システムソフトウェアと
オペレーティング・システム 67－1
(1994. 12. 8)

並列計算機システム向きOSのスケジューラへのGAの適用

柿白 秀雄　新實 治男　柴山 潔

京都工芸繊維大学 工芸学部 電子情報工学科

京都市左京区松ヶ崎御所街道町

あらまし

我々は、並列計算機向きOSのプロセススケジューラに遺伝アルゴリズム(GA)を適用した。その際、GAを用いたOSのスケジューリングモードとして、単一タスクにのみ注目した静的スケジューリングモード、複数タスクに対する動的スケジューリングモードの、2つのスケジューリングモードについて考察した。いくつかの例題について、シミュレーションを行なった結果を報告している。その結果とランダム法での結果とを比較した。どちらのモードともGA法の方がよい結果を得て、GA法の高い準最適解探索能力が示せた。

和文キーワード　GA法, プロセススケジュール, 並列計算機向きOS

Genetic Algorithm Approach to Scheduling Problems of the Operating Systems on Parallel Computers

Hideo KAKISHIRO,Haruo NIIMI,Kiyoshi SHIBAYAMA

Department of Electronics and Information Science, Faculty of Engineering and Design, Kyoto Institute of Technology

gosyokaido-cyo,matsugasaki,sakyo-ku,kyoto

Abstract

This paper describes a method based on Genetic Algorithm to solve process scheduling problems of the Operating Systems on parallel computers. In this method, we propose two scheduling modes. The first one is static scheduling mode on which a single task is scheduled independently of the other tasks. The second one is dynamic sheduling mode on which each task is scheduled on its invocation. This paper also describes effectiveness of Genetic Algorithm through several simulation results.

英文 key words　Genetic Algorithm, process schedule, parallel machine OS

1 はじめに

並列計算機向き OS は、従来の OS の機能に加えて、システム中の多数のプロセッサを有効に使用するための管理機能、すなわち、複数プロセッサ間でのプロセススケジューリング機能が必要である。また、このスケジューリング機能の性能が、システム全体のパフォーマンスに影響を及ぼすと言える。

この様なプロセススケジューリング問題は、いわゆる組合せ最適化問題として定式化される。組合せの総数が少ないうちは、分枝限定法のような、数理計画的手法を用いるのが効果的である。しかし、並列計算機のプロセッサ数の増加や、その上で実行されるプログラムの複雑化に伴うプロセスの増加といった傾向を考慮すると、計算量の点から上記の方法は非現実的である。また、OS のプロセススケジューリング問題として考えると、必ずしも最適解である必要性はなく、むしろ準最適解を効率的に求めることの方が重要な場合もある。

本稿では、並列計算機向き OS のスケジューラのプロセススケジューリングのアルゴリズムに、**遺伝アルゴリズム (GA)** を適用することを考える。そして、OS のスケジューリングモードとして静的スケジューリングモード、動的スケジューリングモードの 2 つのモードについて考察し、シミュレーション結果に基づいて評価を行なった。

2 遺伝アルゴリズムの概要

遺伝アルゴリズム (Genetic Algorithm : GA) は、自然界における生物の進化のメカニズムを模擬する人工モデルである。GA の全体の手順を以下に示す。

Step 1 世代を $t = 0$ として、N個の個体を適当に生成し、初期個体群 $X(0)$ を設定する。ここで個体とは、解となりうる組合せを表現したもので、一般に行列やベクトルで表記される。

Step 2 各個体の目的関数、及び適応度を計算し、適応度に依存した一定のルールで個体の淘汰を行なう。すなわち適応度の低い個体のいくつかは、適応度の高い個体に置き換えられる。

Step 3 一定の確率で個体に交叉や突然変異を起こし、新しい個体を生成する。これらによって、古い個体と置き換わることで、新しい世代の個体群 $X(t+1)$ が生成される。

Step 4 終了条件が満たされなければ、$t = t+1$ として **Step 2** へ戻る。終了条件が満たされれば、目的関数の値を求め、その値が最小の個体を準最適解とする。

本稿において、上記の「個体」は、プロセススケジューリングの組合せパターンの 1 つを表す。また、GA の設定要素として、個体数 N、淘汰の規則、交叉を起こす確率、突然変異を起こす確率、終了条件がある。これらの要素は、スケジューリング中不変でなくても良い。

この様に、GA そのものは何ら複雑ではなく、様々な問題に対して適用が容易である [1]。

3 スケジューリングモード

スケジューリングモードの説明に先立ち、まず、本稿で扱う語句を以下に説明する。

プロセス (Process) プロセッサ上で実行されるロードモジュールの単位。本稿では、プロセッサ上でのプロセスのさらなる細分化は、考えないものとする。

タスク (Task) 1 つのロードモジュールにまとめられたプロセス群のこと。タスクの起動とは、実行形式ファイルの起動と言い換えてもよい。

プロセスチャート (Process Chart) 図 1 に示すもので、タスク中のプロセス群の実行の順序の関係を表す。ノード内の数字はプロセス番号で、右横の添字はプロセスの動作時間、アークはデータの流れる方向を表す。例えば、プロセス 0 について、実行終了後、アークの方向のプロセス (プロセス 1 とプロセス 2) にデータの受渡しが行なわれることを示す。通信はその他のタイミングでは

行なわれないこととする。また、スケジューリングでは、1プロセッサ当り1プロセスが割り当てられる。

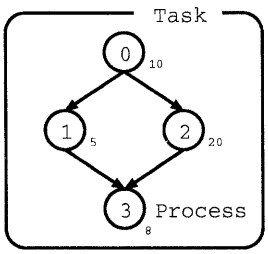

図 1: プロセスチャートの例

スケジューリングでは、プロセスチャートが重要な情報となる。しかし、実行形式ファイルによっては、プロセスチャートが実行前からわかっているとは限らない。プロセスチャートは、わかっている場合には、実行形式ファイル中にタスクの情報とともに存在しているものとする。これらの情報は、コンパイラが実行形式ファイルを出力する時に同時に出力するものとする。

3.1 静的スケジューリングモード

静的スケジューリングモードでは、タスクが起動されると、まず、静的スケジューラによって、指定された数のプロセッサが確保される。この確保されたプロセッサは、タスク終了まで独占して用いられる。次に、起動されたタスクのプロセス群がスケジューリングされ、確保されたプロセッサ上で実行される。つまり、このモードにおいて、実行されるタスクは、他のタスクの実行などの影響を受けずに実行されると仮定されている。

また、このモードでは、スケジューリング時の情報として、プロセスチャートを必要とするので、プロセスチャートが明らかでない実行形式ファイルは実行できない。

以下に、コマンドの起動から終了までの静的スケジューラの動作の様子 (**Step 2** から **Step 6**) を示す。

Step 1 静的スケジューリングモードでタスク (実行形式ファイル) が起動される。

Step 2 スケジューリングの対象となるプロセッサを確保する。

Step 3 実行するタスクのプロセスチャートを読み込む。

Step 4 スケジューリングを行なう。

Step 5 スケジューリングの結果をもとに、実際にプロセスを割り当て実行する。

Step 6 確保したプロセッサを解放する。

Step 7 タスクの終了。

3.2 動的スケジューリングモード

動的スケジューリングモードでは、次々に起動されるタスク中のプロセス群を、システムの状況に応じて、動的スケジューラがスケジューリングして、適当なプロセスに割り当て、実行する。一般的には、このモード上でタスクの実行が行なわれると考えるべきである。

このモードは、静的スケジューリングモードのようにタスクをいかに速く実行させるかというよりもむしろ、多くのプロセッサを多くのタスクで有効に利用するという点を重要視したモードである。したがって、タスクごとのプロセッサの確保は行なわない。また、スケジューリング時の情報としては、プロセスチャートの様なタスク全体のプロセスの流れやプロセスの時間のようなマクロな情報は扱わず、実行中の各プロセスとその子プロセスに関するミクロな情報を扱う。

以下に、一例として図1の様なプロセスチャートをもつコマンドが1つだけ実行された時の、起動から終了までの動的スケジューラの動作の様子を示す (図2も併せて参照)。

Step 1 動的スケジューリングモードでタスク (実行形式ファイル) が起動される。先頭のプロセス 0(p0 と表す) が、スケジューリング対象となる。

Step 2 p0 が適当なプロセッサに割り当てられて実行される。そのとき、p0 の子プロセス p1,p2 がスケジューリング対象となる。

Step 3 p0 が終了して、p1,p2 が実行可能になる。ここでは、p1 だけ適当なプロセッサに割り当てられたとする。そのとき、p1 の子プロセス p3 がスケジューリング対象となる。

Step 4 p2 が適当なプロセッサに割り当てられて実行される。

Step 5 p1 が終了する。まだこの段階では、p3 は実行可能状態にはならない。

Step 6 p2 が終了して、p3 が実行可能になり適当なプロセッサに割り当てられて実行される。

Step 7 p3 が終了してタスクが終了する。

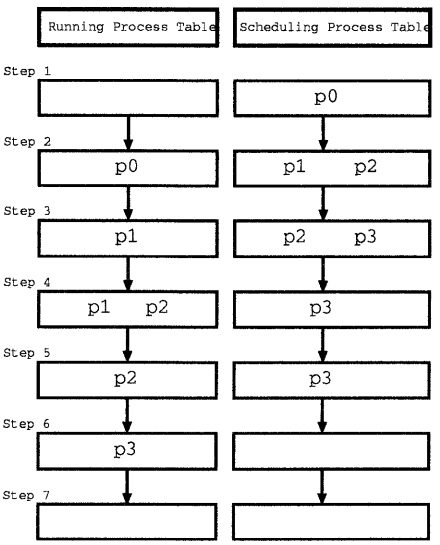

図 2: 動的スケジューラの動作例

4 問題の設定

プロセススケジューリング問題は、N 個のプロセスを M 個のプロセッサに割り当てる組合せの中から、最も適したものを選ぶことである。本稿では、1 プロセッサあたり 1 プロセスが割り当てられるものとし、各モードについてそれぞれ目的関数を設定してその関数値が小さい程、適したものであると判断する。

また、シミュレーションを行なうにあたり、以下の仮定を行なった。

ハードウェア $x \times y$の 2 次元メッシュをトポロジとしてもつ並列計算機であるとする。

スケジューラ 今回、スケジューラの実装の位置を設定しないものとする。これに伴い、スケジューラと各プロセスの通信時間、プロセスをプロセッサにロードする時間は考慮しない。

実行されるタスク 起動されるタスクは、プロセスチャートが予め得られているものとする。

プロセス間通信 プロセス間のデータ通信は、プロセスの終了時にのみ、親プロセスから子プロセスに向けて起こるものとする。この時の通信の時間は、プロセッサ間の距離に比例して設定した。データの量については考慮していない。

個体及び目的関数 個体、目的関数の設定に関しては、5 章で説明する。

個体の適応度 個体 i の適応度 F_iは、以下の式で表されるものとする。

$$F_i = Um + M - f_i$$

但し、$U = 0.2$ で、m, Mは、それぞれその世代における目的関数の最小値、最大値を表す。また、f_iは、個体 i の目的関数値である。

淘汰 淘汰は、適応度が平均値を下回っている個体全てに対して行なわれるものとする。

交叉 交叉については、個体に対して確率 Pc で行なわれる。その方法については、5 章で説明する。

突然変異 突然変異については、個体に対して確率 Pm で起こすものとする。個体を乱数で新たに設定し直す方法をとる。

5 シミュレーション結果

以下にシミュレーション結果を示す。

5.1 静的スケジューリング

このモードでは、N個のプロセスをM個のプロセッサに割り当てようとする時、個体を図3の様なM行N列の行列で表現する。

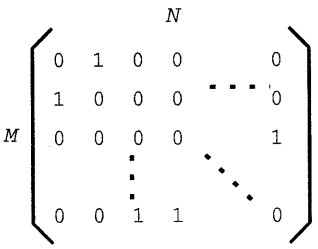

図 3: 個体行列

プロセスnが、プロセッサmに割り当てられているとすると、行列のm行n列の値が1となる。

目的関数は、プロセス0からプロセス$N-1$まで実行した時の時間で表されるとする。プロセスの動作時間と、プロセスが終了した時に子プロセスに通信する時間を考慮する。通信に必要な時間Tcは、以下の式で表されるとする。

$$Tc = Pl \times C$$

ここで、Plはプロセッサ間の距離、Cは単位距離あたりの通信時間を表す。

交叉方法は、多点交叉であり、図4のように2つの個体間でランダムに切れ目を入れて交叉を行なう。

以下に、図5、図6の様なプロセスチャートをもつタスクについて、プロセッサ数4×4、個体数100、世代数5,000、交叉確率$Pc = 0.5$、突然変異確率$Pm = 0.2$、単位距離あたりの通信時間$C = 3$でシミュレーションを行なった。その結果をそれぞれ図7、図8に、ランダム法での結果とあわせて示した。ランダム法での結果の出力方法は、GAと比較するためにある世代gにおいて、

$$g \times (GA\text{の個体数})$$

で表される数のデータを発生させてその中から最も目的関数の小さいものを準最適解としてプロットした。例えば、世代数5,000の時のランダム法での値は、500,000個のデータを発生させた時の準最適解である。

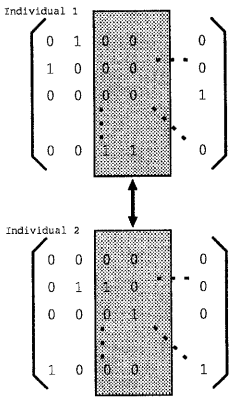

図 4: 交叉の一例

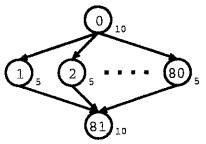

図 5: プロセスチャート 1

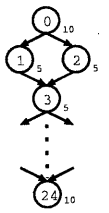

図 6: プロセスチャート 2

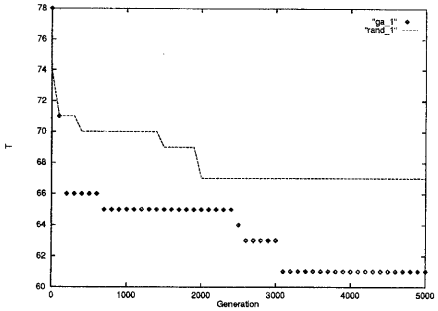

図 7: シミュレーション結果 1

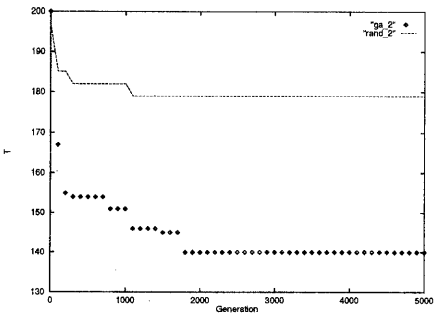

図 8: シミュレーション結果 2

以上より、GA 法による静的プロセススケジューリングは、ランダム法に比べてかなり高い準最適解探索能力がみられるのがわかる。

5.2 動的スケジューリング

このモードでは、個体を図 9の様なベクトルで表現する。ベクトルの長さはプロセッサの総数 Mに等しい。あるプロセスがプロセッサ m に割り当てられているとすると、個体ベクトルの m に対応するする値が 1 となる。そして、この個体ベクトルに対してタスク番号 Tn とプロセス番号 Pn を管理するプロセス管理テーブル (図 10) を個体数分用意する。

	0 1 0 1 0 1 1 1 ▪ ▪ ▪ 1 1 1
Tn	- 1 - 4 - 2 ▪ ▪ ▪ 2 5 1
Pn	- 3 - 0 - 1 2 ▪ ▪ ▪ 2 1 3

図 10: プロセス管理テーブル

個体 i の目的関数の値は、以下の式で表される。

$$f_i = \sum_{m=0}^{M-1} (Bm + Pm + Lm)$$

但し、Bm、Pm、Lm は以下の様に定義されるものとする。

Bm プロセッサ m が動作中でさらにスケジューリング中のプロセスが割り当てられている時、値 100 をとり、それ以外の時は 0 である。

Pm プロセッサ m にスケジューリング中のプロセスが割り当てられている時、値 100 をとり、それ以外の時は 0 である。

Lm プロセッサ m にスケジューリング中のプロセスが割り当てられていて、親プロセスとの距離が lであるとき、値 $l \times C$をとり、それ以外の時は 0 である。

交叉の方法は、個体内で図 11の様にランダムで行なう。その時、交叉に関わった位置に対応するプロセス管理テーブルも変更する。

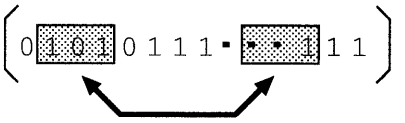

図 11: 交叉の例

ここで、図 12の様なプロセスチャートを持つタスクを、同時に 6 つ起動したとして、プロセッ

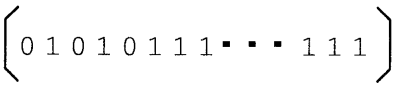

図 9: 個体ベクトル

サ数4×4、個体数100、交叉確率 $Pc = 0.5$、突然変異確率 $Pm = 0.2$、単位距離あたりの通信時間 $C = 3$ で、シミュレーションを行なった。図13にGA法の結果、図14にランダム法の結果を示す。

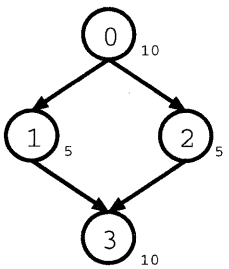

図 12: プロセスチャート

図13、図14から、GA法の方がランダム法と比較して無駄なくスケジューリングされているのがわかる。

6 おわりに

前章でのシミュレーションの結果より、GA法を用いたプロセススケジューリングは、単純なランダム法を用いたものと比較して、高い準最適解探索能力があることが確認された。しかし、並列計算機向きOSのスケジューリングアルゴリズムとして実装されるまでには、まだ多くの障害を越えなければならない。例えば、スケジューリングそのものにかかる時間の問題はその1つであり、現実には大変重要な問題である。

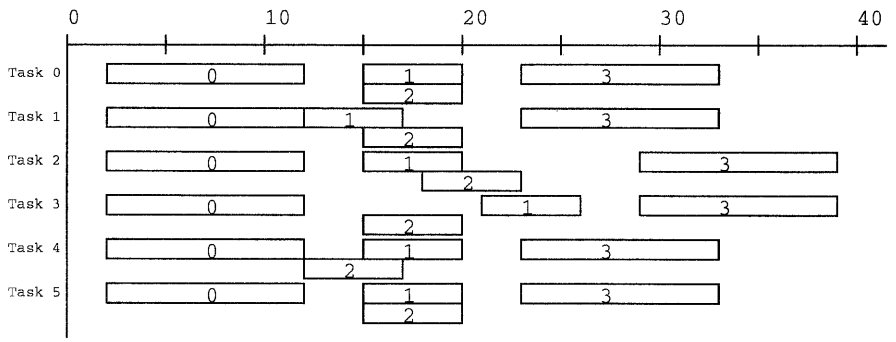

図 13: シミュレーション結果 (GA法)

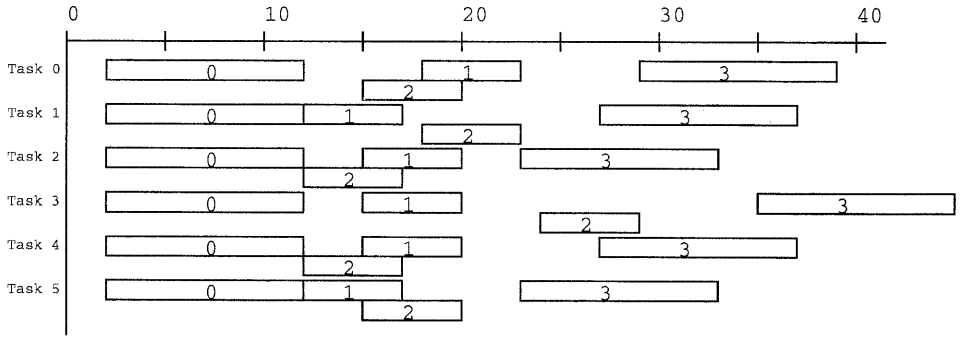

図 14: シミュレーション結果 (ランダム法)

今後は、GA の高速化のためのチューニングを行なう予定である。

そこで、GA の高速化にあたって重要な事柄をいくつか挙げておく。

個体のコード化法 GA は個体に対して演算を行なうものなので、個体の表記方法の工夫はスケジューリング時間を顕著に短縮させると考えられる。

目的関数 目的関数の計算式が簡単であれば、スケジューリング時間も短縮される。

交叉方法 交叉は、GA の演算の中で最も重要であるので、交叉方法としてどんな方法をとるのかが、スケジューリング時間に影響を与える。

本稿では、OS のスケジューリングアルゴリズムに GA を適用し、2 つのスケジューリングモードを提案した。その 2 つのモードについてシミュレーションによるデータ収集を行ない、評価を行なった。先にも記したが、GA そのものはランダム法ベースの非常に簡単なアルゴリズムである。一方、そのチューニング方法は逆に非常に難しいものである。

将来の並列計算機向き OS にこのアルゴリズムが実用的に実装されるかどうかは不確定だが、GA の発展型のアルゴリズムには将来性がある。今後は、高速かつ効率的に収束するチューニング方法やアルゴリズムを研究していく予定である。

謝辞

本研究の一部は、文部省科学研究費補助金 (重点領域研究 (1)；04235103) による。

参考文献

[1] 三宮:遺伝アルゴリズムによる最適化問題の解法, システム制御情報学会, 研究発表講演会講演論文集, pp.9-18,1992.

[2] 奥川:並列計算機アーキテクチャ, コロナ社,1991.

[3] 西川, 玉置:ジョブショップ・スケジューリング問題に対する遺伝アルゴリズムの一構成法, 計測自動制御学会論文集,Vol.27,No.5, pp.593-599,1991.

[4] 中済:並列化 GA を用いたプロセススケジューリング, 第 4 回並列人工知能研究会誌, pp.1-4,1994.