GA における再利用の評価

鈴木郁真[†] 池内康樹[†] 津邑公暁[†]
中島康彦[††,†††] 中島浩[†]

遺伝的アルゴリズムにおいて最も処理時間を要する適合度計算に対し，再利用を適用することで高速化する手法を提案し，再利用の有効性を示す。適合度計算の入力となる遺伝子が，前世代で処理された遺伝子と多くの共通部分を持つことから，適合度関数を分割することで再利用の効果を引きだす手法について述べる。GENEsYs を用いて評価した結果，平均 27%，最大 83%のサイクル数を削減できた。更に，関数分割などの改良を施すことにより，平均 38%，最大 86%までこれが向上した。特に適合度計算に要する時間が長い適合度関数について，再利用の効果がより大きくなることが分かった。

An Evaluation of Reuse with Genetic Algorithms

Ikuma Suzuki,[†] Yasuki Ikeuchi,[†] Tomoaki Tsumura,[†]
Yasuhiko Nakashima[††,†††] and Hiroshi Nakashima[†]

This paper describes a speedup technique with computational reuse for the fitness calculation of GA programs. A genotype has many genes in common with its parental genotypes. Therefore, partial results of fitness calculation are reusable. Through the result of an evaluation with GENEsYs: a well-known GA software, we show that the maximum ratio of the cycle reduction reaches 83%, while accomplishing average reduction of 27%. Futhermore, dividing fitness procedures raises the maximum ratio to 86% and average ratio to 38%.

1. はじめに

遺伝的アルゴリズム (Genetic Algorithm: 以下 **GA**) は，解探索に古くから用いられている有効な手法のひとつである。しかし，GA はその有効性ゆえに解決困難な問題に適用されることがほとんどであり，現在の計算機性能をもってしても，膨大な処理時間を要する場合が多い。

このため，現実的な時間で有効な解を得るためには，並列化を用いた GA の高速化が不可欠とされてきており，理論と実装の両面において多くの並列 GA の研究が行われている[1)]。一般に並列 GA では，プロセッサ間で個体集団の移送を行う必要があり，これが高速化する際のボトルネックとなるため，個体集団をコンパクトに表現することで移送コストを軽減する研究なども行われている[2),3)]。

他の GA 高速化のアプローチとして，専用ハードウェアを用いたものがある。やはり多くのハードウェアベース GA が研究されているが[4)~6)]，顕著な高速化を得るためにはハードウェアは複雑かつコスト高となる。また，専用ハードウェアを用いたアプローチは遺伝オペレータが固定される場合が多く，汎用性に欠け，アルゴリズムの変更等に適応することが困難である。

そこで本稿では，**再利用**[7)] を用いた，並列化とは全く着眼点の異なる GA の高速化手法を提案する。再利用とは，関数などの命令区間の実行時にその入出力の組を記憶し，再び同じ入力によってその命令区間が実行されようとした場合，記憶してある出力を書き戻すことで命令区間の実行自体を省略する高速化手法である。

GA の処理において，前世代と現世代の個体の遺伝子表現に共通部分が多いことから，個体の適合度計算などの部分に再利用が効果的に適用可能であることを示す。ただし遺伝子表現は過去に出現した遺伝子表現と完全には一致しないことから，再利用を効果的に適用するためのプログラミング手法について述べ，評価結果によりその効果を示す。再利用による GA 高速化は，従来の並列 GA と独立に適用可能な手法である。よってこれらを併用することで，更なる高速化も可能となると考えられる。

以下 2 章では，適用手法である再利用の概要について述べ，3 章で，GA の一般的な処理プロセスについ

† 豊橋技術科学大学
Toyohashi University of Technology
†† 京都大学
Kyoto University
††† 科学技術振興機構さきがけ研究 21
PRESTO, JST

て概説しながら，再利用の適用可能性について探る。4 章では具体的な再利用の適用手法について述べ，5 章で汎用 GA ソフトウェアである GENEsYs を用いた評価結果を示す。最後に 6 章で各 GA パラメータと再利用の効果の関係について考察する。

2. 再利用実行モデル

本章では，GA の高速化手法のベースとなる再利用実行モデルについて概説する。

2.1 概　　要

再利用は，プログラムを構成する命令区間を多入力・多出力の複合命令としてとらえ，過去に行った複合命令の実行結果を記録しておくことで，同一入力による当該複合命令の実行自体を省略する高速化手法である。従来多くの研究が行われてきた値予測および投機的実行は，数多くの命令の投機あるいは実行結果の破棄が必要であるのに対し，再利用は実行する必要のある命令列そのものを削減できるという点で，従来とは発想の異なる高速化技術である。

従来の再利用研究では，単命令を対象とする単純なもの[8] や，コンパイル時に再利用のための情報を埋めこむもの[9),10)] などが提案されている。これに対し我々は，再利用技術に基づいた汎用プロセッサを提案しており[7]，これまでメディア処理を始めとするアプリケーションにおいて有効な結果を示している[11),12)]。我々の実行モデルでは，再利用対象とする命令区間として，多くの命令を含みかつ始点と終点を用意に特定できる，関数およびループを仮定する。これにより，再コンパイルや，静的解析に基づく付加情報埋め込み (バイナリアノテーション) を必要とせず，既存バイナリに変更を加えることなく高速化が可能となる。

2.2 命令区間の検出

まず再利用を実現するためには，再利用可能な命令区間を検出する必要がある。我々が提案する機構では，命令区間を関数およびループイタレーションとしている。関数に含まれる命令は，call/jump 命令の分岐先から ret 命令までであることから，call/jump 命令の検出から ret 命令の検出までを，再利用対象の命令区間とする。一方ループの場合は，一度後方分岐命令が検出され，その後方分岐が成立した後に再び同じ後方分岐が検出されたとき，その後方分岐命令と後方分岐先までをループイタレーションとして検出する。これにより，動的に命令区間を検出することが可能となる。

次に，例えばある関数 f を再利用するためには，f の実行時にその入出力を記憶する必要がある。関数 f を呼びだす関数を f_{p} とすると，f の入力となりうるのは，大域変数および f_{p} の局所変数である。そこで，f の入出力をメモリマップ上で識別するためには，大域変数と f の局所変数の境界，および f の局所変数と f_{p} の局所変数の境界を確定しなくてはならない。

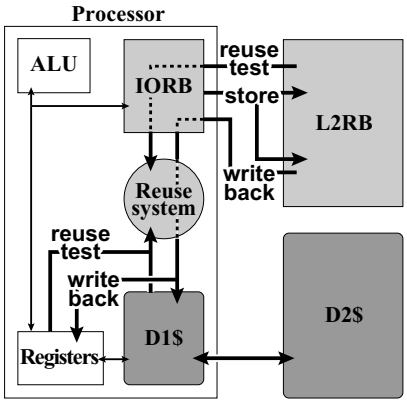

図 1　再利用機構

我々の再利用機構では，SPARC ABI[13] に従って記述されたプログラムに対し，この規定に基づく条件を仮定することで，これを実現している。具体的には，SPARC ABI の

- %sp 以上の領域のうち，%sp+0〜63 はレジスタ退避領域，%sp+68〜91 は引数退避領域。
- 構造体を返す場合の暗黙的引数は%sp+64〜67
- 明示的引数はレジスタ%o0〜5，および%sp+92 以上。

という条件に加え，一般的に OS が実行時のデータサイズおよびスタックサイズの上限 (LIMIT) を決めることから，以下を仮定することで，識別が可能となる。

- 大域変数は LIMIT 未満。
- %sp は必ず LIMIT より大きく，LIMIT〜%sp の領域は無効。

2.3 入出力記録表

命令区間にとって入力とは，レジスタおよび主記憶に対する参照である。命令区間は，レジスタおよび主記憶を参照することで入力を得，処理を行った結果をレジスタおよび主記憶に書き戻すことで出力を行う。このとき，入力値が等しい間は出力も同じであり，入力が異なった命令以降の出力は枝分れ的に派生していく。つまり参照順に入力を並べた場合，レジスタ番号や主記憶アドレスをノードとし，その格納値を枝とするような多分木に含まれる 1 パスとして，ある入力セットは表現される。

再利用を実現するためには，このような多分木で表現された，過去に実行された命令区間の入力セットを記憶するための**再利用表** (reuse buffer) が必要となる。さてこの再利用表は，命令区間実行時には単純に入力の一致検索が行われるだけでなく，命令区間による出力の書き込みや，自らが書き込んだ出力の再度読み出しなど，頻繁に入出力処理を行う必要がある。多分木構造は，多種の入力パターンを表現するには適しているものの，このような頻繁な読み書きが行われる場合はその低速さが問題となってしまう。そこで，単

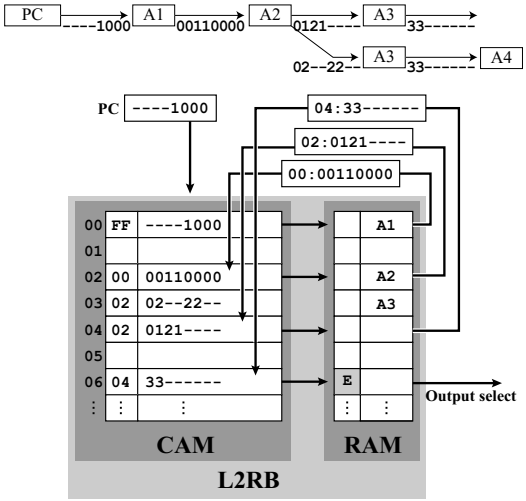

図 2 L2RB の構造と動作

一の入力パターンを記録できる小規模かつ高速な再利用表 (以下，**IORB**) と，過去の出現した複数の入力パターンを格納する再利用表 (以下，**L2RB**) を用意する。命令区間実行中は IORB を用い，命令区間実行終了時に IORB から L2RB に入力セットを保存することにより，アクセス効率のよい再利用表を実現する。

2.4 再利用機構

我々の実行モデルが想定する再利用機構の概要を**図 1** に示す。プロセッサは，命令区間実行中に L2RB を参照し (reuse test)，入力の一致比較を行うと同時に，IORB に入出力セットを登録しながら命令区間を実行する。入力セットが完全に一致するエントリが見つかった場合，当該エントリに対応する出力を L2RB からキャッシュおよびレジスタに書き戻す (write back)。これにより命令区間の実行が省略される。入力が一致しなかった場合，命令区間実行終了時に IORB の内容を L2RB に格納する (store)。

ここで再利用表，特に L2RB における入力一致比較のための連想検索コストが，再利用に要するオーバヘッドの大部分を占める。よって L2RB は，連想検索を高速に行える必要がある。本機構ではこの L2RB を中容量の汎用 CAM (Content-Addressable Memory) を用いて構成することを想定しており，これにより再利用オーバヘッドを比較的小さく抑えることができる。

L2RB の構成と動作を**図 2** に示す。前述のように，入力パターンは図 2 上部に示したような多分木で表現できる。これを，L2RB 内の CAM 部に枝，RAM 部にノードを対応させることで，折畳んで格納する。CAM 部の各エントリには親エントリを表すインデクスを用意し，入力値と組み合わせたものをキーとして検索を進めていく。アドレスを格納する RAM 部のエントリには終端フラグを設けることで，入力一致比較の終端を示すとともに，可変長の入力セットを格納可能とする。

図 2 の例では，まず PC の値が参照され，その値 `----1000` およびツリーのルートを表すインデクス `FF` のセットをキーとして L2RB が検索される。例では L2RB のインデクス `00` の行がマッチする。そこで RAM 部の行 `00` が参照され，次に参照すべきアドレス `A1` を得る。次にアドレス `A1` の値が読みだされ，その値 `00110000` およびインデクス値 `00` のセットが次の検索キーとなる。このようにして L2RB の検索を繰り返し，アドレス格納表において終端フラグ `E` を検出すると，全ての入力が一致したことが保証される。この場合，保存されている対応する出力値を読みだし，レジスタおよび主記憶に書き戻すことで命令区間の実行を省略する。

3. 遺伝的アルゴリズム

3.1 概　　要

GA は，生物の進化プロセスをもとにして構築された解探索手法である。定式化することが難しいと考えられる問題に対し，進化を計算機上でシミュレートすることで，有効に解の探索を行うことを目的として使用される。

具体的には，対象となる問題の解を**遺伝子表現** (genotype) にコーディングし，さまざまな遺伝子表現を持つ個体同士で生殖活動を行わせることで次世代の子孫を作りだしながら，それらの個体の適合度に応じて選択・淘汰を行っていくことで，集団全体の適合度を向上させ，これを繰り返すことで最適な解を探索する。

3.2 プ ロ セ ス

GA では一般的に，生殖，適合度計算，個体選択というプロセスで 1 世代の処理が構成され，これを繰り返し行うことにより解探索を行う。本節では，GA に再利用を適用する背景として，これらの処理プロセスを順に概説し，それぞれの再利用適用可能性について考察する。

生　　殖

生殖処理では，遺伝子表現に対して**交叉** (**crossover**) および**突然変異** (**mutation**) の処理を行い，次世代の遺伝子表現を生成する。

交叉は，まず一定の確率 (交叉率) に従い個体を選択し，選択された個体同士で一部の**遺伝子** (gene, 遺伝子表現の構成要素) を交換する操作である。交叉手法には，遺伝子表現内の一点でのみ交叉を行う一点交叉，多点で行う N 点交叉，全ての遺伝子を両親の同位置の遺伝子のどちらかからランダムに選択する一様交叉などがある。N 点交叉 ($N = 2$) の例を**図 3** に示す。

突然変異は一定の確率 (突然変異率) に従って，全個体のなかから遺伝子を選択し，その遺伝子に対して書

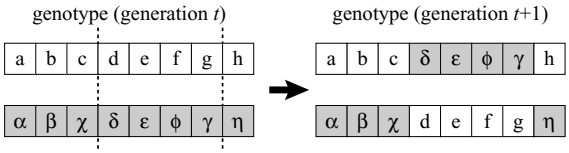

図 3 N 点交叉 ($N = 2$)

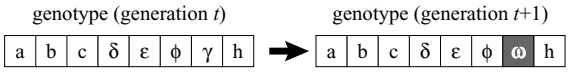

図 4 突然変異

き換えを行う操作である (**図 4**)。遺伝子が単純な 0/1 で表現されている場合は，当該遺伝子に対してビット反転が行われる。

さて，生殖における主な処理は，遺伝子表現の変更であり，プログラム中ではそのほとんどは配列のコピー処理で占められる。この計算量は軽微であり，再利用オーバヘッドなども考慮すると，再利用の効果は低いと考えられる。

適合度計算

次に，生殖で生成された各個体の適合度の計算を行う。一般には，遺伝子表現を入力とし適合度を計算・出力する関数を用意し，その関数を適用することで各個体の適合度を算出する。この際，遺伝子表現は関数の入力として扱いにくいため，一般に遺伝子表現に対して何らかの変換が行われたうえで，適合度が計算される。なお，生殖の対象とならず前世代から変化していない遺伝子表現に関しては，一般に適合度計算は省略される。

適合度計算は，GA において最も時間を要する処理であり，この部分を高速化することによって GA 全体の所要時間を大きく削減できる可能性がある。さて，適合度計算関数の入力となる遺伝子表現は，前世代に存在した遺伝子表現の交叉によって生成されたものであり，その一部は常に前世代の個体と共通している。つまり，適合度計算において構成遺伝子の部分集合に対する処理が含まれている場合は，その処理は再利用が可能であると考えられる。また，要する処理量から考えても，適合度計算は再利用に適した処理であると言える。

個 体 選 択

算出された適合度の高いものほど多産であり，低いものほど淘汰されやすくなるように個体を選択する処理である。適合度に比例した割合で個体を次世代に残す方法，適合度の高いものだけを残す方法，集団からランダムに個体集団の部分集合を選択し，その中で最も良い個体を選択することを繰り返すことで次世代の個体集団を生成する方法などがある。

個体の選択処理では，すでに計算された適合度に基づく比較やソートがその多くを占めるため，やはり生殖と同様処理量が軽微であり，再利用適用による効果は低いと考えられる。

分割前

```
double f( x, n )
 double x[];
 int    n;
{
  int    i;
  double sum = 0.0;
  for( i=0; i<n; i++ ){
    sum += x[i]*x[i];
  }
  return( sum );
}
```

分割後

```
double f( x, n ){
  double x[];
  int    n;
{
  int    i;
  double sum = 0.0;
  for( i=0; i<n; i+=6 ){
    sum += g(x[i],   x[i+1],
             x[i+2], x[i+3],
             x[i+4], x[i+5]);
  }
  return( sum );
}

double g( x1, x2, x3, x4, x5, x6 )
  double x1, x2, x3, x4, x5, x6;
{
  return( x1*x1 + x2*x2 + x3*x3
        + x4*x4 + x5*x5 + x6*x6 );
}
```

図 5 適合度関数の分割例

4. 再利用の適用

前章で述べたように，最も再利用の効果が見込めるのは適合度計算である。しかし，処理対象となる遺伝子表現は過去に出現した遺伝子表現と完全には一致していないため，再利用の高い効果を得るためにはプログラミングに工夫が必要である。本章では，適合度の算出プロセスにおいて，再利用率を向上させる方法について述べる。

4.1 遺伝子の格納アドレス

前述のように処理対象となる遺伝子表現は前世代からその一部を受け継いでいるため，入力セットのサブセットは過去に用いられた入力セットと一致し，再利用による効果が見込める。

しかし，一般に個体が異なる遺伝子表現はプログラム上で別配列に格納されており，実行時の主記憶値アドレスが異なる。すなわち，適合度計算の入力値が一致する場合でも，その値が格納されているアドレスが異なるため，再利用機構は同一入力であるという判定を行うことができない。

これを回避するには，1 つの遺伝子表現が格納できる配列を用意し，処理対象となる遺伝子表現を常に一度その配列にバッファリングした上で適合度計算を行うようにする。これにより，入力アドレスの違いが解消でき，再利用率を十分に引きだすことが可能となる。

4.2 入力遺伝子数

適合度計算の入力には，当然ながら遺伝子表現に含まれる全ての遺伝子が使われる。しかし遺伝子表現は，過去に出現した遺伝子表現とは部分的には共通しているものの完全に同じではない。つまり，遺伝子表現全体を処理対象とするような適合度計算関数は，入力値が過去と完全に一致することはなく，再利用の効果が望めない。

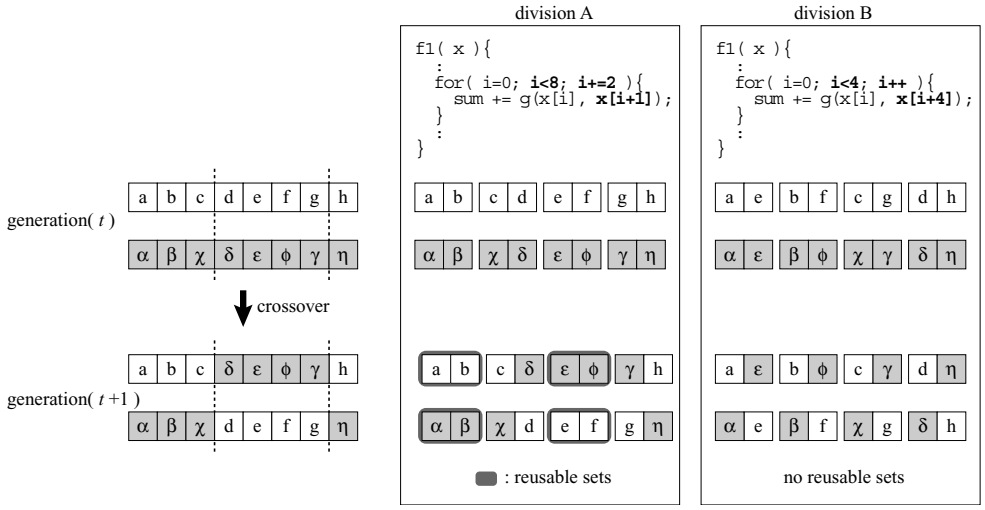

図 6 共通遺伝子の局所性と再利用可能性

ただし一般に適合度計算は，遺伝子表現の一部にのみ依存するような処理に分割できる可能性が高い。このような場合，遺伝子表現の一部を処理するサブ関数を定義し，それらの結果から適合度を計算するようにすることで，そのサブ関数に対する再利用の効果が望める。例えば De Jong のテスト関数[14] のひとつとして知られる sphere function は，各遺伝子の自乗の総和を計算する適合度関数である。この場合，**図 5** 左のように記述すると関数 `f()` に対する入力は過去と一致しないため再利用不可能であるが，図 5 右のように書きかえることで関数 `g()` について再利用可能となる。

ただし，サブ関数の入力遺伝子数をあまり多く仮定すると，入力が完全に一致する確率が低くなり，再利用率も低く抑えられてしまう。また，入力遺伝子数を少なく仮定すると，サブ関数自体の処理量が少なくなることで，入力が一致した場合に削減されるサイクル数が少なくなり，再利用の効果があまり上がらない可能性がある。このため，入力遺伝子数を適切に設定する必要がある。

4.3 前世代と共通する遺伝子の局所性

上述の関数分割を行う場合，再利用による効果は，サブ関数の入力とする遺伝子の選び方にも依存する。というのも，特に交叉方法が N 点交叉である場合，前世代の個体と共通している遺伝子集合には局所性があるためである。

図 6 に，単純な例を示す。この例では遺伝子表現長は 8 であり，2 つずつの入力セットに分割されて関数 `g()` により処理された結果を全て加算することで適合度を求める。さて，分割 A および 分割 B では，サブ関数 `g()` の定義は同じであるが，`g()` の入力の選択の仕方が異なっている。分割 A では連続した遺伝子が同じ入力セットになるように選択しており，分割 B では遺伝子表現上で 4 だけ離れた遺伝子どうしを同じ入力セットとしてまとめている。さて，図の箇所で 2 点交叉が行われた場合，分割 A では世代 t において，半数の処理が前世代と共通しており，再利用可能となる。これに対し，分割 B では世代 t における全ての処理は前世代と異なっており，全く再利用が行われないことが分かる。

一般に，入力となる遺伝子表現の全長を n，構成する遺伝子を x_i $(1 \leq i \leq n)$，サブ関数が入力とする遺伝子数を m と仮定すると，遺伝子表現は s $(= n/m)$ 個のサブセットに分割されて処理される。このとき連続した遺伝子が同じサブセットに含まれるよう，入力サブセット I_j $(0 \leq j \leq s-1)$ を $I_j = \{x_{j\times m+1}, \cdots, x_{j\times m+m}\}$ と設定する必要がある。例えば 2 点交叉では，こうすることで s 個のサブセット中少なくとも $(s-2)$ 個は前世代で処理したものと全く同じとなり，高い再利用率が見込める。これに対し，$I_j = \{x_{j+1}, x_{j+s+1}, \cdots, x_{j+(m-1)s+1}\}$ のように離れた遺伝子が同一入力セットとなるようなサブ関数を設定した場合，再利用率は大幅に低下すると考えられる。

5. 評　　価

5.1 評 価 環 境

評価には，再利用機構を実装した単命令発行の SPARC-V8 シミュレータを用いた。各パラメータを**表 1** に示す。キャッシュ構成や命令レイテンシは，SPARC64-III[15] を参考にしている。再利用表については，まず IORB を一次キャッシュと同サイズの 32KB (32Byte 幅 ×256 エントリ ×4 セット) とし，レイテンシも一次キャッシュと同じと仮定した。L2RB については，2MB (256bit 幅 ×64K エントリ) の CAM とし，

表 1 シミュレーション時のパラメータ

D1 Cache 容量	32 KBytes
ラインサイズ	32 Bytes
ウェイ数	4
レイテンシ	2 cycles
ミス ペナルティ	10 cycles
D2 Cache 容量	2 MBytes
ラインサイズ	32 Bytes
ウェイ数	4
レイテンシ	10 cycles
ミス ペナルティ	100 cycles
Register Window 数	4 sets
Window ミス ペナルティ	20 cycles/set
ロードレイテンシ	2 cycles
整数乗算 〃	8 cycles
整数除算 〃	70 cycles
浮動小数点加減乗算 〃	4 cycles
単精度浮動小数点除算 〃	16 cycles
倍精度浮動小数点除算 〃	19 cycles
IORB サイズ	32 KB
L2RB サイズ	2 MB
交叉率	60.0 %
交叉点数	2
突然変異率	0.1 %
個体数	50
世代数	25 世代
他のパラメータ	GENEsYs の default 値

レイテンシとしてレジスタとの比較に 32Byte/cycle, キャッシュとの比較に 32Byte/2cycle を仮定した。

またロードモジュールとしては，汎用 GA ソフトウェアである GENEsYs[16] 1.0 を gcc 3.0.1 (-O2 -msupersparc) でコンパイルし，スタティックリンクにより生成したものを用いた。古くから広く利用されている汎用 GA ソフトウェアに Genesis[17] があるが，GENEsYs は Genesis を機能拡張したプログラム集である。GENEsYs では個体選択のスキームなどが拡張されており，適合度関数についても，ベンチマークテストや定量的評価によく用いられる De Jong のテスト関数，巡回セールスマン問題，フラクタル関数など，さまざまな標準的な関数が実装されている。

なお GENEsYs では，適合度計算のための遺伝子表現変換として，グレイコードの逆変換 (Degray) と，char 型配列から整数への変換 (Ctoi) を行っている。参考に，この部分のコードを**図 7** に示す。

5.2 結　　果

評価モデルとしては，以下に示す 3 つを仮定し，比較を行った。各モデルが要した実行サイクル数を，GENEsYs が持つ 24 種の適合度関数 (f1〜f24) 全てについて測定した結果を**図 8** に示す。なお，各関数の詳細については文献 16) を参照されたい。

(O) オリジナル (再利用なし)
(R) 再利用
(R') 4 章で示した改良を加えた上で再利用を適用

各適合度関数のグラフは，左から順に (O), (R), (R') が要した実行サイクル数を表しており，それぞれ (O) のサイクル数を 1 として正規化している。また凡例については，cross は交叉，mutate は突然変異，degray および ctoi は遺伝子表現変換，fitness は適合度計算，select は個体選択の処理に要したサイクル数内訳をそれぞれ示している。なお misc は，初期化などのプロセスに要したサイクル数を示している。また GENEsYs は，各世代毎に個体の評価値平均などの計算を行っている。これは必ずしも GA に必要な処理ではないが，今回は GENEsYs における高速化を評価するという意味で，この処理に要する時間もそのまま残した。これをグラフ中では measure としている。全ての凡例は，入力一致比較などの再利用に要したオーバヘッドを含んだサイクル数を表している。

```
void Degray(char* inbuf, char* outbuf,
            register int length) {
  register int i, last;
  for (last=0, i=0; i<length; i++) {
    if (inbuf[i])  outbuf[i] = !last;
    else           outbuf[i] = last;
    last = outbuf[i];
  }
}


int Ctoi(register char* buf, register int length) {
  register int i, answer;
  for (answer=0,i=0; i<length; i++) {
    answer <<= 1;
    answer += *buf++;
  }
  return(answer);
}
```

図 7 GENEsYs の遺伝子表現変換関数

グラフから，適合度関数の計算部分である degray, ctoi，fitness が再利用の適用により大幅に削減されていることが分かる☆。はからずも GENEsYs のソースコードが，4 章で述べたような手法にある程度沿っていたものに関しては，(R) でも高い効果が得られている。また，(R) では再利用の効果が低いものに関しても，(R') によって効果が向上している。結果，(R) でも最大 83%，平均 27% と高いサイクル数削減率が得られたが，(R') により更にこれを最大 86%，平均 38% まで引きあげることができた。f12 や f14 など，再利用による効果がほとんど得られなかったものに関しても，再利用オーバヘッドによる性能低下は発生しなかった。

また，f5，f11，f13，f16，f17 など，(O) において fitness の割合が高い適合度関数，すなわち処理全体に要する時間が他よりも大きい関数に対し，より効果的に再利用が働く傾向があることにも注目されたい。なお，この 5 関数における (R') の平均サイクル数削減率を算出したところ，66% という値になった。

☆ なお，GENEsYs の f11, f12, f14 においては，Degray/Ctoi による遺伝子表現変換の処理は行われない。

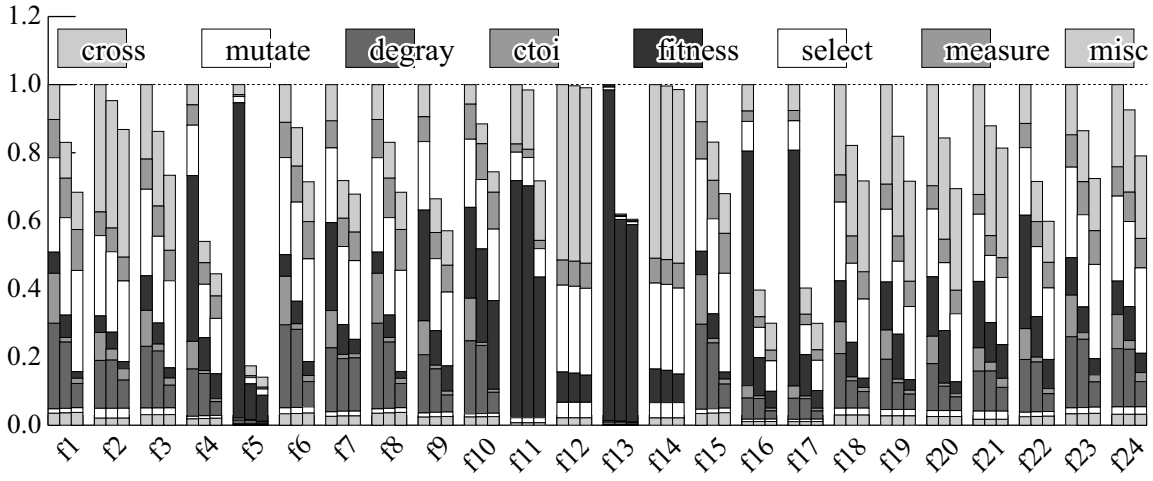

図 8 実行サイクル数の比較（GENEsYs）

5.3 各関数の分割手法と再利用の効果

全ての適合度関数は，複数の構成遺伝子に対する処理を行うサブ関数に分割を行った。サブ関数の入力遺伝子数の目安としては，6 を最大とするできるだけ大きな数を設定した。以下，いくつかの関数に対し，具体的な分割方法を説明する。

f10 は巡回セールスマン問題である。各遺伝子には都市の ID がコーディングされており，各遺伝子が示す都市間の距離を計算する。本稿では，この距離計算の部分を関数として切りだし，再利用対象とした。しかし適合度計算の大部分の処理はソートが占めており，図 8 からも分かるように効果はあまり高くない。

f11 は自己相関関数の一種であり，遺伝子表現上の一定間隔離れた遺伝子間での一致比較を，間隔を変化させながら行う関数である。各間隔における計算結果の自乗和が適合度となる。一致比較自体の処理量は軽微であり，(R) では再利用の効果はあまり得られていない。しかし (R') ではバッファリングを行うことにより，遺伝子表現が完全に一致した場合の処理を削減できた。

f12 は，各構成遺伝子の 0 とのハミング距離の総和を計算する。本稿では 4 遺伝子 の総和を計算する関数の切り出しを行った。しかしハミング距離の計算処理が軽微であるため，関数呼び出しのコストが相対的に大きくなり，顕著な効果は得られていない。f14 も同様の傾向を示している。

f24 は，下記のような式で表される関数である。a および b は定数ベクトルである。

$$\sum_{i=1}^{11}\left(a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right)^2$$

遺伝子長が 4 と短いため，(R) では交叉の影響で同じ入力セットがほとんど出現せず，fitness 部分における高速化は実現できていない。そこで，$x_1(b_i^2 + b_i x_2)$ および $b_i^2 + b_i x_3 + x_4$ を計算する関数を切り出したところ，(R') では fitness の処理時間を約半分に削減することができた。

その他の関数に対しては，1 構成遺伝子 に対する処理量をある程度見込めると考えたため，各構成遺伝子を処理する関数の切り出しも行っている。例えば f1 は正規化した各遺伝子の自乗和を計算する適合度関数である。これを，6 遺伝子を入力とする関数および 1 遺伝子を入力とする関数の 2 段階に切り出した。後者は，1 遺伝子を正規化した後，その自乗を計算する関数となる。こうすることで，遺伝子長 n を s 個のサブセットに分割して処理する関数を定義する場合，2 点交叉では $(s-2)/s$ セットが再利用可能となるが，残りの $2/s$ セットを構成する各遺伝子に関しても，突然変異対象となったもの以外には再利用が適用可能となる。

6. 考　　察

ここでは，再利用による高速化に対し，GA の各パラメータおよび CAM 容量が与える影響について考察する。

6.1 交叉率・突然変異率

GENEsYs では，前世代から変化のなかった遺伝子表現，すなわち交叉および突然変異の対象とならなかった遺伝子表現に関しては，適合度計算を省略する。よって仮に交叉率を変更した場合でも，4.3 節で示した再利用率に対する影響はなく，今回の結果とほぼ同じ結果が得られると考えられる。これに対し，突然変異率が大きくなった場合は再利用率が低下すると考えられるが，一般に GA では突然変異率をあまり大きな値に設定することはないため，プログラム全体の

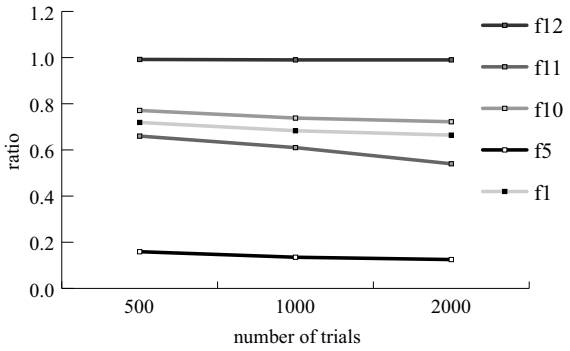

図 9 評価回数とサイクル数の関係

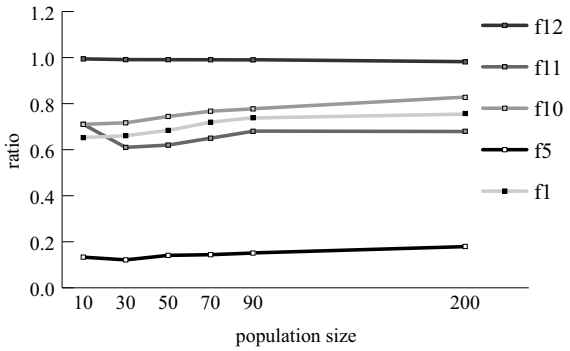

図 10 個体数とサイクル数の関係

速度に対する影響は軽微であろう。

6.2 適合度評価回数

適合度の評価回数が増大するに従い，全体における misc の占める割合が低下する。これにより再利用対象となる適合度計算の割合が増加するため，再利用の効果も大きくなる。**図 9** は，適合度の評価回数を変化させたときの適合度関数 f1, f5, f10, f11, f12 の所要サイクル数を示したものである。適合度計算に要するサイクル数 (fitness, degray, ctoi) が占める割合が大きいものほど再利用の効果が大きいという特徴が，適合度評価回数が大きくなるにつれ明確に現れていることが分かる。

6.3 個 体 数

個体数に関しては，大きくなるに従って遺伝子表現パターンも多種となり，再利用率を維持するためには適合度関数の入力パターンをより多く保存できる必要がある。このため，L2RB を構成する CAM 容量が許す範囲で再利用率は維持されるが，L2RB 溢れが発生するとある程度再利率は低下すると予想できる。

図 10 は，個体数を変化させたときの所要サイクル数を示したものである。この結果では，個体数を 50 以上に増やした場合でも僅かな速度低下が見られるだけである。なお，個体数を増やした場合の速度低下は，再利用テスト回数が増加したことによる再利用オーバヘッドの増大，および隔世世代の遺伝子パターンが再利用表で保持できなくなったことに起因すると考えられる。

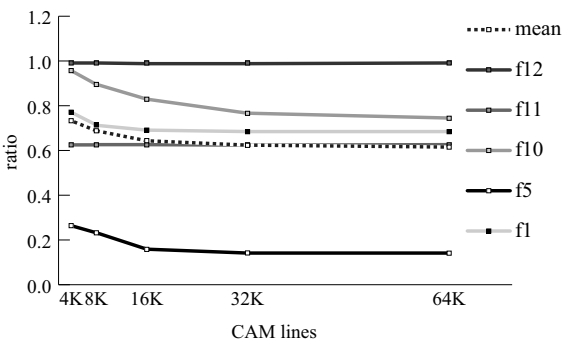

図 11 CAM サイズとサイクル数の関係

個体数を増加させることによって，遺伝子表現がとりうる全パターンのうち再利用表が記憶できる遺伝子表現のパターンの割合は，大きく減少すると考えられる。しかし速度低下が僅かなことより，その影響をほとんど受けてないことが分かる。逆に個体数を減少させた場合でも速度向上が僅かであることより，多くの適合度関数では隔世世代に対する再利用率はそもそも高くないと考えられる。

6.4 交叉アルゴリズム

交叉方法に関しては，N 点交叉の場合，あまり N が大きくなると再利用率はある程度低下すると考えられる。しかし，4.3 節で述べたように，遺伝子長 n を s 個のサブ関数の入力セットに分割した場合，N 点交叉では s のうち $s-N$ が必ず前世代で処理された入力セットと同じとなり，再利用可能となる。よってサブ関数の再利用率はほぼ $(s-N)/s$ となる。このことから，遺伝子表現長 n が十分長ければ，s が十分大きくなり，N による影響は小さく抑えられる。

ただし，一様交叉を用いる場合，再利用の効果は N 点交叉ほどは期待できない。しかしその場合でも，再利用オーバヘッドが小さいことから，再利用を適用した場合でも性能悪化はほとんど発生しない。また，個々の遺伝子に対する処理は，突然変異対象となったもの以外において再利用可能であることに変わりはない。GENEsYs で一様交叉を用いて評価したところ，平均削減サイクル数 11%，最大削減サイクル数 70% という値となった。

6.5 CAM サイズ

図 11 に，CAM サイズを変化させた場合のサイクル数削減率の変化を示す。横軸は CAM の行数，mean は全 24 関数の平均を示している。この結果から，GENEsYs 程度の規模の GA プログラムでは，CAM 行数は 16K エントリ (約 1MB) でほぼ最大性能が得られることが分かる。また，4K エントリ (約 256KB) まで小さくした場合でも，平均 27% という良好な結果が得られた。従来の GA 専用ハードウェアよりも，

比較的小さなハードウェア量で高い効果が得られることが分かる。

7. おわりに

本稿では GA に対し，並列 GA とは異なる高速化のアプローチとして，再利用を適用することによる手法を提案した。GA の処理プロセスについて概説し，一般に最も時間を要する処理である適合度の算出に対して，再利用が効果的であることを述べた。再利用の際，入力となる遺伝子表現のアドレスの違いを吸収するためにバッファリングが必要であることを述べた。また N 点交叉では，遺伝子表現が前世代の遺伝子表現と多くの共通部分を持つことから，適合度計算の関数を分割することで，再利用率を向上させることができることを示した。

汎用 GA ソフトウェアである GENEsYs を用いて再利用の効果を評価した結果，24 種全ての適合度関数において平均 27%，最大 83% のサイクル数を削減することができることを示した。更に，適合度関数の分割を行うことにより，これを平均 38%，最大 86% まで引きあげることができた。また，全体の処理時間に占める適合度関数の割合が高い，本来最も高速化が望まれる関数において，再利用の効果が特に高いことを示し，そのような 5 関数で平均 66% のサイクル数を削減できることを示した。

なお，本稿で提案した再利用による高速化は，従来 GA の高速化手法として多く用いられてきた並列 GA と競合するものではない。よって，双方を組み合わせることによって，更なる高速化が可能となると考えられる。

参考文献

1) Cantú-Paz, E.: *Efficient and Accurate Parallel Genetic Algorithms*, Genetic Algorithms and Evolutionary Computation, Vol. 1, Kluwer Academic Publishers (2000).
2) Harik, G. R., Lobo, F. G. and Goldberg, D. E.: The Compact Genetic Algorithm, *IEEE Transactions on Evolutionary Computation*, Vol. 3, pp. 287–297 (1999).
3) Lobo, F. G., Lima, C. F. and Mártires, H.: An Architecture for Massive Parallelization of the Compact Genetic Algorithm, *Proc. of the Genetic and Evolutionary Computation Conference* (2004).
4) Scott, S. D., Samal, A. and Seth, S.: HGA: A Hardware-Based Genetic Algorithm, *Proc. of the ACM/SIGDA 3rd Int. Symp. on Field-Programmable Gate Arrays*, pp. 53–59 (1995).
5) Kajitani, I., Hoshino, T., Nishikawa, D., Yokoi, H., Nakaya, S., Yamauchi, T., Inuo, T., Kajihara, N., Iwata, M., Keymeulen, D. and Higuchi, T.: A Gate-Level EHW Chip: Implementing GA Operations and Reconfigurable Hardware on a Single LSI., *ICES*, pp. 1–12 (1998).
6) Shackleford, B., Snider, G., Carter, R. J., Okushi, E., Yasuda, M., Seo, K. and Yasuura, H.: A High-Performance, Pipelined, FPGA-Based Genetic Algorithm Machine, *Journal of Genetic Programming and Evolvable Machines*, Vol. 2, pp. 33–60 (2001).
7) 津邑公暁, 笠原寛壽, 清水雄歩, 中島康彦, 五島正裕, 森眞一郎, 富田眞治: 大容量汎用 3 値 CAM を用いた並列事前実行機構の効率的実現, 先進的計算基盤システムシンポジウム SACSIS2004 論文集, 情報処理学会, pp. 251–259 (2004).
8) Sodani, A. and Sohi, G. S.: Dynamic Instruction Reuse, *Proc. 24th International Symposium on Computer Architecture*, pp. 194–205 (1997).
9) Huang, J. and Lilja, D. J.: Exploring Sub-Block Value Reuse for Superscalar Processors, *PACT* (2000).
10) Connors, D. A., Hunter, H. C., Cheng, B. and Hwu, W. W.: Hardware Support for Dynamic Activation of Compiler-Directed Computation Reuse, *9th ASPLOS*, pp. 222–233 (2000).
11) 津邑公暁, 清水雄歩, 中島康彦, 五島正裕, 森眞一郎, 北村俊明, 富田眞治: ステレオ画像処理を用いた曖昧再利用の評価, 情報処理学会論文誌：コンピューティングシステム, Vol. 44, No. SIG 11(ACS 3), pp. 246–256 (2003).
12) 竹村尚大, 津邑公暁, 中島康彦, 五島正裕, 森眞一郎, 富田眞治: MP3 エンコーダの分析及び曖昧再利用の適用による高速化, 情処研報 2003-ARC-152 (HOKKE 2003), pp. 145–150 (2003).
13) Paul, R.: *SPARC Architecture, Assembly Language Programming and C*, Prentice-Hall (1999).
14) Jong, K. D.: *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*, PhD Thesis, University of Michigan (1975).
15) HAL Computer Systems/Fujitsu: *SPARC64-III User's Guide* (1998).
16) Bäck, T.: GENEsYs 1.0. Software distribution and installation notes (1992).
17) Grefenstette, J. J.: GENESIS: A System for Using Genetic Search Procedures, *Proc. of the 1984 Conference on Intelligent Systems and Machines*, pp. 161–165 (1984).