l 個の特定の目的を優先して探索する**偏向 MOGA** と 1 個以上の一般的な MOGA（**通常 MOGA**）から成る．l は対象となる問題の目的の数である．通常 MOGA と偏向 MOGA はそれぞれ**通常選択**と**偏向選択**を用いる．通常選択と偏向選択は MGG モデルによる個体の置き換えを決定するときに用いる個体の優劣を決定する基準が異なる．通常選択は，個体を更新する基準として，子個体が親個体を支配しているとき個体を入れ替える．それに対し，偏向選択では，ある特定の目的に関して，子個体が親個体より優れている場合，子が親を支配していなくても，個体の入れ替えを行なう．偏向選択を用いる

ことで特定の目的を優先した探索を行うことができる．また，島間で情報の共有を行うために一定期間ごとに隣りの島から 1 つの個体を移民する．

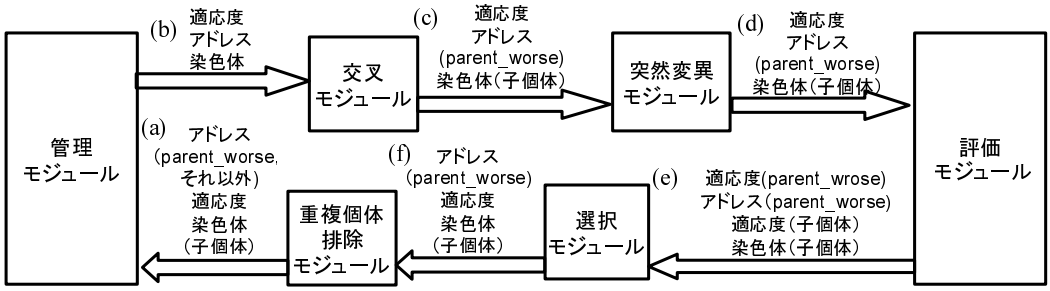

図 15 基本アーキテクチャ（MOGA）

3.3 提案アーキテクチャのハードウェア化

本節では，3.2 節で述べたアーキテクチャの実装について述べる．提案アーキテクチャは文献 [27] で提案されているアーキテクチャを拡張したものであり，モジュール単位で構成されている．本節で新たに追加したモジュールは，**選択モジュール（selection module）**，**重複個体排除モジュール（overlap rejection module）**である．

提案アーキテクチャは，最小の MOGA 回路を構築するための基本アーキテクチャ（図 15) と並列回路を構築するための並列アーキテクチャ（図 16) から成る．

モジュールの設計の概要を 3.3.1 節で述べる．そして従来から存在するモジュール群を 3.3.2 節で，新たに多目的最適化問題に適用するために追加した拡張モジュール群について 3.3.3 節で述べる．

3.3.1 モジュールの設計概要

提案アーキテクチャでは個体は染色体と適応度から構成される．各個体の染色体は，n ビットの 2 進数で表される．モジュール間の染色体の送受信に用いられるバス幅は m ビットとする．n，m の値はパラメタとして与える．各モジュールは，原則として各クロックごとに m ビットのデータを受け取り，c クロック後に m ビットの演算結果を出力できるように設計する（ここで c は定数)．また，データ処理中も，各クロックごとに新たなデータを受け取れるよう設計する．すなわ

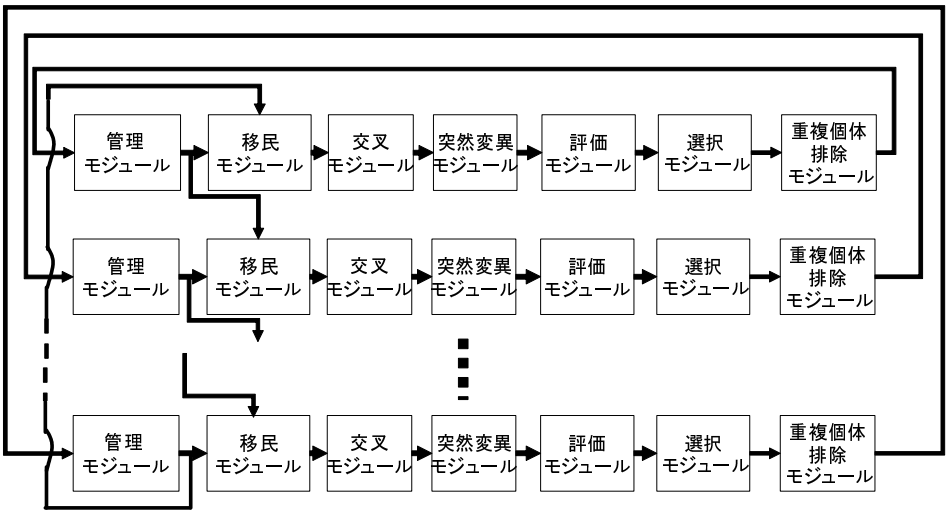

図 16 並列アーキテクチャ（MOGA）

ち，$n \geq m$ の場合，$\lceil \frac{n}{m} \rceil$ クロックが一染色体あたりのスループットとなる．各モジュールは，順番にデータを受信しつつパイプライン的にデータを逐次処理する．

3.3.2 従来モジュール群

管理モジュール 管理モジュールは個体群を保持するメモリを含む．このモジュールはメモリより個体を読み出し，それらを 1 つずつ交叉モジュールへ送信する（図 15 （b））．同時に，重複個体排除モジュールから情報を受信する（図 15 （a））．その際，このモジュールは重複個体排除モジュールから受信した情報に従い個体の更新を行う．

交叉モジュール 交叉モジュールは $\lceil \frac{n}{m} \rceil$ クロック前に受信した個体（*parent1* とする）の染色体，アドレス，適応度を保持するためのレジスタ r を保持している．交叉モジュールは交叉操作を *parent1* の染色体と受信した最新の個体（*parent2* とする）の染色体に適用し，新しい子個体の染色体を生成し，突然変異モジュールへ送信する．同時に交叉モジュールでは，選択操作の一部も行う．*parent1* と *parent2*

の適応度を比べ，支配されている方のアドレス（*parent_worse* とする）と適応度を突然変異モジュールへ出力する（図 15 （c））．もし，2 つの親が互いに支配しない場合は，*parent1* の個体の適応度とアドレスを交叉モジュールへ送信する．

突然変異モジュール　突然変異モジュールは，交叉モジュールより受信した子の染色体に，与えられた確率で突然変異を適用することによってできた染色体を評価モジュールへ送信する．また，交叉モジュールより受信した親個体のアドレスおよび適応度を評価モジュールへ出力する（図 15 （d））．

評価モジュール　評価モジュールでは，突然変異モジュールより受信した子の染色体を評価し，各目的に対する適応度を求める．そして，評価モジュールより受信した情報に子の適応度を加えた情報を選択モジュールへ送信する（図 15 （e））．

移民モジュール　移民モジュールは，回路の並列化を行う際に用いられる．移民モジュールは，通常属している島の管理モジュールと他の島に属する管理モジュールから情報を受信する．通常，交叉モジュールは自身が属する島の管理モジュールから個体を読み込むが，決められた周期で他方の島の管理モジュールから個体を読み込み交叉モジュールへ出力する．

3.3.3 拡張モジュール群

選択モジュール　選択モジュールは，新たに作成された子個体の適応度と染色体，一方の親個体の適応度，アドレスが入力として与えられる．選択モジュールでは入力された親個体と子個体の適応度を比較し，個体群の中で，子個体を親個体と入れ替えるかどうかを決定する．その結果は重複個体排除モジュールへ *selected_flag* として与えられる（入れ替えが発生するとき，*selected_flag*=1 とする）．選択モジュールは子個体の適応度と染色体，*selected flag*，親個体のアドレスを重複個体排除モジュールへ送信する．

重複個体排除モジュール　重複個体排除モジュールは，3.2.3 節で述べた重複個体排除機構を実装する．このモジュールは，選択モジュールより子個体の情報，親個体のアドレス，*selected_flag* を受信する．そして，3.2.3 節で述べた処理を行い，その結果である子個体の情報，アドレス，個体の更新を行うかどうかを表す $overwrite\_flag$ を管理モジュールへ送信する（個体の更新を行うとき，

$overwrite\_flag = true$ とする）.

表 5 最大動作周波数と回路規模

selection module	Size of circuits (Logic Elements)	MOF (MHz)
normal (1)	7,484	85.7
normal (2)	15,472	76.3
normal (3)	23,156	81.6
normal (4)	30,843	76.9
normal (6)	46,284	85.5
none (1)	6,821	107.7
none (2)	14,141	74.6
none (3)	21,161	86.9
none (4)	28,181	78.2
none (6)	42,300	78.1

3.4 実験結果・評価

はじめに，提案アーキテクチャをハードウェア回路として実装した場合の回路規模と最大動作周波数を測定した．その際，MOGA 回路を VHDL で記述し，Quartus II を用いて論理合成を行ない最大動作周波数（Maximum Operating Frequency, MOF）を測定した[1]．対象問題には，多目的ナップサック問題 [8] の 2KP50-50（目的数 2 個，パレート最適解 52 個）を用いた．交叉操作には半一様交叉を用いた．半一様交叉は，染色体上の各遺伝子をどちらの親から引き継ぐかをランダムに決定する手法であり，それぞれの親から引き継ぐ遺伝子の数は同じである．突然変異操作では，ビット毎に突然変異を行なうかどうかを突然変異率により決定し，

[1] ターゲットデバイスとして，Altera 社の CycloneII シリーズ（回路規模 4,608〜68,416 LE）[2] を用いたが，回路規模の都合上，島あたりの個体数が 64 での実装は困難であった．提案アーキテクチャは，パイプライン処理での実装を考慮して作成しているため，個体数が増加しても最大動作周波数は低下しないと考え，ここでは島あたりの個体数が 2 で論理合成を行い，それより得られた最大動作周波数をもとに計算速度を評価した．

ビット反転を行う手法を用いた．

それぞれの最大動作周波数と回路規模を表 5 に示す．表 5 では，“none” は選択モジュールを実装を用いない場合を, “normal” は選択操作に通常選択を用い実装した場合を示す．括弧内の数字は島の数（並列度）を示す．また，設計した MOGA 回路は島単位で任意に偏向選択と通常選択を切り替えができるように設計を行っている．そのため，偏向選択を用いた回路の最大動作周波数は，通常選択を用いた回路と同じ動作周波数で動作が可能である．この結果より，並列度に対して線形に回路規模が増大している。このことから、回路規模の予測が容易であり、利用可能なデバイスに対しての並列設計が容易であるといえる．

次に，提案アーキテクチャによる MOGA 回路の解探索能力を調査するため，提案手法をソフトウェア上で実装し幾つかの実験を行った．ここでは，予備実験を行い，交叉率を 0.6 から 1.0，突然変異率を 0.02 から 0.05，個体数 64 から，最も優れた結果を示したパラメタを，交叉率，突然変異率として採用した．実験は 10 試行を行い，結果はその平均値を用いた．

まず，提案した重複個体排除機構の能力を調査するために，1,000,000 クロック経過時における mormal(1) と none(1) の解空間中の探索状態を図 17，図 18 を示す．図中の “pareto front” は，解空間中におけるパレートフロントを示し，“non-dominated solutions” は，それぞれの個体群中のパレート最適解を示している．これらの図では，10 試行を行った実験の内，最も優れた結果を図示している．また，図を見やすくするために個体群中のパレート最適解のみプロットしている．図 18 より重複個体排除機構を用いることで，パレートフロント上のパレート最適解を多様性を維持しつつ探索していることがわかる．これは，図 17 より MGG モデルによる個体の置き換えを行った場合でも同じであることがわかる．

次に，MGG モデルによる選択操作の違いによる性能比較のため，noraml（6），biased（6），none（6）の評価回数辺りの，見つけたパレート最適解の個数を図 19 に示す．ここでの “biased” は，提案した並列アーキテクチャで実装した MOGA 回路である．この結果より，本論文で採用している MGG モデルによる個体の置き換えは，多目的問題に対しても有効であることがわかる．また，選択操作にバイアスをかけた並列 MOGA 回路は，通常選択のみを用いる並列 MOGA 回路より，

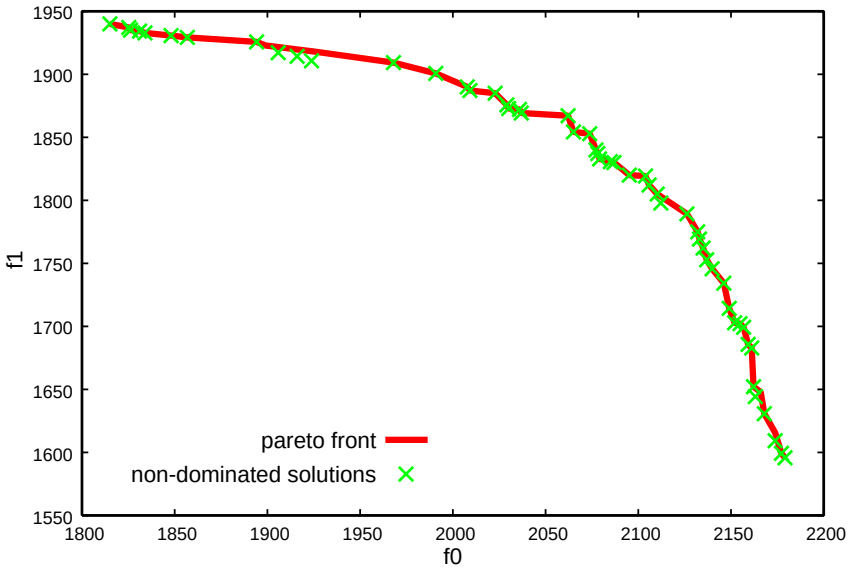

図 17 パレート最適解: normal (1)

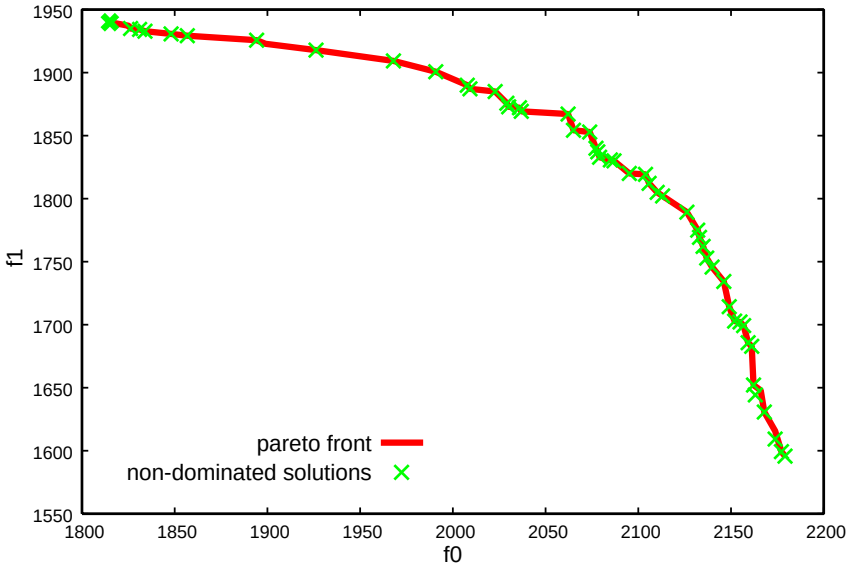

図 18 パレート最適解: none(1)

優れた性能を示すことが確認できる．

最後に提案手法と NSGA-II の比較を行った．ここでは NSGA-II はソフトウェアとして実装した．gcc version 3.35，最適化オプション O3 を用いてコンパイルを行い，Pentium 4（2.4GHz），256MB memory，linux 2.6.10 上で実行した．また，予備実験より NSGA-II のパラメタとして個体数は 64，交叉率は 1.0，突然変異率は 0.04 を用いた．実験結果より，NSGA-II は 1 評価あたりの処理時間は 1.05×10^{-4} 秒であった．表 6 に，それぞれの手法で得られたパレート最適解の数が約 30，40 の時の島あたりの評価回数と処理時間を示す．ここでの “found solutions” は，最終的に得られたパレート最適解の数を示している．評価時間は，提案アーキテクチャをハードウェア上に実装したものとして計算を行っている．表 6 より，提案アーキテクチャによる MOGA 回路は並列度が上がるにつれ性能が向上していることがわかる．しかし，評価回数あたりの性能比較では，いずれの場合でも提案手法は NSGA-II より劣っていることがわかる．これは，提案手法で採用された重複個体排除機構が NSGA-II で採用されてているランク戦略と混雑度を用いた選択機構より性能が劣っているためであると考えられる．しかし，充分に時間をかけた場合，最終的に発見されたパレート最適解の個数より提案手法は NSGA-II とほぼ同等の性能かより優れた性能を示すことがわかる．また，提案手法をハードウェア化し，パイプライン処理で動作させた場合，提案手法は NSGA-II と比較して，提案手法は NSGA-II より約 636〜3,270 倍優れた性能を示すことが分かった．

表 6 提案手法の比較結果

selection module	found solutions	lap time (30 solutions)		lap time (40 solutions)	
		evaluations	time (msec.)	evaluations	time (msec.)
normal (1)	45.6	182,000	2.12	542,000	6.32
normal (2)	50.3	192,000	2.52	418,000	5.48
normal (3)	50.8	76,000	0.931	208,000	2.55
normal (4)	51.5	87,000	1.13	227,000	2.95
normal (6)	52.0	88,000	1.03	178,000	2.08
none (1)	43.1	355,000	7.65	586,000	12.6
none (2)	50.1	252,000	3.38	400,000	5.36
none (3)	49.6	243,000	2.80	427,000	4.91
none (4)	50.7	186,000	2.38	284,000	3.63
none (6)	51.3	185,000	2.37	294,000	3.76
biased (3)	50.5	64,000	0.784	124,000	1.52
biased (4)	51.4	49,000	0.637	121,000	1.57
biased (6)	52.0	42,000	0.491	97,000	1.13
NSGA-II	43.8	96,000	10.1×10^3	352,000	37.0×10^3

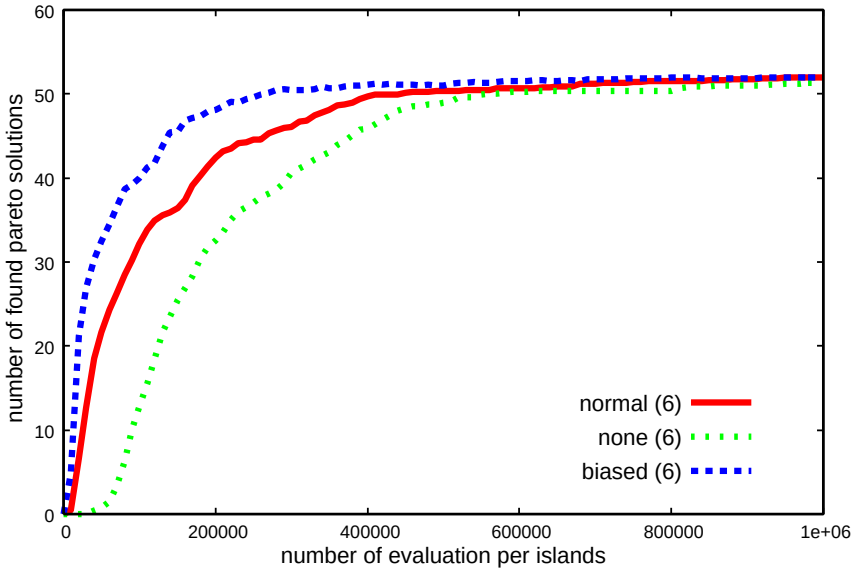

図 19 並列度 6 における得られた多目的最適解の数

3.5 まとめ

本章では，MOGA 回路を実装するためのアーキテクチャを提案した．提案アーキテクチャでは，ハードウェア化に適した重複個体を排除するための選択操作と島ごとに異なる部分のパレート最適解を探索するための並列アーキテクチャを含む．提案アーキテクチャによる MOGA 回路の性能を調査するため，提案アーキテクチャによる MOGA と既存の手法である NSGA-II とを比較した．結果，提案手法は NSGA-II より最大約 3,270 倍優れた性能を示すことを確認した．

4. まとめ・結論

本論文では，FPGA 上への適切な SOGA 回路を実装するための手法とその設計を支援するための手法を提案した．また，MOGA のハードウェア実装に適したアーキテクチャおよび実装手法を提案した．

提案アーキテクチャによる GA 回路の性能を確認するため，それぞれソフトウェア上で実装した一般的な SOGA および MOGA との比較を行った結果，提案手法で実装された回路は，最大約 3,270 倍優れた性能を示すことを確認した．最後に，回路規模予測モデルの有効性を確認するために，提案アーキテクチャにより作成した SOGA 回路の回路規模の予測値と実測値を比較した．その結果，回路規模予測モデルによる予測値と実測値の誤差は最大 3%に収まることを確認した．これらのことより提案したアーキテクチャおよび回路規模予測モデルは様々な組合せ最適化問題を様々な FPGA デバイス上に実装する際に有効であると考えられる．

本論文で提案した手法を用いることで，組込みシステム等で用いる GA チップを作成する際に大幅に開発期間や開発費用を削減することができ，より安価な GA チップを市場に供給することができる．その GA チップを搭載することで，容易に GA を利用したアプリケーションをリソースの少ない携帯などの情報機器で使用できる．その結果，GA を利用したルート探索やスケジュール探索などの人の行動をサポートするアプリケーションをいつでも，どこでも使用することができ，その時点における最適な行動の指針を知ることができるようなると考えている．

本論文で提案した重複個体排除機構は，ハードウェア化した際の回路規模が比較的大きいため，回路規模が小さい FPGA 上に実装することは困難である．そのため，実際に提案した MOGA 回路を実装するためには，より回路規模を小さく実装するための手法を今後開発していく必要があると考える．

また，その他の今後の課題として，将来的に本研究を発展させ，より GA 回路の設計および実装を容易にするため，解のコーディング，交叉，突然変異，評価，選択のアルゴリズムを記述するための高水準言語を設計し，RT レベル回路記述生成ツールを作成すること，論理合成を行なうことなく消費電力を予測するモデルを考案すること，提案アーキテクチャによる GA 回路を進化型ハードウェアに組込み，その進化型ハードウェアの学習機構として利用することを行いたいと考

えている.

謝辞

本論文作成及び研究全般にわたり，適切な御指導および御助言を賜わりました伊藤実教授に厚く御礼申しあげます.

また，本論文の作成および研究にわたり有益な情報や御助言を賜わりました藤原秀雄教授に厚く御礼申しあげます.

本論文の作成および研究全般に，適切な御指導および御助言を賜りました安本慶一助教授に厚く御礼申しあげます.

本論文の作成および日頃から研究，英語の論文の作成にあたり終始指導して頂いた柴田直樹滋賀大経済学部助教授に厚く御礼申しあげます.

本論文の作成および遺伝的アルゴリズム全般に関してアドバイスして頂いた村田佳洋助手に深く感謝します.

また，研究に関してのみならず，研究室の生活において手助けして頂いた木谷友哉助手に深く感謝します.

最後に日頃から公私に渡り，お世話になりました伊藤研究室の皆様に感謝します.

参考文献

[1] Cyclone FPGA Family Data Sheet. http://www.altera.co.jp/literature/hb/cyc/cyc_c5v1_01.pdf.

[2] Cyclone II デバイス・ファミリ・データ・シート. http://www.altera.co.jp/literature/hb/cyc2/cyc2_cii5v1_01_j.pdf.

[3] Processor spec finder. http://www.univ-valenciennes.fr/ROAD/MCDM/ListMOKP.html.

[4] Chatchawit Aporntewan and Prabhas Chongstitvatana. A Hardware Implementation of the Compact Genetic Algorithm. *Proceedings of the 2001 Congress on Evolutionary Computation (CEC2001)*, pp. 624–629, 2001.

[5] Erick Cantú-Paz. A Survey of Parallel Genetic Algorithms. *Technical Report IlliGAL 97003, University of Illinois at Urbana-Champaign*, 1997.

[6] Deepti Chafekar, Jiang Xuan, and Khaled Rasheed. Constrained multi-objective optimization using steady state genetic algorithms. *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, pp. 813–824, 2003.

[7] Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pp. 849–858, 2000.

[8] Xavier Gandibleux. Mcdm numerical instances library. http://www.univ-valenciennes.fr/ROAD/MCDM/ListMOKP.html.

[9] Paul Graham and Brent Nelson. A Hardware Genetic Algorithm for the Traveling Salesman Problem on SPLASH 2. *Proceedings of Field-Programmable Logic and Applications*, pp. 352–361, 1995.

[10] Tetsuya Higuchi Hidenori Sakanashi, Masaya Iwata. Lossless Compression of Very High Resolution Bi-level Images Using Genetic Algorithm. *Journal of Information Processing Society of Japan*, Vol. 45, No. 5, pp. 1460–1470, 2004.

[11] Tomoya Kitani, Yoshifumi Takamoto, Keiichi Yasumoto, Akio Nakata, and Teruo Higashino. A Flexible and High-Reliable HW/SW Co-Design Method for Real-Time Embedded Systems. *Proceedings of 25th IEEE Int'l. Real-Time Systems Symp. (RTSS2004)*, pp. 437–446, 2004.

[12] Li Chunlin Li Layuan. Genetic Algorithm-Based QoS Multicast Routing for Uncertainty in Network Parameters. *Proceedings of Web Technologies and Applications, 5th Asian-Pacific Web Conference (APWeb 2003)*, pp. 430–441, 2003.

[13] Sadiq M. Sait and Habib Youssef. *Genetic Algorithms(GAs) Iterative Computer Algorithms with Applications in Engineering*, pp. 109–181. THE IEEE COMPUTER SOCIETY, 1999.

[14] Hiroshi Satoh, Isao Ono, and Shigenobu Kobayashi. Minimal Generation Gap Model for GAs Considering Both Exploration and Exploitation. *Proceedings 4th Int'l Conf. on Soft Computing (IIZUKA'96)*, pp. 494–497, 1996.

[15] Barry Shackleford, Etsuko Okushi, Mitsuhiro Yasuda, Hisao Koizumi, Katsuhiko Seo, Takahashi Iwamoto, and Hiroto Yasuura. High-performance hardware design and implementation of genetic algorithms. In *Hardware implementation of intelligent systems*, pp. 53–87. Physica-Verlag GmbH, 2001.

[16] Hisashi Shimodaira. An Empirical Performance Comparison of Niching Methods for Genetic Algorithms. *IEICE Trans. Inf. & Syst.*, Vol. E85-D, No. 11, pp. 1872–1880, 2002.

[17] Brad T. Skinner, Hung T. Nguyen, and Dikai K. Liu. Performance study of a multi-deme parallel genetic algorithm with adaptive mutation. *Proceesdings of the 2nd International Conference on Autonomous Robots and Agents*, pp. 88–94, 2004.

[18] Tatsuhiro Tachibana, Yoshihiro Murata, Naoki Shibata, Keiichi Yasumoto, and Minoru Ito. Flexible Implementation of Genetic Algorithms on FPGA. *Proceedings of Fourteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA2006)*, p. 236, 2006.

[19] Tatsuhiro Tachibana, Yoshihiro Murata, Naoki Shibata, Keiichi Yasumoto, and Minoru Ito. General Architecture for Hardware Implementation of Genetic Algorithm. *Proceedings of Field-Programmable Custom Computing Machines (FCCM'06)*, pp. 291–292, 2006.

[20] Tatsuhiro Tachibana, Yoshihiro Murata, Naoki Shibata, Keiichi Yasumoto, and Minoru Ito. Hardware Implementation Method of Multi-Objective Genetic Algorithms. *Proceedings of IEEE Congress on Evolutionary Computation (CEC2006), (DVD-ROM)*, 2006.

[21] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

[22] 丸山敦史, 柴田直樹, 村田佳洋, 安本慶一, 伊藤実. P-tour：観光スケジュール作成支援とスケジュールに沿った経路案内を行うパーソナルナビゲーションシステム. 情報処理学会論文誌, Vol. 12, No. 45, pp. 2678–2687, 2004.

[23] 小野巧, 佐藤浩, 小林重信. サブシーケンス交換交叉と GT 法に基づくジョブショップスケジューリングの進化的解法. 電子情報通信論文誌, Vol. 117-C, No. 7, pp. 888–895, 1997.

[24] 清田高徳, 辻康孝, 野田理, 近藤英二. 遺伝的アルゴリズムを用いた多目的ファジィ満足化手法による 1 型サーボ系の一設計法. 日本機械学会論文集 C 編, Vol. 70, No. 696, pp. 2392–2398, 2004.

[25] 若林真一, 小出哲士, 八田浩一, 中山喜勝, 後藤睦明, 利根直佳. 交差手法の適応的選択機能を組み込んだ遺伝的アルゴリズムの LSI チップによる実現. 情報処理学会論文誌, Vol. 41, No. 6, pp. 1766–1776l, 2000.

[26] 前川景示, 玉置久, 喜多一, 西川示韋一. 遺伝的アルゴリズムによる巡回セールスマン問題の一解法. 計測自動制御学会論文集, Vol. 31, No. 5, pp. 598–605, 1995.

[27] 橘達弘, 村田佳洋, 柴田直樹, 安本慶一, 伊藤実. FPGA 上への遺伝的アルゴリズムの柔軟な実装手法の提案. 電子情報通信学会論文誌, Vol. J89-D, No. 6, pp. 1182–1191, 2006.

[28] 廣安知之, 廣安博之, 三木光範, 渡邊真也, 上浦二郎. 現象論モデルと遺伝的アルゴリズムによるディーゼルエンジン燃焼効率の多目的最適化. 自動車技術会論文誌, Vol. 35, No. 1, pp. 51–56, 2004.

[29] 大林茂. 航空機の多目的最適設計. 人工知能学会誌, Vol. 18, No. 5, pp. 495–510, 2003.

[30] 北浦理, 浅田英昭, 松崎元昭, 河合隆光, 安藤秀樹, 島田俊夫. パイプラインストールを除去した遺伝的アルゴリズム専用ハードウェア. 計測自動制御学会論文集, Vol. 35, No. 11, pp. 1496–1504, 1999.

[31] 小林亮一, 阿部正英, 川又政征. 遺伝的アルゴリズムを用いた不連続閉曲線抽出手法の FPGA 上での実現. 電子情報通信学会技術研究報告, no.CAS2000-131,DSP2000-189,CS2000-151, pp. 29–36, 2001.

業績リスト

論文誌

1. 橘 達弘，村田 佳洋，柴田 直樹，安本慶一，伊藤 実：FPGA 上への遺伝的アルゴリズムの柔軟な実装手法の提案，電子情報通信学会論文誌，Vol.J89-D, No.6，pp.1182-1191，2006 年 6 月.

国際会議

1. Tatsuhiro Tachibana, Yoshihiro Murata, Naoki Shibata, Keiichi Yasumoto and Minoru Ito: Flexible Implementation of Genetic Algorithms on FPGA, Proceedings of Fourteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA2006), p.236, Poster session, February, 2006.

2. Tatsuhiro Tachibana, Yoshihiro Murata, Naoki Shibata, Keiichi Yasumoto and Minoru Ito: General Architecture for Hardware Implementation of Genetic Algorithm, Proceedings of Field-Programmable Custom Computing Machines (FCCM'06), pp.291–292, Poster session, April, 2006.

3. Tatsuhiro Tachibana, Yoshihiro Murata, Naoki Shibata, Keiichi Yasumoto and Minoru Ito: Hardware Implementation Method of Multi-Objective Genetic Algorithms, Proceedings of IEEE Congress on Evolutionary Computation (CEC2006), (DVD-ROM), July, 2006.

研究会

1. 橘 達弘，村田 佳洋，柴田 直樹，伊藤 実：探索空間の地形を判断する親子エージェントを用いた組合せ最適化アルゴリズム，夏のＬＡシンポジウム，2003 年 7 月．

2. 橘 達弘，村田 佳洋，柴田 直樹，伊藤 実：探索空間の地形を判断するエージェントを用いた組合せ最適化アルゴリズム，情報科学技術フォーラム（FIT 2003）講演論文集 （CD-ROM），2003 年 9 月．

3. 橘 達弘，村田 佳洋，柴田 直樹，安本慶一，伊藤 実：巡回セールスマン問題を対象とする遺伝的アルゴリズムの FPGA 上への実装，情報科学技術フォーラム（FIT 2004）講演論文集，(CD-ROM)，2004 年 9 月．

4. 橘 達弘，村田 佳洋，柴田 直樹，安本慶一，伊藤 実：FPGA 上への GA 回路設計支援ツールの提案，電子情報通信学会，第 2 回リコンフィギャラブルシステム研究会 (RECONF)，RECONF2005-42，pp.7–12，2005 年 9 月．