設 計 自 動 化 89－8
(1998. 9. 22)

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の実現

北浦 理　杉浦 弘幸　川合 隆光
安藤 秀樹　島田 俊夫
名古屋大学大学院工学研究科電子情報学専攻
〒464-8603 名古屋市千種区不老町
Tel 052-789-2727
E-mail : {kitaura,sugiura,kawai,ando,shimada}@shimada.nuee.nagoya-u.ac.jp

あらまし　遺伝的アルゴリズム(Genetic Algorithm:GA)は、大規模な最適化問題の有効な解法の一つとして知られている。GA の計算時間を短縮する試みとして、これまでにいくつかの GA エンジンが提案されてきた。H^3エンジンは従来の GA エンジンにおいて全計算時間の大半を占めていたパイプラインストールを除去した GA エンジンである。H^3エンジンは同一の処理を行うソフトウェアに比べ約 500 倍高速であることがシミュレーションにより示されている。
我々は H^3 エンジンを FPGA に実装し、性能を評価した。H^3 エンジンは同一の問題を解くソフトウェアに比べて約 750 倍の速度を得た。製作した H^3 エンジンのハードウェア量を測定した結果およそ 35K ゲート相当であった。

キーワード　遺伝的アルゴリズム，専用ハードウェア，パイプラインストール，FPGA

Implementation of a GA Engine without Pipeline Stalls

Osamu Kitaura　Hiroyuki Sugiura　Takamitsu Kawai
Hideki Ando　Toshio Shimada
Department of Information Electronics,
Graduate School of Engineering, Nagoya University
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan.
Tel +81-52-789-2727
E-mail : {kitaura,sugiura,kawai,ando,shimada}@shimada.nuee.nagoya-u.ac.jp

Abstract　Genetic Algorithms(GAs) are effective for large-scale optimization problems. Several attempts to make GA engines to reduce computation time of GAs have been made. H^3 engine is a GA engine that reduces computation time in GAs remarkably by eliminating pipeline stalls. The software simulation results for H^3 engine have shown that it performs about 500 times faster than software GAs.
We implemented H^3 engine on FPGAs and evaluated the performance. H^3 engine was proven to operate about 750 times faster than software GAs. The hardware amount of H^3 engine was equivalent to about 35K gates.

key words　Genetic Algorithm, Custom Computing Machine, Pipeline Stall, FPGA

1. はじめに

遺伝的アルゴリズム (Genetic Algorithm:GA)[1)2)]は生物の進化を工学的にモデル化した探索アルゴリズムである。GA は巡回セールスマン問題や配置問題など組合せ最適化問題を始めとし、さまざまな分野に応用されている。しかし GA の処理は対象とする問題によっては計算量が非常に膨大になるため、実用的時間で解を得ることが難しい。計算時間を短縮するためには、GA を高速実行する専用ハードウェア、すなわち **GA エンジン**が必要である。

これまでいくつかの GA エンジンが提案された[3)4)5)6)]。これらは、各処理をハードウェア化することにより、計算時間を短縮しているが、全実行時間の大半をパイプラインストールが占めている。これに対し、浅田らはパイプラインストールを除去した GA エンジンとして **H^3エンジン**[7)8)]と呼ぶアーキテクチャを提案した。H^3エンジンには次の 2 つの特徴がある。

- 各世代の 1 個体のみを置き換える **steady state GA**[9)10)]を用いる。
- ルーレット選択を行う際、2 分探索と線形探索を併用する。

H^3エンジンは、基本的な GA である**単純遺伝的アルゴリズム (Simple GA:SGA)** に基づく従来の GA エンジンに対して 17～60 倍、ソフトウェアで実行した SGA に対して約 500 倍高速であることがソフトウェアシミュレーションによって示されている。

我々は H^3エンジンを Field Programmable Gate Array (FPGA) 上に実装し、H^3エンジンの実機の評価を行った。これにより、文献[7)8)]に示されたシミュレーション結果を検証した。実現した H^3エンジンは同一の問題を SGA により解くソフトウェアに比べて約 750 倍の速度を得られることを確認した。また実装時のハードウェア量を測定し、GA の規模を拡張した場合のハードウェア量についての考察を行った。

2章では H^3エンジンの概要について述べる。3章で今回製作した H^3エンジンの設計と仕様について説明する。4章で H^3エンジンの実装効率と実行時間について評価する。5章で考察を行う。6章で結論を述べる。

2. H^3エンジン

本章ではまず H^3エンジンの概要について述べる。次に、H^3エンジンで採用された steady state GA について述べ、2 分探索と線形探索を併用したルーレット選択について述べる。

2.1 H^3エンジンの概要

従来の GA エンジンとして SPGA[3)]や HGA[5)]などがある。これらは基本的な GA である SGA をハードウェア化したものである。FPGA で実現した SPGA や HGA は、汎用プロセッサでのソフトウェア処理と比べ高速ではあるが、汎用プロセッサが FPGA の 10 倍のクロック周波数で動作すると仮定する場合、その速度向上比は約 7 倍から 18 倍にとどまっている。これは、これらの GA エンジンではパイプライン処理による高速化を図ってはいるものの、パイプラインがストールする時間が、その全実行時間の大半を占めているからである。パイプラインのストールは具体的には以下の 2 つの点で起こる。これを図 1 を用いて説明する。

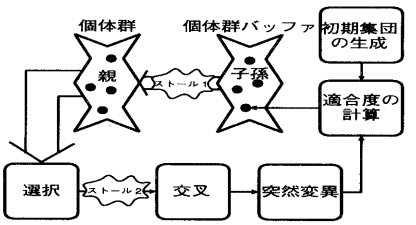

図 1　従来の GA エンジンのモデル

- **ストール 1**
  SGA とルーレット選択を用いた場合、選択を行う以前にその世代の全ての個体の適合度を求める必要がある。GA エンジンでは図 1のように、選択、交叉、突然変異、適合度の計算の流れでパイプライン処理が行われる。この 1 つの流れが N 回 (ただし N は 1 つの個体群中に含まれる個体の総数) 繰り返されて、1 世代の処理が終る。しかし、世代 t 中の全ての個体の適合度の計算が完了しなければ次世代 $t+1$ に入るべき個体の選択が開始できないため、パイプラインがストールする。
- **ストール 2**
  ルーレット選択を直接ハードウェアアルゴリズムに変換するとストール 2 が生じる。以下にこの理由を説明する。ルーレット選択は各個体の選択確率として、全個体の適合度の総和に対する各個体の適合度の占める割合を用いる手法である。この割合が扇型の角度に比例するようなルーレットを用意する。図 2(a) に個体数が 8 個の場合を示す。扇型の中に示された数が各個体の適合度を表し、外側に示された数は各個体の番号を表す。このルーレットを回し、ルーレットの矢印の位置に来た個体を次の世代の個体として選択する。以下にルーレット選択の手順を示す。

  手順 1 : 世代 t の個体群 $P(t)$ の中の N 個の個体の適合度 $f_i (1 \leq i \leq N)$ の総和 $S_N = \sum_{i=1}^{N} f_i$ を求める。

  手順 2 : $[0, 1]$ の乱数 $rand$ を発生させ、$r = rand \times S_N$ とする。

  手順 3 : $\sum_{i=1}^{k} f_i \geq r$ となるような最小の k を求

めて、k 番目の個体を世代 $t+1$ に生き残る個体の候補とする。

通常、**手順3** において k を求める操作は線形探索により行われる。この操作をハードウェアアルゴリズムに置き換えると図2(b) に示すような回路になり、選択のために加算と比較を繰り返し行わなければならない。この回路は逐次処理のループ構造になっているため、パイプラインがストールする。

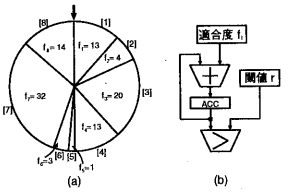

図2 ルーレット選択

ストール1およびストール2を除去するために、H^{3}エンジンでは2つの提案を行う。ストール1に対しては、各世代で1個体のみを置き換える steady state GA[9)10)] を導入する。ストール2に対しては、ルーレット選択を行う際、2分探索と線形探索を併用する。これにより、現実的なハードウェア量でルーレット選択回路のパイプライン化を実現することが可能になる。

2.2 steady state GA

steady state GA のモデルを図3に示す。選択を行う時は個体群より1個体が選ばれ、その後の交叉、突然変異、適合度の評価は1個体ずつ行われる。適合度が評価された新個体を個体群に投入し、同時に最古個体を個体群から破棄する。常に個体群が変化し続けるため、ルーレット選択で用いる適合度の総和を1イタレーションごとに更新する必要がある。この計算は現在の総和から最古個体の適合度を減算し、新個体の適合度を加算するだけなので短時間で計算できる。以上より、ストール1は除去できる。

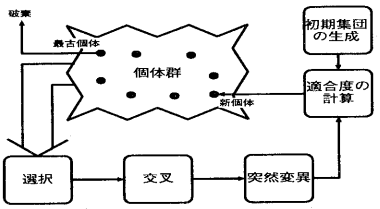

図3 steady state GA のモデル

2.3 2分探索と線形探索を併用したルーレット選択

2分探索と線形探索を併用したルーレット選択の手順を以下に示す。個体群中の j 番目までの個体の適合度の和を S_j とする。ただし、$S_0 = 0$ とする。探索範囲が $N/2^D$ (ただし、$0 \leq D \leq \log_2 N$) の大きさになるまで2分探索を用いる。ここで、D は2分探索を用いる回数、つまり分割回数である。ただし、N は2のベキ乗とする。D の最大値は $D = \log_2 N$ となる。探索範囲が $N/2^D$ のサイズになったあとは線形探索を行う。

H^{3}エンジンのルーレット選択の手順

1. 個体群 P の中の N 個の個体の適合度 $f_i (1 \leq i \leq N)$ の総和 $S_N = \sum_{i=1}^{N} f_i$ を求める。更に個体群の半分 $N/2$ までの和 $S_{N/2} = \sum_{i=1}^{N/2} f_i$ も求める。同様に $N/4$ までの和 $S_{N/4} = \sum_{i=1}^{N/4} f_i$、$3N/4$ までの和 $S_{3N/4} = \sum_{i=1}^{3N/4} f_i$ を求める。細分化された個体群の数が 2^D 個になるまで S_j を求める。

2. [0,1] の乱数 $rand$ を発生させ、$r = rand \times S_N$ とする。

3. 2分探索を用いて、探索する範囲の先頭の個体番号 x とそこまでの適合度の和 y を求める。

- **a.** $j = N/2, x = 1, y = S_0 = 0$ とする。現在探索している深さ $level$ を $level = 1$ とする。
- **b.** $r < S_j$ なら x と y は変更せずに、$j = j - N/2^{level+1}$ とする。$r \geq S_j$ なら $x = x + N/2^{level}$, $y = S_j$, $j = j + N/2^{level+1}$ とする。
- **c.** $level = level + 1$ とし、$level > D$ なら終了する。そうでない場合は、**3.** へ戻る。

$D = \log_2 N$ の場合は、x 番目の個体を選択し、終了する。そうでない場合は、**4.** へ進む。

4. $y + \sum_{i=x}^{k} f_i \geq r$ となるような最小の k を求めて、k 番目の個体を選択し、終了する。

1. と **2.** は、2分探索を用いたルーレット選択を行うための準備を行っている。**3.** は、探索する範囲を限定するため、2分探索を用いている。**4.** は、従来のルーレット選択と同様に線形探索を行うことにより、個体を選択している。**3.** について図4を用いて説明する。斜線部分は各探索ステップでの探索範囲を示している。まず Step1 では、r と $S_{N/2}$ の比較を行う。この比較により r が前半か後半のどちらに位置しているかを調べ、位置している方に探索範囲を限定する。同時に探索範囲の先頭の個体番号 x とそこまでの適合度の和 y を求める。次に Step2 では、Step1 で定められた探索範囲を更に半分に分割して r がどの範囲に位置しているかを調べ、再び探索範囲を限定する。同時に x と y を求める。以下同様にして分割回数 D になるまで繰り返す。

H^{3}エンジンでは、分割回数を $D = \log_2 \frac{N}{4}$ としている。すなわち、探索範囲が残り4個体になったところで、手順4による線形探索で個体を選択している。線形探索は組合せ論理回路により実現する。2分探索と線形探索をどのような比率で組み合わせるのが良いのかについて、5.1節で議論する。

3. H^{3}エンジンの設計と実現

本章では製作した H^{3}エンジンの概要、設計環境、実

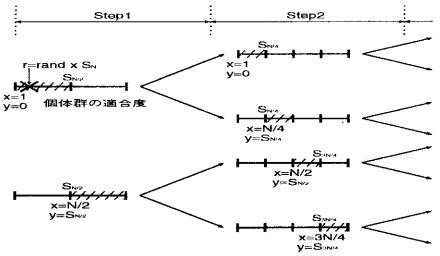

図4　2分探索の手順

現環境およびアーキテクチャについて述べる。

3.1 概　　要

今回実現した H^3 エンジンの概要について述べる。

規模

遺伝子長 8 ビット、適合度長 16 ビット、個体数 8 のモデルを設計した。

アプリケーション

関数最適化問題を扱う。適合度関数は $f(x) = |-(x-a)^2 + b|$、遺伝子表現にグレイコードを用いる。

遺伝的オペレータ

選択手法としてルーレット選択を用いる。また、エリート保存戦略を採用する。エリート保存戦略は、その時点までに現れた最も適合度の高い個体 (エリート) を常に個体群中に残す手法である。交叉には 1 点交叉を用いている。

遺伝的操作に用いる確率

交叉確率は 40%、突然変異確率は 10%とした。

入出力

GA で用いる乱数のシードを H^3 エンジンに入力する。回路が動作を始めてから最適解が生成されるまでに費したクロック数、つまり GA の処理終了までにかかったクロック数を出力する。

3.2 実 現 環 境

H^3 エンジンは、パーソナルコンピュータ (PC) の PCI バスに接続した OPERL ボード[11]上の FPGA に実装した。PC は Pentium プロセッサを搭載した IBM PC/AT 互換機である。FPGA は ATT2C40(40K ゲート相当)[12]を使用した。システム構成を表 1に示す。

FPGA ボードのクロックは PCI バスから供給されている 33MHz である。入出力のデータは PCI バスを通して PC とやりとりされる。入出力のバス幅は 32 ビットである。

表1　システム構成

PC	Intel Pentium 133MHz
FPGA ボード	OPERL ボード
FPGA	Lucent Technologies ATT2C40

3.3 設 計 環 境

設計は VHDL を用いて行った。論理合成ツールに Synopsys 社の VHDL Compiler と FPGA Compiler を、配置配線ツールには Lucent Technologies 社の ORCA Foundry を使用した。また、設計時のシミュレーションは Synopsys 社の VHDL System Simulator を用いて行った。

3.4 H^3 エンジンのアーキテクチャ

H^3 エンジンのアーキテクチャを図 5に示す。なお、実際は各ユニット内で遅延が問題となる箇所に遅延解消のためにパイプラインレジスタを挿入してあるが、簡単化のため以下では触れないこととする。Random Number Generator(RNG) は H^3 エンジンの中で用いる乱数を生成する。Fitness(F) Unit は、適合度を求める。Summation(S) Unit、Binary Search(B) Unit および Linear Search(L) Unit からなる Selection Unit で、ルーレット選択を行う。Crossover/Mutation(CM) Unit は、交叉と突然変異を行う。以下、各ユニットの動作について説明する。

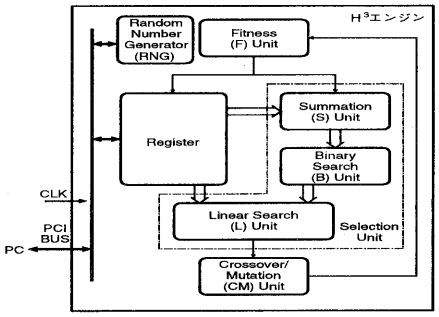

図5　H^3 エンジンのアーキテクチャ

3.4.1 RNG(Random Number Generator)

RNG は H^3 エンジンの中で用いる乱数を生成する。乱数生成には系列長 93 ビットの M 系列乱数[13]を用いる。M 系列乱数はシフトレジスタと、排他的論理和ゲートにより構成される。

初期化時は PCI バスからシードとなる数列をうけとる。初期化時以外は 1 クロックにつき 45 ビットの新しい乱数を生成し出力する。

3.4.2 F Unit(Fitness Unit)

個体の適合度を計算する。入力として初期化時は RNG から、初期化時以外は CM Unit から遺伝子を受け取る。計算した適合度は、F Unit の出力として Register と Selection Unit に送られる。

3.4.3 Selection Unit

ルーレット選択とエリート保存選択を併用し選択を行う。現実的なハードウェア量でパイプラインストールが生じないようにするため、2 分探索と線形探索を組み合わせて実現する。S Unit と B Unit で 2 分探索を用いて探索範囲を限定し、L Unit で線形探索を行う。図 6に S Unit の、図 7に B Unit および L Unit の回路の構成を示す。

S Unit(Summation Unit)

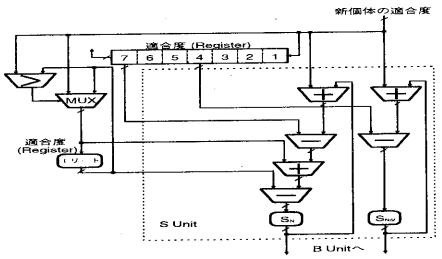

図6　S Unit と Register(適合度保存部) のアーキテクチャ

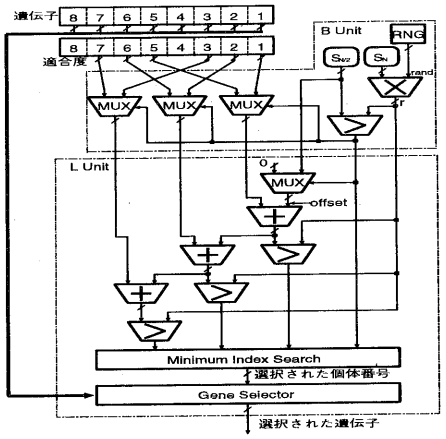

図7　B Unit と L Unit のアーキテクチャ

ルーレット選択を行う際2分探索を用いるため、S Unit は全体の適合度の合計と半分までの個体の適合度の合計を計算する。

入力は、新しい個体、新しいエリートの個体、一番古い個体、古いエリートの個体および4番目の個体の各適合度であり、B Unit に S_N と $S_{N/2}$ を出力する。

B Unit(Binary Search Unit)

B Unit は2分探索を行う。RNG から乱数 $rand$ を受けとり、$r = rand \times S_N$ を求める。求められた r と S_j の比較をし、比較の結果つまり2分探索により限定した探索範囲を示す情報をコントロール信号として L Unit に出力する。

L Unit(Linear Search Unit)

L Unit は B Unit により限定された探索範囲について線形探索を行う。なお、線形探索は状態回路を用いるのではなく、組み合わせ回路により実現する。B Unit より送られてきたコントロール信号により、探索する4個体をマルチプレクサで選択する。マルチプレクサ以降は、$y + \sum_{i=x}^{x+3} f_i \geq r$ の判定を各 i について行い、Minimum Index Search で上記条件を満足する最小の i を求める。Gene Selector により i 番目の遺伝子を選択する。

入力は B Unit からのコントロール信号と選択対象にある全ての個体の遺伝子と適合度および r 、出力は選択された個体の遺伝子である。

3.4.4 CM Unit

Crossover Unit

連続して入ってくる2つの遺伝子を一組と見て、その組に対し一点交叉を行う。図8に回路の構成を示す。交叉対象は、入力された遺伝子と1クロック前に入力されバッファに格納されていた遺伝子の組である。RNG より入力される乱数が、閾値 (交叉確率) を超える場合に交叉を行う。交叉点も乱数により決定する。Crossover Control はこれらの判定を行い、各遺伝子座に対し信号を出力する。これに従い交叉が行われる。交叉により生成される2個体のうち、1個体はそのまま出力され、もう1個体はバッファにいったん格納され1クロック後に出力される。この切替えは出力部のマルチプレクサが行う。

入力として乱数と遺伝子を受け取り、交叉後の遺伝子を Mutation Unit に出力する。

Mutation Unit

突然変異を行う。図9に回路の構成を示す。Mutation Control が各遺伝子座に対し、入力された乱数を用い突然変異確率に従って突然変異を行うかどうかの判定を行い、信号を出力する。これ従い突然変異が行われる。

入力として乱数と遺伝子を受け取り、突然変異後の遺伝子を F Unit へ出力する。

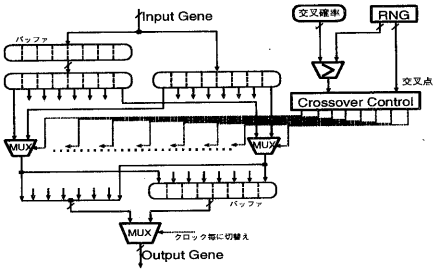

図8　Crossover Unit

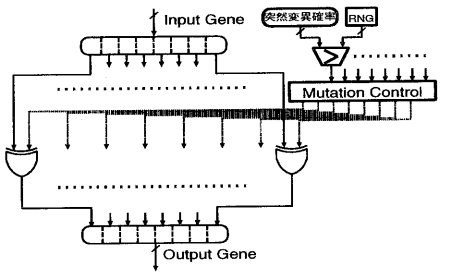

図9　Mutation Unit

3.4.5 Register

個体群の遺伝子と適合度を保持する。図6に適合度に関する部分を示す。通常の個体7個体を保持するレジスタ及びエリート保存用のレジスタ☆とエリート保存を実現するための周辺回路によって構成される。

☆ 図7における8番目の個体に相当する

4. H^3エンジンの評価

4.1 ハードウェア量

H^3エンジンをFPGA(ATT2C40)に実装した際のデータを以下に示す。

- FPGAリソースの使用率

  FPGA内の各リソースの使用率を表2に示す。なお、このデータは配置配線を実行した後のORCA Foundryの出力である。

表2 FPGAリソースの使用率(ATT2C40使用時)

リソース名	使用個数/全個数	使用率
IO	41/172	23%
LOGIC	752/900	83%
SPECIAL	1/7205	0%
PFU	752/900	83%
GSR	1/1	100%

表の各リソースは次のような意味を持っている。

IO　I/Oの数

LOGIC　ロジックセルの合計数

SPECIAL　トライステートとその他の特殊なライブラリの使用数

PFU　プログラマブル・ファンクション・ユニットの使用数

GSR　グローバルセットリセット端子の使用状況

なお、ATT2C40ではロジックセルはPFUしかないため、ロジックセルの使用数とPFUの使用数は同じ値である。48ゲート/PFUなので、PFU使用数より、製作したH^3エンジンの規模は約35Kゲート相当である。

- 各ユニットの占める割合

  H^3エンジン全体に対し、各ユニットの占める割合を表3に示す。この値は論理合成後のFPGA Compilerの出力である。表中の面積はATT2Cシリーズのテクノロジ・ライブラリの基本単位であるPFUの個数である。PFU個数が表2のPFUの合計値と異なるが、これはPFUを構成するF/FやLUTを全て使用すると配線が困難になり、また高速な動作も困難となるため、マッピング時にリソース使用効率をある程度低下させるためである。

表3 各ユニットの占める割合

ユニット名	面積	全体における割合
RNG	27.9	5%
F Unit	73.5	14%
Register	62.1	11%
S Unit	37.5	7%
B Unit	133.9	25%
L Unit	157.3	29%
Cross over	17.6	3%
Mutation	11.6	2%
合計	537.2	100%

4.2 実行時間

測定結果を表4に示す。最適解が得られるまでを1回とし、10万回実行を繰り返した時の総クロック数と実行時間である。実行時間はクロック数にH^3エンジンのクロックサイクル時間30nsを乗じて得た値である。

表4 測定結果(10万回の合計)

クロック数	実行時間(sec)
6601277	0.198

ハードウェアとソフトウェアで実行した場合の実行時間を比較する。測定は3種類:H^3エンジン、steady state GAを用いたソフトウェア、SGAを用いたソフトウェアについて行った。ソフトウェアはDEC AlphaStation(クロック周波数333MHz)上で実行した。なお、ソフトウェアの実行時間の測定にはATOM[14]を用いた。

測定結果を表5に示す。表5より、H^3エンジンはsteady state GAを用いたソフトウェアに比べ約480倍、SGAを用いたソフトウェアに比べ約750倍高速であることがわかった。

5. H^3エンジンの規模の拡張に関する考察

実用的な問題を解くためには、今回製作したプロトタイプよりかなり規模を大きくする必要がある。H^3エンジンの規模(遺伝子長, 適合度長, 個体数)を拡張する場合問題となる箇所は図5のSelection UnitとRegisterである。その他の箇所は大きく構成を変更する必要はない。Selection UnitとRegisterは、個体数が増えると回路規模がそれにともない大きく増加する。そのためSelection Unitは、2分探索と線形探索を組み合わせる比率を変更するなどして回路規模を小さくする必要がある。Registerについてはより大きな容量が必要となりFPGAに入らなくなると、外部にメモリを付加するなどの変更を加える必要がある。

本章ではまずH^3エンジンのSelection Unitのハードウェア量についての考察を行う。次に規模を大きくした場合のRegisterの構成法について考える。

5.1 Selection Unitのハードウェア量

4章において今回実現したH^3エンジンのハードウェア量を示した。本節ではこのデータをもとに、Selection Unitにおいてどこまで B Unitによる2分探索で探索範囲を限定し、どこからL Unitによる線形探索を行うのが適当であるかについて考える。これにより、H^3エンジンが現実的なハードウェア量でパイプラインストールをなくすために、選択時に2分探索を用いることの妥当性について検証する。また、ハードウェア量を最小にする2分探索の探索レベル数(分割回数)を求める手法を示す。

はじめに個体数がN(Nは2のベキ乗である任意の数)である場合について考える。次に個体数を今回実現したモデルと同じく8とした場合について考える。

表5 ハードウェアとソフトウェアの実行時間の比較(10万回の合計)

実行時間 (sec)			実行時間の比	
H^3エンジン	steady state GA(soft.)	SGA(soft.)	steady state GA(soft.)/H^3エンジン	SGA(soft.)/H^3エンジン
0.198	95.25	148.60	481	751

なお、簡単化のためにハードウェア量の少ない箇所のハードウェア量および、それぞれの探索の範囲を変更するときに構成に変更のない箇所のハードウェア量を無視し、加算器、減算器と比較器のハードウェア量のみについて考える。また、加算器、減算器や比較器のビット幅は全て32ビットとする。

5.1.1 個体数 N の場合

2.3節において説明したように個体群中の j 番目までの個体の適合度の和を S_j とし、細分化された個体群の数が 2^D 個になるまで S_j を求める必要がある。よって、S Unit と B Unit で用いる加算器と減算器の数はそれぞれ求めなければならない S_j の個数である 2^D にエリート保存のため1を加えた値である 2^D+1 である。比較器の数は探索範囲を D 回分割して r を含む範囲を求めるため、D となる。

またL Unitでは、2分探索により $N/2^D$ 個にまで限定された探索範囲から線形探索により個体を求めるため、必要となる加算器の数は $N/2^D-1$ である。比較器の数も同様に $N/2^D-1$ となる。

以上より、個体数を N 、分割回数を D とした時、Selection Unit 全体での加算器の数は $2^D+N/2^D$ となる。また、減算器の数は 2^D+1 、比較器の数は $D+N/2^D-1$ となる。これらの加算器、減算器、比較器のハードウェア量は今回実装した際のデータでは順に約2.6, 約2.7, 約3.8である。この値の単位はATT2Cシリーズのテクノロジ・ライブラリの基本単位であるPFUの個数である。これらの値を用いると、Selection Unit のハードウェア量 $H(D,N)$ は、以下の式で表される。

$$H(D,N) = 5.3 \times 2^D + 6.4N/2^D + 3.8D - 1.1$$

この式は下に凸な関数であり、最小値をもつ。上の式を最小にする D の値は上の式を D で微分することで得られ、以下の式で表される。

$$D \approx 1.4427 \ln(0.0136103\ (-38+\sqrt{1444+6518.79N}))$$

この式を用いれば、ハードウェア量 $H(D,N)$ が最小になる分割回数 D を求めることができる。N と D を変更した場合のハードウェア量 $H(D,N)$ についてのグラフを図10に示す。例えば $N=128$ の場合は $D=4$ とした時にハードウェア量が最小になる事がわかる。

5.1.2 個体数8の場合

次に、今回実現した個体数8の場合について前節で示したハードウェア規模の予測式の妥当性を検証する。今回実現した H^3 エンジンでは、2分探索を用いて探索範囲が残り4個体になったところで線形探索により個体を選択している。つまり、分割回数を $D=\log_2\frac{N}{4}=1$ と

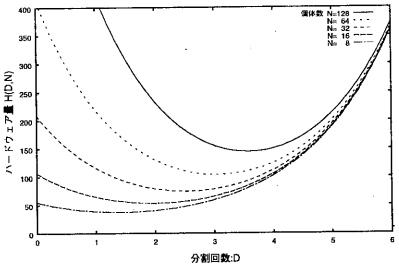

図10 個体数 N と分割回数 D に対する Selection Unit のハードウェア量

している。この場合S Unit と B Unit で用いている加算器の数は3、減算器の数は3、比較器の数は1である。また、L Unit で用いている加算器の数は3、比較器の数は3である。

次に、B Unit で探索範囲を2個体まで限定する場合、つまり $D=\log_2\frac{N}{2}=2$ の場合を考える。S Unit と B Unit で用いる加算器の数は5、減算器の数は5、比較器の数は2である。また、L Unit で用いる加算器の数は1、比較器の数は1である。

さらに、2分探索を用いない場合について、つまり $D=0$ の場合について考える。S Unit と B Unit で用いる加算器の数は2、減算器の数は2、比較器の数は0である。また、L Unit で用いる加算器の数は7、比較器の数は7である。

各々の場合について、加算器、減算器、比較器のハードウェア量を先程と同様に順に約2.6, 約2.7, 約3.8であるとし以上より算出した結果を表6に示す。$D=\log_2\frac{N}{4}=1$ の時、ハードウェア量は最小となる。

表6 分割回数 D を変化させた場合の Selection Unit のハードウェア量

D	加算器	減算器	比較器	合計
2	15.6	13.5	11.4	40.5
1	15.6	8.1	15.2	38.9
0	23.4	5.4	26.6	55.4

5.2 H^3 エンジンの規模を拡張した場合の Register の構成法に関する考察

適合度保存用レジスタと遺伝子保存用レジスタについて別々に考える。また、エリート保存戦略用のレジスタに関してはFPGA内に持つものとする。

適合度保存用レジスタについて

適合度が必要となるのは以下の場合である。

- S Unit で適合度の和(適合度の総和、個体群の半分までの和等)を求めるために、求める和の数だけ読み込む。

- B Unit で二分探索を行い探索範囲を限定したのち、L Unit で探索する $N/2^D$ 個の個体の適合度を読み込む。
- 新しく計算した適合度を書き込む。

これらを同時にアクセスするため、外部にメモリを付加して実現しようとすると必要なバス幅を得ることが現実的には不可能であると思われる。よって、適合度保存用レジスタはFPGA内のレジスタで構成する必要がある。なお、バス幅による制限はFPGAチップのI/Oピンの数による制限とほぼ等しいものである。

例えば、個体群数が128で、適合度長を32ビットとした時必要なビット数は4096ビットである。このサイズならばレジスタをFPGA内にSRAMによって構成してもF/Fによって構成しても現在入手可能な規模のFPGA内に十分おさまる。

遺伝子保存用レジスタについて

遺伝子が必要となるのは以下の場合である。

- 選択された遺伝子を読み込む。
- 交叉・突然変異が終った後、新しい遺伝子を書き込む。

遺伝子に関して必要なアクセスは上に示した毎クロック1度づつの読み書きである。遺伝子長は一般に適合度長より長く、FPGA内に保存することが困難であると思われる。そのため、外部にメモリを付加する構成にする必要がある。こちらもバス幅が問題となることが考えられるが、適合度の場合よりもその問題は小さい。必要なバス幅が得られる場合、H^3エンジンの動作するクロックに対して、1クロック中に2回アクセスできる速度の高速なメモリを用いることで実現が可能である。

6. まとめ

H^3エンジンをFPGA上に実装した。これにより、H^3エンジンはH^3を用いたソフトウェアに比べ約480倍、SGAを用いたソフトウェアに比べ約750倍速度向上することがわかった。また、H^3エンジンのハードウェア量に関する考察を行った。

今回設計したモデルは遺伝子長8ビット 個体数8である。実用的な問題をこの仕様のGAで解くのは難しい。実装した結果から考えると、適合度関数にかかるハードウェア資源が少なければ、今回用いたサイズのFPGAでも遺伝子長や個体群サイズを拡張することが可能であると思われる。

FPGAの集積度は年々高くなっているため、近い将来に大規模なGAをH^3エンジンで実現できるようになる可能性は高い。また、H^3エンジンのシステムには、可変構造が必要なく、専用LSIとして実現できる箇所が存在する。それらを、FPGAより高速な専用LSIとし、再構成可能でなければならない機能ユニットをFPGA上に複数用意し並列化すれば、FPGAと専用LSIそれぞれの長所を生かしたシステムが可能になる。

今後はより広範囲なアプリケーションへの適用性を探求すると共に、大規模なH^3エンジンのシステムの構築について考える予定である。

謝辞

本研究の一部は、名古屋産業科学研究所との共同研究「次世代CPUのアーキテクチャ」の支援により行いました。ここに感謝の意を表します。

参考文献

1) J. H. Holland : *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.
2) D. E. Goldberg : *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, 1989.
3) P. Graham and B. Nelson : "A Hardware Genetic Algorithm for the Traveling Salesman Problem on Splash 2," *Field Programmable Logic and Applications*, pp.352-361, August 1995.
4) P. Graham and B. Nelson : "Genetic Algorithms In Software and In Hardware - A Performance Analysis Of Workstation and Custom Computing Machine Implementations," *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pp.216-225, April 1996.
5) S. D. Scott, A. Samal, and S. Seth : "HGA: A Hardware-Based Genetic Algorithm," *Proceedings of the 1995 ACM/SIGDA 3rd Int'l Symp. on FPGA*, pp.53-59, February 1995.
6) 大島龍之介, 松本尚, 平木敬 : "再構成可能な遺伝的算法エンジンの研究," *The Third Japanese FPGA/PLD Design Conf. & Exhibit*, pp.541-548, July 1995.
7) 浅田英昭, 杉浦弘幸, 川合隆光, 安藤秀樹, 島田俊夫 : "遺伝的アルゴリズムの専用ハードウェア化," 第7回インテリジェント・システム・シンポジウム講演論文集, pp.359-364, 1997年11月.
8) H. Asada, H. Sugiura, T Kawai, H. Ando, and T. Shimada : "H^3: High-speed Hardware for Human-like genetic algorithm," *Proceedings of Third International Symposium on Artificial Life and Robotics*, pp.190-195, January 1998.
9) D. Whitley and J. Kauth : "Genitor: A Different Genetic Algorithm," *Proceedings of the 4th Rocky Mountain Conference on Artificial Intelligence*, June 1988.
10) K. A. De Jong and J. Sarma : "Generation Gaps Revisited," *Foundations of Genetic Algorithms-2*, Morgan Kaufmann Publishing, pp.19-28, 1992.
11) 市川周一, 島田俊夫 : "PCIバスに付加する再構成可能ボードの試作評価," 信学技報 *VLD96-85*, pp.159-166, 1996年12月.
12) *AT&T* フィールド・プログラマブル・ゲートアレイ データブック, AT&T Microelectronics , 1995.
13) D. E. Knuth, 渋谷政昭訳 : 準数値算法/乱数, サイエンス社, 1981.
14) A. Srivastava, A. Eustace : "ATOM: A System for Building Customized Program Analysis Tools," *WRL Research Report 94/2*, Digital Equipment Corporation, March 1994.