並列化および再利用による GA の高速化

新美明仁[†] 池内康樹[††,☆] 鈴木郁真[††,☆☆]
津邑公暁[†] 松尾啓志[†] 中島康彦[†††]

我々は，再利用および並列事前実行を用いた高速化手法を提案している．しかし，並列事前実行では投機に多数のコアを割り当てても更なる高速化は見込めず，逆に速度の低下を招くこともある．また，GA のように並列事前実行の効果が得られないプログラムも存在することから，全てのコアを投機に割り当てる方法は有効とは言えない．この様な場合に，従来投機に割り当てていたコアをメインコアとしても用いることで並列処理を行い，この投機コアとメインコアの数を動的に変えて実行を行う手法が考えられる．本稿ではこの効果を検証するため，2 つのメインコアが再利用表を共有する機構を実装し，GA において評価，検討を行った．汎用 GA ソフトウェア GENEsYs の，24 種の適合度関数に対しコア数 2 つで再利用を行った場合，コア数 1 つで再利用を行わない場合に比べ最大 14.2 倍，平均 3.1 倍の高速化を実現した．このように GA のような並列化が容易なプログラムについては，再利用と並列処理を組み合わせることで更なる効果を得られることが分かった．

Speed-up of GA by Parallelization and Auto-Memoization

Akihito Niimi,[†] Yasuki Ikeuchi,[††,☆] Ikuma Suzuki,[††,☆☆]
Tomoaki Tsumura,[†] Hiroshi Matsuo[†]
and Yasuhiko Nakashima[†††]

We have proposed parallel early computation, an asymmetrical speculative multi-threading with auto-memoization technology. However, some programs benefit not so much from parallel early computation bacause of low hit-ratio of input speculation, and other programs such as GA benefit little. Hence, we aim the combination of parallel processing and early computation. This system dynamically changes the ratio between the number of main cores and speculative cores according to the behavior of target programs, and two or more main cores share the memoization information. This paper proposes a speedup technique with the dual-core auto-memoization processor for the fitness calculation of GA programs. Through the result of an evaluation with GENEsYs, a well-known GA software, we show that dual-core auto-memoization processor gains significantly large speedup, up to 14.2-fold and 3.1-fold on average.The method of combining auto-memoization technique with parallel processing is effective in such as genetic algorithm programs.

1. はじめに

命令レベル並列性 (ILP) に基づく高速化手法は頭打ちになりつつある現在，我々は ILP に依存しない再利用を用いた高速化手法を提案している．再利用とは，先行命令列の実行結果を保存しておき，同一入力値で実行が行われた際に，過去の実行結果を再利用することによって，高速化を行う技術である．

また，これに投機的マルチスレッディング (Speculative Multi-Threading) を組み合わせた並列事前実行を提案している[1)]．この方法は，予測に基づいて事前に実行した多数の投機実行スレッドの結果を，通常実行スレッドが同一入力を検出した際に自身の過去の計算結果と同様に再利用するものである．これまでの研究で，GA のような入力パターンの局所性が高いプログラムにおいては再利用のみで最大 83 %[2)]，並列事前実行では一般的なベンチマークプログラムにおいても Stanford で 42 %，SPEC CPU95 で 30 %の平均サイクル数削減率を実現している．しかし，並列事前実行を適用しても高速化を図ることのできないプログラムも存在する．例として，GA が挙げられる．並列事前実行は予測された入力値に基づいて投機実行を行うため，関数の入力が単調に変化する場合では高速化

† 名古屋工業大学
Nagoya Institute of Technology
†† 豊橋技術科学大学
Toyohashi University of Technology
††† 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
☆ 現在, (株) ACCESS
Presently with ACCESS Co.,Ltd.
☆☆ 現在，トヨタ自動車（株）
Presently with Toyota Motor Corp.

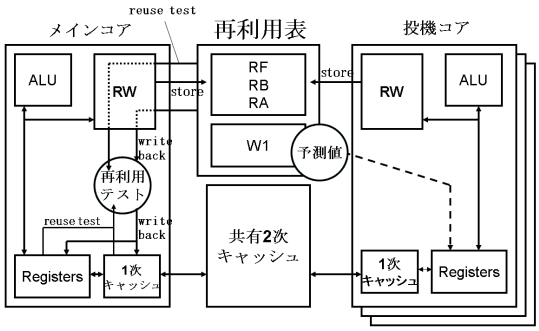

図 1 再利用と並列事前実行機構の構成

が図れるが，GA の様なプログラムは高速化されない．

一方で，近年 CPU に搭載されるコア数が増加している．これまでプロセッサは，動作周波数の向上，プロセスの微細化，トランジスタの高集積化などの方法で性能向上を図ってきた．微細化により，同じサイズのプロセッサに搭載できるトランジスタ数が増加し，動作周波数も向上する．逆に消費電力は，回路を流れる電流が減少するため一世代前とあまり変わらない．しかし，微細化が進むにつれリーク電流が増加するなどの問題から消費電力が増加してしまい，それに伴う発熱の増加が問題となっている．そこで現在では，プロセッサの搭載コア数を増やすことで動作周波数あたりの性能向上を図る手法が注目されている．

並列事前実行はメインコアと投機コアを用いることで高速化を図る手法である．しかし多数のコアを投機コアに割いた場合では，遠く先まで予測することになるため予測のヒット率は低下する．また GA のようなプログラムにおいては少ない投機コアでも効果が得られないことから，多数のコアを投機のみに用いることは無意味である．そこで本研究では，並列事前実行の効果の得られないプログラムにおいては，従来投機に割り当てていたコアをメインコアとして用い，並列処理することによる高速化手法を提案する．具体的には，プログラムに適した構成で実行するために，マルチコア CPU においてメインコアと投機コアの数を動的に変化させることを想定した．

本稿では，従来投機に割り当てていたコアをメインコアとして用いた場合の並列処理について考え，2 つのメインコアが再利用表を共有する機構を提案し，GA においてその有効性を検証した．再利用は並列化とは独立な高速化技術であるため，単一コアで再利用した場合に比べて更なる高速化が期待出来る．

2. 再利用および並列事前実行

2.1 再 利 用

再利用は，既に実行した関数などの入出力を記憶しておき，次に同じ入出力の関数を実行する場合，既に実行済みの出力を用いることにより，関数の実行自体を省略し高速化を図る手法である．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や，コンパイル時に再利用のための情報を埋め込むもの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，これまでメディア処理を始め GA などのプログラムにおいて有効な結果を示している．再利用機構と並列事前実行の構成を**図 1** に示す．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，関数とループを仮定している．具体的には，関数に含まれる命令では，call/jump 命令の分岐先から return 命令を再利用対象として検出する．ループの場合には，1 度後方分岐命令が検出され，その分岐が成立した後，再び同じ後方分岐命令が検出されたときその区間を再利用対象とする．

再利用機構は，入出力を記憶する再利用表と呼ばれる記憶領域と再利用表への書き込みバッファRW を持つ．図 1 において再利用表中の RF は，再利用可能な関数の管理表を示し，RB は入出力を管理する表，RA は入力セットのアドレスを記憶する表，W1 は出力値を記憶する表をそれぞれ示している．我々の再利用機構は，再利用対象となる命令区間を検出すると，再利用表から入出力が一致するエントリを検索し，完全に一致するエントリが見付かったら，再利用表から一致したエントリに対応する出力を W1 からキャッシュとレジスタに書き戻すことで命令区間の実行を省略する．また，一致するエントリが見付からなかった場合は，通常どおり実行を行い，その際のレジスタおよびキャッシュの参照を入力，書き込みを出力として RW に記憶しておき，命令区間を最後まで実行したあと，まとめて対象命令区間の入出力セットを再利用表に登録する．ここで，我々は再利用表を CAM (Content-Addressable Memory) を用いて構成することを仮定している．これは，入力一致比較を行うときの再利用表検索オーバヘッドを小さく抑えるためである．再利用表の検索は，命令区間を実行する度に行わなければならず，連想検索を高速で行える CAM が必要である．

2.2 並列事前実行

我々が提案する再利用は，既存のプログラムを変更せずに高速化を図ることのできる手法であるが，どんなプログラムでも効果があるわけではない．関数の引数が単調に変化する場合などが該当する．この様な場合では並列事前実行を用いることができる．並列事前実行 (Parallel Early Computation) は，再利用表に登録されている命令区間に対してこれから出現するで

あろう入力を予測し，メインコア（通常実行スレッド）と並行して投機コア（投機実行スレッド）において，その予測された入力を用いて投機実行し，結果を再利用表に登録する手法である．並列事前実行は，単調に変化する入力を予測することができるため，再利用の効果を得ることができる．またこの手法は，複数のスレッドを同時実行できるマルチコア CPU などにおいて効果を発揮するため，今後 CPU の搭載コア数が増加していくことを考えると，有効な手法であると言える．しかし，増加したコア全てを投機に用いても高速化は図れない．これは，予測が一般的に遠い先の予測ほどより当たる確率が下がると言え，複数の投機コアを使うことでより先まで予測しても，再利用できないエントリが再利用表に登録さることで再利用率の低下を招く恐れがあるからである．また，入力が単調に変化しないプログラムも存在することも理由として挙げられる．例として，GA は関数の入力が単調に変化せず，並列事前実行では高速化できない．そこで，投機コアに割り当てるコアをメインコアに割り当て並列処理を行うことで高速化を図る．本稿では GA を対象とした並列処理を考える．

3. 遺伝的アルゴリズム

3.1 概　　要

GA は，生物の進化のプロセスを計算機上にエミュレートすることで，効率的に解の探索を行うことを目的として使用されるアルゴリズムである．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，遺伝子表現で表されるさまざまな個体同士で生殖活動を行わせ，次の世代の子孫を作り出す．それらの個体を適合度に応じて選択・淘汰を繰り返しながら，最適な解を探索する．GA では生殖，適合度計算，個体選択というプロセスで 1 世代の処理を構成し，それを繰り返すことで最適な解を探索する．GA のプロセスを**図 2** に示す．

3.2 GA のプロセス

3.2.1 生　　殖

生殖の処理では，遺伝子表現に対し交叉（crossover）および突然変異（mutation）の操作を行い，次の世代の遺伝子表現を作る．交叉は，まず一定確率（交叉率）で個体を選択し，選択された個体同士で遺伝子配列の一部をを入れ換えることによって次の世代の個体とする操作である．交叉の方法には，遺伝子の 1 箇所をランダムに選び，その箇所で遺伝子を交換する 1 点交叉，ランダムな N 点で遺伝子を交換する N 点交叉，同位置の遺伝子からランダムに選択する一様交叉などがある．ここでは，2 点交叉の例を**図 3** に示す．2 組の遺伝子表現を親として選択し，遺伝子表現上の任意の 2 点間の遺伝子を親同士で入れ換え次の世代の遺伝子表現を生成する方法である．

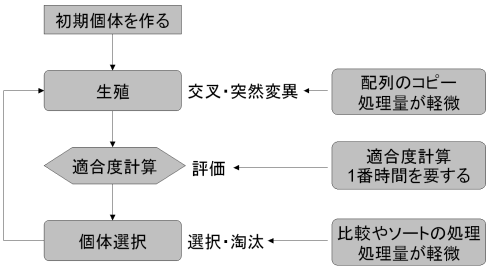

図 2　GA のプロセス

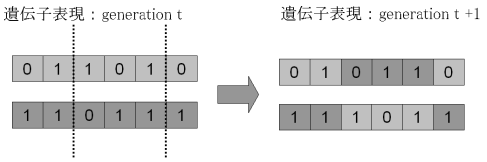

図 3　2 点交叉

突然変異は，一定の確率（突然変異確率）で全体の中から個体を選択し，その遺伝子に変更を加える操作である．遺伝子が単純な 0/1 で表現される場合，ビットの反転を行うことで遺伝子に変更を加える．

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

3.2.2 適合度計算

適合度計算とは，生殖で生成された各個体の適合度の計算を行うもので，生成された個体の評価を行う．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，遺伝子表現に対して何らかの変換がなされたうえで，適合度の計算を行う．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，適合度計算は省略される．

適合度計算は，GA においては最も時間を要する処理である．よってこの部分を高速化できれば，GA 全体の処理時間を大きく削減できる可能性がある．適合度計算で処理する遺伝子表現には，前世代の遺伝子表現から交叉により生成されたことで，遺伝子表現の一部が常に前世代の個体と共通するという特徴がある．つまり，適合度計算において構成遺伝子の同一部分に対する処理が含まれている場合，その処理は再利用が可能であると考えられる．よって適合度計算は再利用に適した処理であると言え，再利用できれば大幅に時間の削減が期待できる．

3.2.3 個 体 選 択

個体選択とは次の世代にどの個体を残すかを決めるための処理である．個体選択には，単純に適合度の高い N 個の個体を残すエリート選択，適合度に基づいた

選択確率で次世代へ個体を残すルーレット選択，集団から完全にランダムで個体を選択する方法などがある．

個体の選択では，すでに計算された適合度に基づく比較やソートが多くを占めるため，生殖処理と同様に処理量が軽微であり，再利用の適用による効果は低いと考えられる．

3.3 再利用の適用

すでに，我々は再利用による GA の高速化を提案している．[2] GA の処理には，前世代と現世代とで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できる．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することは少ないため，このままでは高い再利用効果を得ることは難しいと考えられる．そこで，再利用効果を最大限引き出すためのプログラミング手法を適用する．

遺伝子の格納アドレス

前述のように，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，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うえで適合度計算を行うようにする．これにより，再利用率向上を図ることができる．

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，遺伝子表現全体を処理対象とする適合度計算関数は，入力値が過去のものと完全に一致することはなく，再利用の効果が望めない．ただし適合度計算は，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場合がある．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ことで，そのサブ関数に対する再利用の効果が望める．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，再利用率も引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も減少し，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．

サブ関数の入力遺伝子数設定を含む適合度計算関数の構造変換は，元の構造や計算の内容に強く依存する．従って一般的な変換アルゴリズムやそれに基づく自動化は現時点では実現できていない．しかし本稿では後

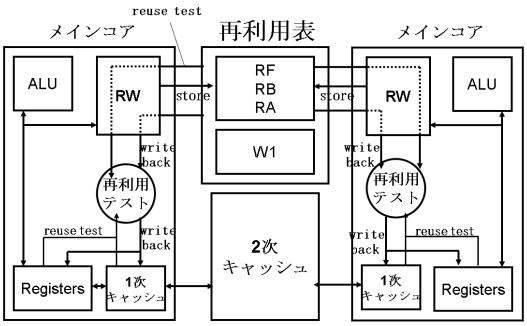

図 4 提案手法の概要

述のように GENEsYs[3] の 24 種類の関数全てについて，以下のガイドラインに沿った変換が可能であることと，その多くに効果が表れること見出している．

(1) 関数が総和などの縮約演算ループを主構造とする場合には，適切な多重度 n に基づくループアンローリングを行い，その結果得られるループ 1 回分をサブ関数とする．

(2) (1) で得られたサブ関数，あるいはアンローリング不能の場合は元の関数の計算量を見積もる．見積りが一定値を越えた場合には，計算式を分割して，(最小単位は 1 遺伝子入力) 各々をサブ関数とする．

前世代との遺伝子の共通部分

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．特に交叉方法が N 点交叉の場合，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．2 点交叉では連続する遺伝子をセットとしてサブ関数の入力とすることで高い再利用率が見込める．

4. 提 案 手 法

4.1 概　　要

ここまで，GA に再利用を適用することで高速化が図れることを述べてきた．一方で，GA は並列化が容易なプログラムだという特徴も持っており，これまで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[4][5]．GA の並列処理は遺伝子表現の集まりである個体集団を 2 つのプロセッサコアに均等に割り当てることにより実現する．これは，GA が遺伝子表現の違いで処理の量が変わることが少なく，個体集団の分配だけでほぼ均等に処理量を分割することが可能だからである．

従来の並列事前実行機構ではメインコアが単一で残りを投機コアとしていたが，本稿ではメインコアを複数とする．よって，複数のコアが再利用表を共有することによりそれぞれ再利用が可能となるばかりでなく，それぞれのコアが再利用表に登録したエントリを互い

に利用することが期待される．

4.2 実　　装

今回の実装では GA への適用を考え，2 つのメインコアで 1 つの再利用表を共有する機構の実装を行った．投機コアの実装は簡単化のため省略した．本実装の構成を**図 4** に示す．実装に当たっては，並列事前実行プロセッサのシミュレータをベースにメインコアを複数化し，再利用表をお互いに共有できるようにした．

さて，再利用表の容量には限りがあるため，再利用表が溢れる前にある一定の時間で最も再利用されてから時間の経過しているエントリを消去するパージ機構を実装している．再利用機構はリングカウンタにより時刻管理を行っており，このカウンタは再利用表に一定数のエントリが追加されるごとにインクリメントされる．各エントリは TSID とよぶタイムスタンプを保持しており，再利用表に登録された時や再利用が行われたときに，現在の時刻をここに保持する．また時刻がインクリメントされた時点で，もっとも古い時刻を TSID に持つ全てのエントリを再利用表から削除する．今回実装の都合上から，リングカウンタをコア間で共有した．動作はそれぞれのコアが再利用表に登録する時点で，カウンタの値から再利用表中の最も古い TSID を持つエントリを削除する．リングカウンタは 2 つのコアがそれぞれ進めるため，1 つのコアで実行する場合より約 2 倍多くパージが起こるが，2 つのコア両方が再利用出来るエントリは再利用が行われたときに TSID を更新するため再利用表に残りやすい．結果両方のコアで再利用出来る可能性が高いものだけが残ることになり有効に再利用できると言える．また今回は簡略化のため，2 次キャッシュは共有せずに 2 つのコアそれぞれが持つという形で実装を行った．

5. 評　　価

5.1 評 価 環 境

評価には，メインコアを 2 つにした並列事前実行プロセッサシミュレータを用いた．測定時のパラメータを**表 1** に示す．キャッシュや命令レイテンシは，SPARC64-III[6] を参考にしている．また，評価プログラムには汎用 GA ソフトウェアである GENEsYs を用いた．GENEsYs には，GA のベンチマークや定量的評価に良く用いられる 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の標準的な関数をはじめとする，24 種の適合度関数が実装されている．今回は適合度関数計算を再利用対象とし，以下の場合について評価を行った．

- (1N) コア数 1，再利用なし
- (1R) コア数 1，再利用あり
- (2R) コア数 2，再利用あり

GENEsYs のパラメータを**表 2** に示す．(1N)，(1R) では，初期個体を 50 個として測定を行った．(2R) では個体を複数のグループに分け，それぞれを独立に処理する．今回の実験では 50 個の個体をそれぞれのコアに割り当てることを想定し，コア数 2 の場合は初期個体数をそれぞれのコアで 25 個とした．

表 1　シミュレータパラメータ

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
RW サイズ	32 KB
再利用表サイズ	2 MB

表 2　GA のパラメータ

交叉率	60 %
交叉点数	2 点
突然変異確率	0.1 %
個体数	50 個
世代数	25 世代
他のパラメータ	default 値

5.2 結　　果

実験の結果を**図 5** に示す．図 5 の横軸は GENEsYs の持つ 24 種類の適合度関数を示している．縦軸は，サイクル数の比を表しており，左から順に (1N)，(1R)，(2R) をそれぞれ示し，(1N) のときを 1 として正規化している．なお，(2R) のグラフは 2 つのコアの内，サイクル数の多い方を示した．凡例は，GENEsYs の持つ処理時間の内訳をサイクル数の比で示している．GENEsYs の主な処理は，交叉 (Cross)，突然変異 (Mutate)，グレイコードの逆変換 (Degray)，char 型配列から整数への変換 (Ctoi)，適合度計算 (Fitness)，個体選択 (Select) に分けられる．Mutate，Degray は遺伝子表現の変換に用いられる処理である．また GENEsYs は，各世代ごとに個体の適合度平均などの計算を行っている．この処理を Measure とし，上記以外の初期化などの処理を Misc とした．

図 5 より (2R) は (1N) に対して最大 93 %のサイクル数削減，14.2 倍の高速化を実現したことになる．さらに平均では 68 %のサイクル数削減，3.1 倍の高速化となった．まず Fitness を見ると，すべての関数で再利用の効果が得られサイクル数が削減されていること

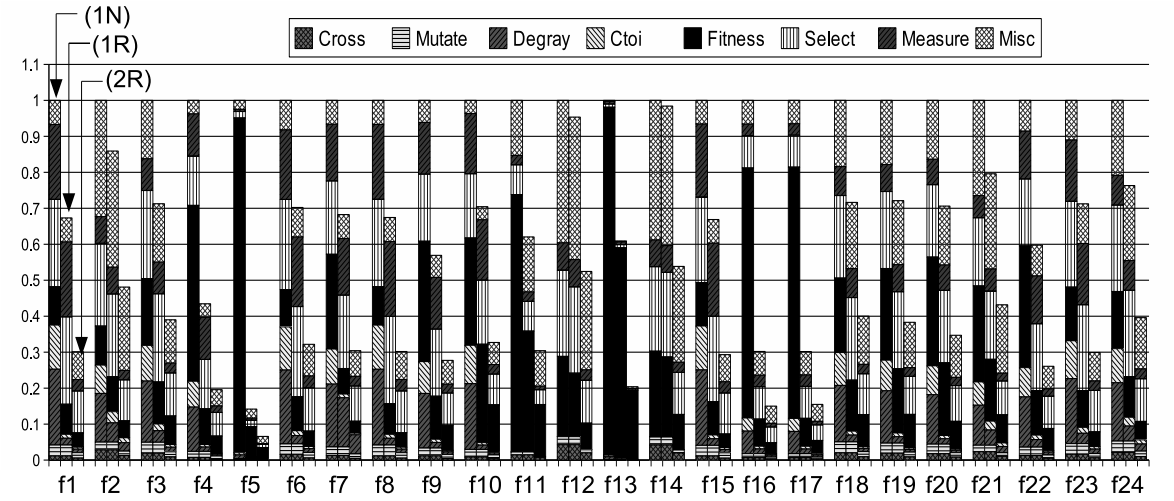

図 5 評価結果

が分かる．特に f4，f5，f11，f13，f16，f17 は，総サイクル数に対して Fitness の割合の高い関数であり，提案手法である (2R) は (1R) の半分以下にまでサイクル数の削減を実現している．これは，一方のコアが再利用表へ登録したエントリを他方のコアが再利用することができているためである．Degray，Ctoi でも同様の効果が得られていることから，GA のような繰り返し同じ処理を行うプログラムにおいて非常に効果的な手法であることが確認できた．また，そのほかの処理の内訳（Cross，Mutate，Select) を見ると (2R) では (1N) の約半分になっていることがわかる．これは初期個体をそれぞれのコアに分けているため各コアの処理量も半分になっているからである．しかし，Misc は (2R) が (1N) の 2 倍以上を要している．これは初期化等の処理が個体数に依らないためである．にもかかわらず f12，f14 などの Misc の割合が高いプログラムにおいても (2R) は (1N) の 2 倍に近い性能を実現できており，提案手法の有効性が示せた．

6. おわりに

本稿では GA において，並列化と再利用に着目し，2 つのコアを用いて再利用表を共有する手法を提案した．GENEsYs での評価においてコア数 2 で再利用ありではコア数 1 で再利用無しと比較して最大 14.2 倍，平均 3.1 倍の高速化を実現した．本来並列化しただけの手法では最大 2 倍の高速化しか実現できないが，提案手法ではほぼ全ての関数において 2 倍以上の高速化を実現したことから，再利用表を 2 つのコアで共有することで，更なる再利用効果が得られることを確認した．提案手法は並列事前実行の効果が得られるプログラムでは投機コアを実行に多く割り当て，効果の得られないプログラムでは投機コアの数を減らし，その分をメインコアに割り当てるように投機コアとメインコアの数を実行時に動的に変えることを想定している．よって今後はプログラムに応じて，投機コアとメインコアの数の割り当てを動的に変更する手法について検討する予定である．

また，今回は GA を対象として評価したが，GA 以外のプログラムに対する本提案手法の有効性も検証して行きたい．

参 考 文 献

1) Tsumura, T., Suzuki, I., Ikeuchi, Y., Matsuo, H., Nakashima, H. and Nakashima, Y.: Design and Evaluation of an Auto-Memoization Processor, *Proc. of Parallel and Distributed Computing and Networks*, pp.245–250 (2007).
2) 鈴木郁真，池内康樹，津邑公暁，中島康彦，中島浩：再利用による GA の高速化手法，情報処理学会論文誌：コンピューティングシステム， Vol.46, No.SIG 16(ACS 12), pp.129–143 (2005).
3) Bäck, T.: GENEsYs 1.0. Software distribution and installation notes (1992).
4) 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).
5) Cantú-Paz, E.: *Efficient and Accurate Parallel Genetic Algorithms*, Genetic Algorithms and Evolutionary Computation, Vol.1, Kluwer Academic Publishers (2000).
6) HAL Computer Systems/Fujitsu: *SPARC64-III User's Guide* (1998).