計算機アーキテクチャ 141－12
（２００１．１．２６）

遺伝的アルゴリズムの高速実行に適した命令セットを持つRISCプロセッサDLX-GA

小泉 慎哉[†] 若林 真一[†] 小出 哲士[†‡] 井村 紀道[†] 藤原 一成[†]
[†]広島大学工学部
〒739-8527 東広島市鏡山一丁目4番1号
[‡]東京大学大規模集積システム設計教育研究センター
〒113-8656 文京区本郷7-3-1
E-mail:{koi, wakaba}@ecs.hiroshima-u.ac.jp, koide@vdec.u-tokyo.ac.jp

本稿では，遺伝的アルゴリズム(GA)の実行における計算時間の短縮を目的として，任意のGAを高速に実行可能なRISCプロセッサDLX-GAを提案する．提案プロセッサDLX-GAは，DLXアーキテクチャをベースとしたRISCプロセッサであり，GAの実行において多用されるビット演算命令や乱数発生命令，SIMD型命令等をサポートし，これらを6段のパイプラインで処理することによりGA実行の高速化を実現する．GAは，提案プロセッサ上にソフトウェアとして実現するため，任意のGAが実行可能となる．提案プロセッサの命令セットを用いると，2点交差等の遺伝オペレータの実行に要するクロック数の90%以上の減少が達成可能である．

A RISC Processor DLX-GA with Instruction Set Suitable for High-Speed Execution of a Genetic Algorithm

Shinya KOIZUMI[†], Shin'ichi WAKABAYASHI[†], Tetsushi KOIDE[†‡],
Norimichi IMURA[†] and Kazunari FUJIWARA[†]
[†]Faculty of Engineering, Hiroshima University
4-1, Kagamiyama 1 chome, Higashi-Hiroshima 739-8527 JAPAN
[‡]VLSI Design and Education Center, The University of Tokyo
7-3-1, Hongo, Bunkyo-ku, Tokyo 113-8656 JAPAN
E-mail:{koi, wakaba}@ecs.hiroshima-u.ac.jp, koide@vdec.u-tokyo.ac.jp

This paper proposes a new RISC processor for high speed execution of genetic algorithms (GAs). The proposed RISC processor is designed based on the DLX architecture, and a new instruction set which is effective for high-speed execution of GAs is adopted. Since a GA is implemented as software on the proposed processor, any type of GA can be realized. Using the instruction set of the proposed processor, more than 90% reduction of the number of clocks to execute GA operators such as 2-point crossover can be achieved.

1 はじめに

遺伝的アルゴリズム (Genetic Algorithm, GA) は，自然界の遺伝メカニズムに基づく探索アルゴリズムとして，1970 年代に John Holland によって開発された [1]．GA は VLSI 設計自動化や画像処理，スケジューリング問題等の工学の様々な分野における多くの大規模最適化問題を効率良く解くヒューリスティック手法として知られている．

しかしながら，GA は優れた解探索能力を持つ一方で，汎用逐次処理コンピュータ上に実現し実行した場合，複雑な問題に対しては多大の計算時間を要するという問題点を持つ．この問題点の改善策として，いくつかのアプローチが報告されている．まず 1 つが並列 GA である．一般に，並列 GA は良いパフォーマンスを示している．さらに，並列 GA は通常はソフトウェアとして実現されるため，任意の GA を実現するのが容易である．一方，並列 GA の主な問題点としては，大量のハードウェア資源を必要とするということが挙げられる．

GA の高速実行のもう 1 つのアプローチは，ハードウェア化である．例えば，Scott らは再構成可能な FPGA を用いて定常状態 GA を実現するハードウェア GA を提案している [6]．Yoshida らは粗粒度の並列処理を実現した GA 向けの VLSI を提案している [9]．著者らも適応的遺伝的アルゴリズムのハードウェア化について研究し，実際に Genetic Algorithm Accelerator (GAA) として LSI チップ化している [8]．GA のハードウェアとしての実現は，一般にとても良いパフォーマンスを達成している．しかし，このアプローチの主な問題点は，ハードウェアは選択・交差・突然変異等の遺伝オペレータが前もって固定されているため，適用可能な問題が限られているということである．

そこで本研究では，GA の効率的な実行に適した命令セットを持つ新しい RISC プロセッサ DLX-GA を提案する．提案プロセッサ DLX-GA は，DLX アーキテクチャ [3] をベースとして設計されており，ロード/ストアや算術論理演算等の通常の命令に加えて，GA の実行において多用されるビット演算命令や乱数発生命令，SIMD 型命令など，GA の高速実行に有効な命令を備えている．そして，これらの命令を 6 段のパイプラインで処理することにより高速な命令実行を実現する．また，プロセッサには，並列 GA として動作させる場合における他のプロセッサとの通信のための割込み機構を備え，さらに処理の高速化のためにオンチップの命令キャッシュを備えている．このようなシステム構成により任意の GA を高速に実行可能とする．

以降では，**2.** で遺伝的アルゴリズムとその特性，**3.** で提案プロセッサ DLX-GA のアーキテクチャ，**4.** で DLX-GA のパフォーマンスの見積り，**5.** でまとめと今後の課題について述べる．

2 準備

2.1 遺伝的アルゴリズム GA

一般的な GA は以下の流れに沿って解の探索を行なう．まず，解こうとする問題の候補解をコード化し，染色体として表現する．そして決められた個体数の染色体をランダムに生成し，初期集団を作り，各々の個体がどれくらい問題の解として適しているかを示す適応度の評価値を計算する．この適応度評価値の計算方法は解こうとする問題により異なっており，問題に依存する部分である．そして，この個体集団からランダムに個体を選びだし，これを親として交差・突然変異等の遺伝的操作を施して新しい個体 (子孫) を生成する．基本的に適応度の高い個体がより多くの子孫を残す仕組みになっている．

以下では簡単にこれらの遺伝的操作について説明する．交差とは図 1 のように親として選ばれた二つの個体に対し乱数で選ばれたポイントを境にビットの交換を行う操作である．図 1では 3 ビット目から 5 ビット目の間の 2 点間で交差を行っている．このような交差手法を 2 点交差と呼ぶ．突然変異とはビットを一定の確率で変化させる操作であり，図 1 では子孫 1 の 2 ビット目が突然変異を起こしてビットが反転している．

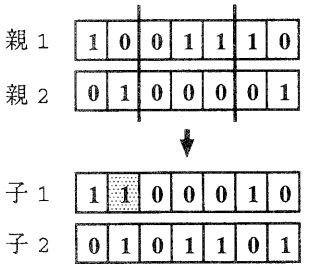

図 1 2 点交差，突然変異

このようにして生成された新たな個体はその適応度評価値が計算され，個体集団の中に組み込まれる．その後，個体集団に対して適応度評価値の優劣に応じて個体が一定数残るように取捨選択し，次世代の集団を形成する．これら一連の流れを規定回数，もしくは求める解が出るまで繰り返す．

上で述べたように GA で特定の問題を解く際には，複雑な個体表現と，解空間を効率的に探索するよう設計された遺伝オペレータを用いる必要がある．このため，汎用 GA システムを開発する場合は，問題に応じて任意の遺伝オペレータを実現し，効率良く実行可能なメカニズムが必要となる．

一般に，計算機械におけるプログラム可能性 (Programmability) は，ソフトウェア，もしくは再構成可能な論理デバイスを用いると実現可能である．GA

をソフトウェアで実現すると，任意のGAがプログラム可能であるが，通常の汎用プロセッサではビット演算や乱数発生を頻繁に行うといったGAの特性のために高いパフォーマンスを得ることは困難である．一方，ハードウェアベースのGAでプログラム可能性を実現するのに，FPGAのような再構成可能な論理デバイスを用いてGAエンジンを構築する方法もあり，FPGAを用いた汎用GAシステムの実現についてもいくつか報告されている [2]．このアプローチの主な欠点は，その多大な設計期間である．FPGA上にGAエンジンを設計することは，汎用プロセッサ上にGAをソフトウェアで実現することよりも難しい．ハードウェアの設計は，ハードウェア記述言語 (HDL) を用いたコーディングを行うだけでなく，タイミング制約と資源の制約の下で配置配線も行わなければならない．さらに，通常HDLコーディングのレベルは，C言語のような高級言語を用いたプログラミングのレベルよりも低い．

本稿では，GA実行のハイパフォーマンスと共に完全なプログラム可能性を実現する新しいRISCプロセッサを提案する．任意のGAが提案プロセッサ上にソフトウェアで実現可能であるため，完全なプログラム可能性が保証される．

2.2 GAの特性

GAの効率的な実行を実現するプロセッサアーキテクチャを得るために，我々はまずGAの一般的な特性について考察し，GA実行時に頻繁に使用されるオペレータのタイプを吟味した．

1. ビット指向のオペレーション
   GAは一般的にAND, ADDのような通常のワード単位のオペレーションだけでなく，ビット指向のオペレーションも頻繁に使用する．つまり，遺伝的操作はメモリワード全体に対してだけでなく，あるワードのある部分に対しても適用される．例えば，nビットの染色体に2点交差が適用されるとき，親の染色体は3つの部分に分けられ (図1)，中間の部分が交換される．通常，交差ポイントはビット位置で指定される．
2. 高頻度な乱数の使用
   GAは選択・交差・突然変異などのアルゴリズムの実行における様々なステージで乱数を頻繁に使用する．
3. SIMD的オペレーション
   一般に，GAでは染色体のセットが人口を構成し，同じ遺伝的操作が人口中のすべての染色体に対して適用される．これらの操作はSIMD (Single Instruction Stream, Multiple Data Stream) 操作とみなすことができる．

GAをPentiumやUltraSPARC等の汎用プロセッサ上にソフトウェアとしてインプリメントする際，上に述べた特性のため，効率的なGAをインプリメントしようとする場合にはいくつかの困難に直面する．まず最初に，汎用のプロセッサはワード単位のオペレーションを目的として設計されているため，ビット指向のオペレーションを実現するには効率が良くない．

2つ目に，GAをソフトウェアでインプリメントするとき，乱数は疑似乱数発生アルゴリズムを用いて発生させるが，疑似乱数発生アルゴリズムは最小でも10命令，一般的には50以上の命令で実現される．GAの実行において乱数は頻繁に使用されるため，乱数発生によるパフォーマンスのオーバーヘッドは無視することができない．

3つ目に，交差などのいくつかの遺伝的操作はSIMD型の命令で実現することが可能であるが，汎用プロセッサはSIMD型の命令をサポートしていない．

3 提案プロセッサ DLX-GA

3.1 命令セットアーキテクチャ

前章で述べたGAの特性の考察結果に基づいて，提案プロセッサの命令セットは，GAのソフトウェアでの高速実行を可能とするような命令セットにした．算術論理演算命令やロード/ストア命令など通常の汎用プロセッサがサポートする命令もまた必要とされるので，DLXアーキテクチャ [3] をベースアーキテクチャとして採用し，浮動小数点命令を除くすべてのDLX命令をサポートしている．DLXの命令セットに加えて，GAを効率的に実行可能とするような30の新しい命令をサポートした．これらの命令は以下の3つのカテゴリに分類される．以下に各カテゴリに属する命令のうちのいくつかについて説明する．

1. ビット演算命令
   このカテゴリに属する命令は，1ワード中の指定された数ビットをオペランドとして扱う．このカテゴリには算術論理演算やMOVEなどを含む22個の新たな命令を考案した．

   `move_bits [rs][rd][ra][rb][rc]`

   この命令はソースレジスタの指定した数ビットをデスティネーションレジスタ内の指定した位置に移動させる．即ち，rd[rb:rb+rc-1] ← rs[ra:ra+rc-1]．

   `and_bits [rs][rd][ra][rb][rc]`

   この命令はソースレジスタとデスティネーションレジスタの指定した数ビットについての論理積を取る．即ち，rd[rb:rb+rc-1] ← rs[ra:ra+rc-1] & rd[rb:rb+rc-1]．

2. 乱数発生命令
このカテゴリには乱数発生に関する 4 つの命令がある．次節で説明するが，提案プロセッサはクロックサイクル毎に 96 ビットの疑似乱数を生成する疑似乱数ジェネレータをプロセッサ内部に持っている．

`reset_rng [ra][rb][rc]`

この命令はレジスタ ra, rb, rc に格納されている数を乱数ジェネレータの初期値として乱数ジェネレータを初期化する．

`set_rand_num [rd][imd]`

この命令は範囲が 0〜imd-1 の乱数をレジスタ rd にセットする．

3. SIMD 型命令
このカテゴリには 4 つの SIMD 型命令が属する．

`rotate_r [rs1][rs2][imd]`

この命令はレジスタ rs1 と rs2 の内容を同時に imd ビット右方向に回転する．

`exchange_rotate_r [rs1][rs2][imd]`

この命令はレジスタ rs1 と rs2 の内容を同時に imd ビット右方向に回転し，rs1 の内容を rs2 に，rs2 の内容を rs1 に格納する．

オリジナルの DLX アーキテクチャは，I 型，R 型，J 型の 3 つの命令フォーマットを持っている．上に述べた新しい命令をインプリメントするために，提案プロセッサには図 2に示される I' 型，R' 型，R" 型という新しい命令フォーマットを追加する．

3.2 マイクロアーキテクチャ

本稿で提案する RISC プロセッサ DLX-GA のシステム構成を図 3に示す．

DLX-GA は 32 ビット RISC プロセッサである．前節で述べたように，DLX-GA は DLX アーキテクチャ [3] に基づいて設計しているため，プロセッサのほとんどの特徴は DLX と同じである．DLX-GA は 32 ビットの固定長命令フォーマットを持つロード/ストア・アーキテクチャである．ハーバードアーキテクチャを採用しているため，プロセッサは命令メモリバスとデータメモリバスを別々に持っている．また，DLX-GA は処理の高速化のためにオンチップの命令キャッシュを持っている．しかし，ハードウェア資源の制約のためデータキャッシュは持っていない．そして，DLX-GA は並列 GA やオプションの外部評価回路との通信を考慮した 2 レベルの割込み機構も備えている．表 1に DLX-GA の仕様を示す．

DLX-GA におけるすべての命令の実行は，命令フ

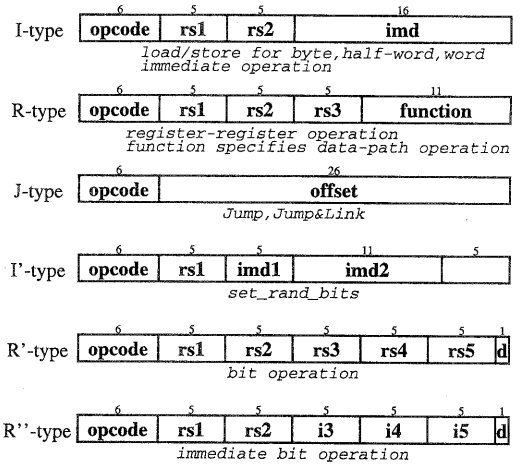

図 2 命令フォーマット

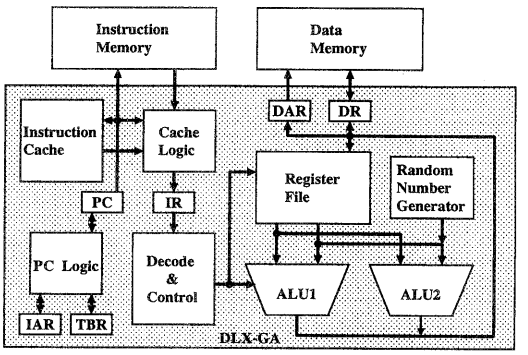

図 3 DLX-GA システム構成

表 1 DLX-GA の仕様

命令メモリアドレスバス	15 ビット
命令メモリデータバス	32 ビット
データメモリアドレスバス	23 ビット
データメモリデータバス	32 ビット
命令キャッシュ	直接マップ方式
	256 ライン
レジスタファイル	32 ビット × 32 ワード
割込み	2 レベル
クロック周波数	100MHz

ェッチ IF，命令デコード ID，実行 EX1，実行 EX2，メモリアクセス MEM，レジスタ書き込み WB の 6 つの基本ステージから成り，これらを 6 段のパイプラインで処理する．パイプラインストールへの対応としては，構造ハザードについては資源の追加を行っており，データハザードについてはフォワーディングを行っている．そして，制御ハザードについては命令デコード (ID) ステージで早期に分岐を行うようにし，それに伴う 1 つのディレイスロットには遅延分岐を採用し，ディレイスロットができないようにしている．

プロセッサのデータパスには，32 ビット × 32 ワードの汎用レジスタ群を実現したレジスタファイルと，SIMD 型の命令を実現するために 2 つの ALU が含まれている．さらに，データパスには線形フィードバックシフトレジスタに基づいた M 系列乱数ジェネレータ [4] が含まれている．この乱数ジェネレータは，96 ビットの疑似乱数を毎クロックサイクル生成する．乱数の精度についてであるが，GA で使用する乱数は適度な一様分布を示す乱数であれば，GA のパフォーマンスに及ぼす影響の差は無いということが文献 [5] で示されており，今回採用した M 系列乱数は適度な一様分布を示すことが文献 [4] で示されている．乱数ジェネレータにより生成された数を 0 から 1 までの固定小数点乱数とし，命令により与えられた定数との乗算を行い，所望の乱数とする．

3.3 設計手順と設計状況

提案プロセッサ DLX-GA は，ローム (株) 製の $4.93mm$ 角，信号ピン数 111 ピン，5 層配線 (PolySi: 2 層，メタル配線: 3 層)，CMOS $0.35\mu m$ テクノロジチップに実装する．ライブラリにはロームのパスポートライブラリを使用する．試作は VDEC(東京大学大規模集積システム設計教育研究センター) に依頼する．

回路設計は Verilog-HDL を用いてレジスタ転送レベルで記述することにより行い，各パイプラインステージをそれぞれ単独のモジュールとして構成し，トップモジュールでそれらのインタフェースを記述した．論理合成においては，後の配置配線で ALU やレジスタファイル等のデータパス系と，命令デコーダ等の制御系を分けて配置するためにフラットにしないで合成している．Verilog-HDL による回路記述は，コメントを含めて約 8000 行である．

シミュレーションは Cadence 社の Verilog-XL シミュレータ，論理合成は Synopsys 社の Design Compiler，配置配線は Avant!社の Apollo を用いて行う．

現在のところ，論理合成結果は，セル数 (キャッシュメモリを除く) が 17,099，ネット数 19,438，ゲート数は 2 入力 NAND ゲート換算で 47,648 となっている．

4 DLX-GA のパフォーマンス見積もり

現在，Verilog-HDL によるレジスタ転送レベルでのプロセッサ設計が終了しデバッグを行っているところで，シミュレーションによるパフォーマンスはまだ得られていない．代りに，ここでは DLX-GA の命令セットの有効性を示す．

4.1 ビット演算命令による速度向上

ここでは，DLX の命令セットを使用した場合と DLX-GA のビット演算命令を使用した場合とで GA の実行速度にどれくらいの差が出るかを，GA の代表的な交差手法である 2 点交差と巡回セールスマン問題用のスワップ突然変異とで見積もった．すなわち，ここでは 2 ワード (64 ビット) 中に 0 から 15 の数値 (4 ビット) の順列をコーディングしたデータに対する 2 点交差と，この順序表現のデータに対するスワップ突然変異を考えた．

図 4(a) に C 言語で記述した 2 点交差を DLX のアセンブリコードを得るための GNU C Compiler でコンパイルして生成したアセンブリコードを示す．DLX の命令セットにはビット演算命令が無いため，交差を行うにはマスクを作成する必要がある．図 4(a) において，左 4 列はマスク作成を行っているアセンブリコードである．そして，右端の 1 列でそのマスクを用いて 2 点交差を実現している．1 命令を 1 クロックで実行すると仮定すると，このコードを実行するためには 138 クロックが必要である．

次に，DLX-GA の命令セットを用いて実現した 2 点交差のアセンブリコードを図 4(b) に示す．DLX-GA プロセッサ対応の C コンパイラは現在開発中であり，まだ利用可能な C コンパイラがないため，ハンドコーディングによるアセンブリコードを示している．新しい命令を有効に利用することで，かなり少ない命令数で 2 点交差が実現できている．1 命令を 1 クロックで実行できるとすれば，このコードの実行に必要なクロック数は 11 クロックである．

表 2 クロック数の比較

プロセッサ	2 点交差	スワップ突然変異
DLX	138	75
DLX-GA	11	7

表 2に 2 点交差とスワップ突然変異に対しての両プロセッサの比較結果を示す．2 点交差とスワップ突然変異のどちらにおいても，90%以上のクロック数の減少が達成されている．尚，DLX の命令セットを用いたコードはコンパイラにより生成したものであるが，これをハンドコーディングでさらに短くすることは非常に困難であることを確認している．

```
L2_LF0:
    lw    r1,-32(r30)
    lw    r2,-36(r30)
    slt   r1,r1,r2
    bnez  r1,L5_LF0
    j     L3_LF0
L5_LF0:
    lw    r1,-32(r30)
    snei  r2,r1,#0
    beqz  r2,L6_LF0
    lw    r1,-20(r30)
    slli  r2,r1,#0x4
    sw    -20(r30),r2
L6_LF0:
    lw    r1,-20(r30)
    addi  r2,r1,#15
    sw    -20(r30),r2
```

```
L4_LF0:
    lw    r2,-32(r30)
    addi  r1,r2,#1
    add   r2,r0,r1
    sw    -32(r30),r2
    j     L2_LF0
L3_LF0:
    nop
    lw    r1,-36(r30)
    sw    -32(r30),r1
L7_LF0:
    lw    r1,-32(r30)
    lw    r2,-40(r30)
    slt   r1,r1,r2
    bnez  r1,L10_LF0
    j     L8_LF0
```

```
L10_LF0:
    lw    r1,-20(r30)
    slli  r2,r1,#0x1
    sw    -20(r30),r2
L9_LF0:
    lw    r2,-32(r30)
    addi  r1,r2,#1
    add   r2,r0,r1
    sw    -32(r30),r2
    j     L7_LF0
L8_LF0:
    nop
    lw    r1,-40(r30)
    sw    -32(r30),r1
```

```
L11_LF0:
    lw    r1,-32(r30)
    lw    r2,-44(r30)
    slt   r1,r1,r2
    bnez  r1,L14_LF0
    j     L12_LF0
L14_LF0:
    lw    r1,-20(r30)
    slli  r2,r1,#0x4
    sw    -20(r30),r2
    lw    r1,-20(r30)
    addi  r2,r1,#15
    sw    -20(r30),r2
L13_LF0:
    lw    r2,-32(r30)
    addi  r1,r2,#1
    add   r2,r0,r1
    sw    -32(r30),r2
    j     L11_LF0
```

```
L12_LF0:
    lw    r1,-12(r30)
    lw    r2,-20(r30)
    and   r1,r1,r2
    lw    r3,-20(r30)
    sub   r2,r0,r3
    subi  r2,r2,#1
    lw    r3,-16(r30)
    and   r2,r2,r3
    or    r1,r1,r2
    sw    -24(r30),r1
    lw    r1,-16(r30)
    lw    r2,-20(r30)
    and   r1,r1,r2
    lw    r3,-20(r30)
    sub   r2,r0,r3
    subi  r2,r2,#1
    lw    r3,-12(r30)
    and   r2,r2,r3
    or    r1,r1,r2
    sw    -28(r30),r1
```

(a) DLXの命令セットを用いた2点交差

```
    lw    r3,-16(r30)
    lw    r4,-20(r30)
    sub   r4,r4,r3
    lw    r5,-24(r30)
    multi r3,r3,r5
    multi r4,r4,r5
    lw    r1,-8(r30)
    lw    r2,-12(r30)
    move_bits r1,r2,r3,r3,r4
    sw    -8(r30),r1
    sw    -12(r30),r2
```

(b) DLX-GAの命令セットを用いた2点交差

図 4 2点交差のアセンブリコードの比較

4.2 乱数発生命令による速度向上

単純 GA [7] の実行において，選択，交差，突然変異，評価，乱数発生が全実行時間に占める割合を観察した結果を表 3に示す．表において，問題 1 は x^{10}の最大化，問題 2 は 32 次元ベクトルの長さ最大化である．表より乱数発生が全実行時間の約半分を占めていることが分かる．汎用プロセッサ上で乱数を発生する場合，通常 50 以上の命令を要するが，DLX-GA は 1 クロックで乱数を発生することが可能である．このことからも DLX-GA 上での GA 実行に対してパフォーマンスの多大の向上が期待できる．

表 3 SGA 実行における各操作の占める割合

	選択	交差	突然変異	評価	乱数発生
問題 1	10.8	13.6		19.4	56.2
問題 2	11.3	15.7		20.5	52.5

(条件：世代数 100，個体数 100，個体長 100 ビット，交差確率 0.6，突然変異確率 0.01，一点交差)

5 あとがき

本稿では任意の GA を高速に実行可能な RISC プロセッサ DLX-GA を提案した．また，GA の実行時間の大部分を占める交差を実際にアセンブリコードで記述し，どのくらいの速度増加が見込まれるかを見積もることにより命令セットの有効性を示した．今後はシミュレーションによるパフォーマンスの評価とチップのレイアウト設計を進めていく．

謝辞:本研究の一部は平成 12 年度科学研究費補助金基盤研究 (C) (2) (課題番号 12838008) による．

参考文献

[1] D. E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company (1989).

[2] P. Graham and B. Nelson: "A hardware genetic algorithm for the traveling salesman problem on Splash 2," *in Field Programmable Logic and Applications*, ed. W. Moore and W. Luk, pp. 352-361, Springer (1995).

[3] J. L. Hennessy and D. A. Patterson: *Computer Architecture: A Quantitative Approach, 2nd Edition*, Morgan Kaufmann Publishers, Inc. (1995).

[4] 柏木 ひろし: "M 系列とその応用," 昭晃堂 (1996).

[5] M. M. Meysenburg and J. A. Foster: "Randomness and GA Performance, Revisited," *Proc. Genetic and Evolutionary Computation Conference*, pp. 425-432 (1999).

[6] S. D. Scott, A. Samal and S. Seth: "HGA: A hardware-based genetic algorithm," *Proc. ACM/SIGDA, 3rd International Symp. on FPGAs*, pp. 53-59 (1995).

[7] R. E. Smith, D. E. Goldberg and J. A. Earickson: "SGA-C: A C-language Implementation of a Simple Genetic Algorithm," TCGA Report No. 91002, Department of Engineering Mechanics, The University of Alabama (1994).

[8] S. Wakabayashi, T. Koide, K. Hatta, Y. Nakayama, M. Goto and N. Toshine: "GAA: A VLSI genetic algorithm accelerator with on-the-fly adaptation of crossover operators," *Proc. International Symp. on Circuits and Systems*, Vol. 2, pp. 268-271 (1998).

[9] N. Yoshida, T. Yasuoka and T. Moriki: "Parallel and distributed processing in VLSI implementation of genetic algorithms," *Proc. 3rd International ICSC Symp. on Soft Computing*, pp. 450-454 (1999).