実用的な近似解を与えるプログラム分割アルゴリズム

齊藤哲哉＊　高田雅美†　城和貴†　國枝義敏＊　福田晃‡
＊ 和歌山大学 システム工学部
† 奈良女子大学 理学部
‡ 奈良先端科学技術大学院大学 情報科学研究科

概要

本論文では，Girkarのプログラム分割アルゴリズムに対してヒューリスティックスを導入し，最適な解が得られる保証はないが，実用的に問題のない程度の近似解が得られることを示す．我々が提案するヒューリスティックスは，対象となるタスク・グラフの枝のリストの並びに注目し，ある枝の位置を基点とし，そこから選択した枝までの距離の平均値を切り上げた値と最大値を評価値として利用する．この評価値によって，いくつ先まで分枝操作をするかを決定し，得られた暫定値を下限値として限定操作を行う．実験の結果，十分に実用的な近似解が得られることがわかった．

A Program Partitioning Algorithm for Practical Approximate Solutions

Tetsuya SAITO＊　Masami TAKATA†　Kazuki JOE†　Yoshitoshi KUNIEDA＊　Akira FUKUDA‡
＊ Faculty of Systems Engineering, Wakayama University
† Faculty of Science, Nara Women's University
‡ Graduate School of Information Science, Nara Institute of Science and Technology

Abstract

In this paper we propose an heuristics, without guarantees for the optimal solution, to obtain practical approximate solutions for the Girkar's branch-and-bound based program partitioning algorithm. Our heuristics, which focuses on a sequence of edges in a task graph, evaluates the maximum and average value of distances between edges. The values are used as an incumbent value for branching. Experimental results show our heuristics has an ability to obtain practical approximate solutions for the algorithm.

1　はじめに

自動並列化コンパイラは，与えられた逐次プログラムを詳細に解析し，各種の最適化手法を適用して，対象となる並列計算機上で効率よく実行できるように変換するコンパイラである．アプリケーション開発者は，自動並列化コンパイラを利用することで，過去の資産を利用できるだけでなく，従来のプログラミング言語やアルゴリズム，デバッグツールを利用して開発を進めることができる．

我々は，分散メモリ型並列計算機を対象とする自動並列化コンパイラNarafraseの開発を行っている．Narafraseでは，その核となる中間表現として，タスクの制御依存，データ依存，ループのネスト・レベル等を表現したタスク・グラフに，各タスクがアクセスを行う変数情報も付加したDPG（Data Partitioning Graph）[2]を採用している．Narafraseの対象とするアーキテクチャは分散メモリ型であるため，プログラムの分割とデータの分割を同一のフレームワークで行う必要がある．DPGはそのフレームワークとしての機能を十分に供給できるため，中間表現として採用を決定した．また，プログラムの分割とデータの分割を同時に最適化するアルゴリズムであるCDPPアルゴリズム[3]もDPG上でのNarafraseの中心的な最適化手法として採用する予定である．

CDPPアルゴリズムは，Girkarが提案した分枝限定法を利用したプログラム分割アルゴリズム[1]の自然な拡張であるが，Girkarのアルゴリズムでは，有効な下限が示されていないため，枝の本数が増えると計算量が増大し，実際の自動並列化コンパイラでの利用は困難であるという問題があった．そのため，この問題点を解決しなければ，CDPPアルゴリズムのNarafraseへの適用も困難である．

そこで本論文では，Girkarのプログラム分割アルゴリズムに対して，ヒューリスティックスを導入し，最適な分割が得られる保証はないが，実用的に問題のない程度の近似解が得られることを示す．分散メモリ型並列計算機では，通信競合等の動的なオーバーヘッドが発生するため，通信に費やされる正確な時間を見積もることは極めて困難である．従って，通信時間もパラメータの一つとして利用するGirkarのプログラム分割アルゴリズムで，正確な最適解を求めたとしても，それが実際の最適な並列プログラムの実行に直結するとは限らない．このような背景から本論文では，プログラム分割では必ずしも最適解を求める必要はなく，分割に必要とされる計算時間とのトレードオフを考えて，その近似解でも積極的に採用することとした．

2　Girkarのプログラム分割アルゴリズム

Girkarのプログラム分割アルゴリズムは，分枝限定法を用いて最適なプログラム分割を見つけることができる．以下では，そのアルゴリズムの概略について説明する．

プログラムは一般に，閉路を持たない重み付き有向グラフ$G=(N,E)$で表すことができる．節点$n\in N$はプログラムのステートメントに相当し，節点のコスト$t(n)$は実行時間を表す．枝$e=(n_i,n_j)\in E$は節点n_iから節点n_jへの通信を表し，枝のコスト$c(e)$は節点n_iと節点n_jが別々のプロセッサに割り当てられた場合に，通信に必要なコストを示してい

る．また，枝 e は依存関係を表し，節点 n_i の実行が終了するまで節点 n_j の実行を開始できないことを示している．

Girkar のプログラム分割アルゴリズムは，枝を unexamined な状態から inter，intra という 2 つの状態に分ける分枝操作を繰り返し行い，部分問題を生成していく．各部分問題は，3 つの組 $< G, A, a >$ を保持している．G はその時点での分割状態を示すグラフであり，A は unexamined な枝の集合，a は G のクリティカルパス長である．暫定解は部分問題の中で最小の a を持つものであり，その部分問題が持つ A，すなわち unexamined な枝の集合が空になればアルゴリズムは停止し，G は最適なプログラム分割を示すグラフとなる．

アルゴリズムの開始時には，初期暫定解として，あらかじめ元のグラフ G のすべての枝を unexamined とし，暫定値として節点の最大コストを与えたものを部分問題のリストに入れておく．続いて，ある探索法を利用して unexamined な枝の集合から任意の枝 $e = (n_i, n_j)$ を選択し，始点 n_i と終点 n_j を別々のプロセッサに割り当てる場合（inter）と，同じプロセッサに割り当てる場合（intra）の 2 通りの場合についてそれぞれ部分問題を生成する．このとき，枝を intra とした場合には，節点 n_i から節点 n_j まで到達できるすべてのパスに含まれる枝の中に inter とした枝が存在するとデッドロックを引き起こすため，この部分問題はリストに追加しない．

Girkar のプログラム分割アルゴリズムにおける分枝限定法の手順をまとめると以下のようになる．

1. 部分問題のリストから暫定解を取り出す．
2. unexamined な枝があるかを調べる．すべての枝が inter もしくは intra に分けられていればアルゴリズムは終了する．このときの構成に含まれるグラフ G が最適な分割を表している．
3. unexamined な枝 $e = (n_i, n_j)$ を 1 つ選択する．
4. 選択した枝 e を inter としてクラス分けした場合の構成を求め，部分問題のリストに加える．
5. 選択した枝 e を intra としてクラス分けした場合の構成を求め，部分問題のリストに加える．ただし，3 で選択した枝の始点 n_i から終点 n_j へ至るすべてのパスについて，inter としてクラス分けされた枝が存在する場合には，部分問題のリストに追加しない．
6. 1 に戻る

3 提案するヒューリステックス

我々はすでに，Girkar のプログラム分割アルゴリズムにおいて，ある時点での暫定解に対して 5 つ先まで分枝操作を実行し，その結果として求められた新たな暫定解が持つ暫定値を，限定操作におけるヒューリスティックスとして利用することを試みている [4]．この試みは，ランダムに生成されたタスク・グラフに対して極めて有効に作用したが，何故 5 つ先までの分枝操作が有効なのかを説明するに至っていなかった．そこで，実際のプログラムからタスク・グラフを生成し，プログラムとして意味のある依存関係（タスク・グラフの枝）を分析することで，節点に対する入力枝や出力枝，依存関係のある節点間の距離（枝の長さ）などがヒューリスティックのための評価値に深く関係することが判明した．

本節では，Girkar のプログラム分割アルゴリズムにおいて，ある暫定解に対して限定操作を行う際に利用するヒューリスティックスを得るために，あらかじめいくつ先まで分枝操作をするかを決めるパターンをいくつか提案する．

我々が提案するヒューリスティックスは，対象となるグラフの枝のリストの並びに注目し，ある枝の位置を基点とし，そこから選択した枝までの距離の平均値を切り上げた値と最大値を評価値として利用する．この評価値によって，いくつ先まで分枝操作をするかを決定する．

パターン 1，2 基点を，入力枝が複数存在する節点への入力枝すべてとし，距離を，基点から基点として選んだ枝と同じ節点への入力枝までとする．評価値は，基点以降にある入力枝に対する距離の平均値（パターン 1）と最大値（パターン 2）とする．

パターン 3，4 基点を，入力枝が複数存在する節点への入力枝でリストの先頭に最も近い枝とし，距離を，基点として選んだ枝と同じ節点への入力枝で，基点から最も遠い入力枝までとする．評価値は，基点以降にある入力枝に対する距離の平均値（パターン 3）と最大値（パターン 4）とする．

パターン 5，6 基点を，入出力枝が複数存在する節点への入力枝すべてとし，距離を，基点から基点として選んだ枝と同じ節点への入出力枝までとする．評価値は，基点以降にある入出力枝に対する距離の平均値（パターン 5）と最大値（パターン 6）とする．

パターン 7，8 基点を，入出力枝が複数存在する節点への入出力枝でリストの先頭に最も近い枝とし，距離を，基点として選んだ枝と同じ節点への入出力枝で，基点から最も遠い入出力枝までとする．評価値は，基点以降にある入出力枝に対する距離の平均値（パターン 7）と最大値（パターン 8）とする．

図 1 のようなグラフを対象とし，ヒューリスティックスの各パターンを計算する例を以下に挙げる．丸が節点を表し，矢印が枝を表す．それぞれに付けられている数字は識別するための番号である．また，ヒューリスティックスに利用する枝のリストはこの番号順に並んでいるものとする．

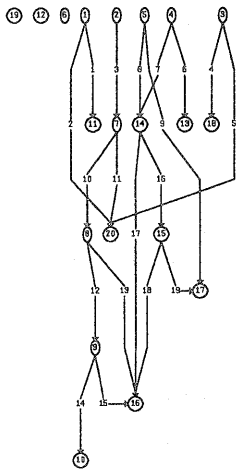

図 1: タスク・グラフの例

パターン **1** $(3+6+1+10+2+2+1)/7=4$

パターン **2** 10

パターン **3** $(9+1+10+5)/4=7$

パターン **4** 10

パターン **5** $(1+3+1+7+6+1+1+1+10+1+2+1+2+2+1+2+1+2+1+1)/20=3$

パターン **6** 10

パターン **7** $(1+9+8+1+1+10+1+10+3+3+5+3)/12=5$

パターン **8** 10

4 ヒューリスティックスの評価

我々の提案するヒューリスティックスの評価をするために，タスグ・グラフを100個ランダムに生成して実験を行った．ただし，実際のプログラムから得られるタスグ・グラフに近い形にするために，節点から出る枝を最大2本に制限し，節点数20，枝数19のタスク・グラフを生成した．節点や枝につくコストは1000までの整数をランダムに付加している．

表1はそれぞれ各ヒューリスティックスのパターンでプログラム分割を求めるのにかかった計算時間（システム時間）を表している．ここで平均時間に注目してみると，特に速度向上が見られるのはパターン1，パターン5，パターン7である．ヒューリスティックスとして平均値を利用するグループ（パターン1，3，5，7）は最大値を利用するグループ（パターン2，4，6，8）より先読みの探索空間が狭いため，ヒューリスティックスを導入したことによるオーバーヘッドが軽減されていると考えられるが，それらの平均時間がオリジナルと比べて短縮されているということから，浅い先読みながら，分枝操作が十分に効率が良く働いていることがわかる．

表1: 計算時間に関するデータ（単位：秒）

オリジナル	0.050	325.720	9.970	34.873
パターン	最小	最大	平均	偏差
1	0.020	17.970	0.804	2.160
2	0.030	132.550	8.332	18.394
3	0.040	75.360	4.769	11.683
4	0.040	233.410	15.808	35.461
5	0.030	7.630	0.384	0.890
6	0.030	142.330	10.582	23.125
7	0.020	30.760	1.830	4.580
8	0.040	235.790	18.888	40.574

表2は，それぞれ各ヒューリスティックスのパターンで求められたプログラム分割におけるクリティカルパス長とGirkarのアルゴリズムで求めた最適解におけるクリティカルパス長との誤差に関するデータを示している．どのパターンの場合も，誤差の平均は多くてパターン5の10%で，それ以外は数%以下という結果になっており，我々が提案しているヒューリスティックスが実用的な近似解を与えていることがわかる．また，最大値を利用するヒューリスティックスのパターン（パターン1，3，5，7）の方が探索空間が広いため，より誤差が少なくなっているが，前述の計算時間の結果と合わせて考えると，ヒューリスティックスの導入によるオーバーヘッドが限定操作の効率を落としていることもわかる．

```
 1: a = 10;
 2: b = 20;
 3: c = 30;
 4: d = 40;
 5: e = 50;
 6: printf("Some output\n");
 7: f = b + 1;
 8: g = f + 1;
 9: h = g + 1;
10: printf("h = %d\n", h);
11: printf("a = %d\n", a);
12: printf("Some output\n");
13: printf("d = %d\n", d);
14: i = d + e;
15: j = i + 1;
16: printf("g+h+i+j = %d\n", g+h+i+j);
17: printf("e+j = %d\n", e+j);
19: printf("Some output\n");
20: printf("a+c+f = %d\n", a+c+f);
```

図2: タスク・グラフに対応するプログラム

表2: 誤差に関するデータ（単位：%）

パターン	最小	最大	平均	偏差
1	0.000	45.960	6.966	10.527
2	0.000	32.720	1.854	5.033
3	0.000	32.720	2.905	6.015
4	0.000	15.300	0.482	2.288
5	0.000	64.003	11.710	14.054
6	0.000	32.720	1.033	4.044
7	0.000	38.532	4.571	7.879
8	0.000	15.300	0.153	1.530

これらの結果から，8つのパターンのうち，計算時間が速く，誤差が少ないパターン1が実用的な近似解を与えるヒューリスティックスであると言える．

5 考察

タスク・スケジューリングやプログラムのタスクへの分割を目的としたアルゴリズム研究では，ランダムに生成されたタスク・グラフを利用してアルゴリズムの検証を行うことが多い．しかしながら，実際のプログラムから生成されるタスク・グラフは，もとのプログラムの持つ特性を反映して作られる物であり，ランダムに生成されたプログラムが意味を持たないのと同様に，ランダムに生成されたタスク・グラフも意味を持たないはずである．本論文で提案しているヒューリスティックスは，並列処理を前提としたタスク・グラフの持つ意味を，データの依存関係という観点から考慮し，決められたものである．本章では提案したヒューリスティックスの持つ，実際のプログラムにおける意味に関して考察する．

図1のタスク・グラフを持つ実際のプログラムの例を図5に示す．

ただし，プログラム中の各ステートメントの先頭にある番号は，節点の番号を表す．このプログラムは，実際のプログラムとしての意味はないが，通常のプログラムの形式には合致している．

枝2の枝は，節点1から節点20に依存関係があることを示している．これを上記プログラムで確認すると，文番号1（節点1）では変数aの値が代入（書込）されており，文番号20（節点20）でその変数の値が出力（変数の読込）されている．従って，節点1と節点20は依存関係にあり，節点1の実行が終わらなければ，節点20の実行を始めることはできない．同様に，枝5の枝は節点3と節点20の依存関係を示している．

ここで注目すべきことは，同一の節点20に入力枝として入る枝2の枝と枝番号5の枝は，節点1ならびに節点3と節点20の依存関係を形成していることである．直観的に考えれば，節点1，3，20は同一プロセッサに割り当てられるように当該プログラムは分割されるべきである．これを本論文の目的であるGirkarのプログラム分割の近似解を与えるヒューリスティックスという観点から見直すと，枝2の枝と枝5の枝の限定操作を行う際に，枝2枝の限定操作は，枝5の枝までの分枝操作が行われた後，すなわち3つ先の分枝操作が終わってから行うべきである．この時，他の強い制約条件がない限り，節点1，3，20は同一プロセッサに割り当てられることになる．

このような着眼点から我々は，複数の入出力枝を持つ節点に注目することで，プログラムの特性，すなわちタスク間の依存関係を考慮したヒューリスティックスが得られるものと考えた．実際，節点に複数の入出力枝が存在する場合，その節点は他の節点と依存関係にあるため，関連する節点はなるべく同一プロセッサに割り当てられるような限定操作が行われるべきである．

以上の議論を踏まえて，本論文で提案したパターン1〜8のヒューリスティックスの意味を考察する．まず，パターン1および2では，複数の入力枝を持つ節点に注目している．これは，複数の節点（ステートメント）で書き込まれた変数を，当該節点（ステートメント）で読み込むことを意味している．複数の入力枝には，分枝限定操作を行う順番を決めるための番号付がなされているが，そのような各枝の距離（番号の差）を求め，その平均数だけ分枝操作を行うのがパターン1であり，最大数分，分枝操作を行うのがパターン2である．明らかにパターン2の方が評価値は大きく，プログラム分割をするのに計算時間が長くなるが，最適解に近い解を求めるには有効な方法である．

パターン3および4では，同じく複数の入力枝を持つ節点に注目するが，複数入力枝のうち，距離が最大となるようなものを算出し，その平均数だけ分枝操作を行うのがパターン3であり，最大数分，分枝操作を行うのがパターン4である．この場合，パターン1よりパターン3の方が必然的に評価値が大きくなるが，パターン2やパターン4程ではない．

パターン5〜8では，複数の入出力枝を持つ節点に注目している点が，パターン1〜4と異なる．特にパターン5では，最も小さな評価値を出す傾向があり，そのため実験でも示したように，計算時間自体は一番速い．しかしながら，最適解との誤差が最大であるという問題以外にも，パターン5を利用すべきではない理由がある．

複数の入力枝を持つ節点は，先行する複数の節点で変数の書き込みが全て終わらなければ当該変数の読み込を行ってはいけない．従って，これに関連する節点は同一プロセッサに配置された方が都合が良い．一方，複数の出力枝を持つ節点は，その節点での変数書き込みが終了しさえすれば，後続する複数の節点は，任意順序で実行可能である．すなわち，通信時間に関する制限が強くない限り，関連する節点を全て同一プロセッサに割り当てる必要はない．

以上の理由により，前章で示した結果，すなわちヒューリスティックスとしてパターン1が最適であるというのは，プログラムの意味を考えても妥当性がある．

なお，[4]にてランダム・タスク・グラフに対するプログラム分割の評価値を5としたのは，本考察で言い換えれば，実験対象のタスク・グラフに対するパターン1もしくはパターン5の評価値の平均値が5であったことに他ならない．

6 結論

我々が提案するヒューリスティックスは，対象となるタスク・グラフの枝のリストの並びに注目し，ある枝の位置を基点とし，そこから選択した枝までの距離の平均値を切り上げた値と最大値を評価値として利用する．本論文では，Girkarのプログラム分割アルゴリズムに対して，我々が提案するヒューリスティックスを導入し，最適な解が得られる保証はないが，実用的に問題のない程度の近似解が得られることを示した．

今後は，実際のプログラムからタスク・グラフを生成し，我々の提案したヒューリスティックスが，実際のプログラムに対してもに実用的な近似解を与えるかどうかを検証していく必要がある．また，ヒューリスティックスを適用するために利用する枝のリストの順序がヒューリスティックスの評価値に大きな影響を与えるので，並び順がどのように影響し，どう評価値と結びついているのかを調べる必要がある．今回はヒューリスティックスの評価値を静的に決定し，それを最後まで適用し続けるという方針を取ったが，各限定操作時に新たな評価値を動的に決定することも考えている．

謝辞

本研究の一部は日本学術振興会未来開拓学術研究推進事業（知能情報・高度情報処理分野）JSPS-RFTF96P00505の援助による．また，Girkarのプログラム分割アルゴリズムに関して，詳細な情報と有意義なコメントを頂いたPolychronopoulos教授に深く感謝する．

参考文献

[1] Girkar, M. and Polychronopoulos, C.: Partitioning Programs for Parallel Execution, *Proceedings of International Conference on Supercomputing*, pp. 216–229 (1988).

[2] Nakanishi, T., Joe, K., Polychronopoulos, C. D., Fukuda, A. and Araki, K.: The Data Partitioning Graph: Extending Data and Control Dependencies for Data Partitioning, *Lecture Notes of Computer Science*, No. 892, Springer, pp. 170–185 (1994).

[3] Nakanishi, T., Joe, K., Saito, H., Fukuda, A. and Araki, K.: CDPP Algorithm: Combined Data and Program Partitioning, *Proceedings of International Conference on Parallel Processing*, Vol. II, pp. 177–181 (1995).

[4] 高田雅美, 齊藤哲哉, 中西恒夫, 城和貴: 分枝限定法を用いたプログラム分割の下界に対する考察, 情報処理学会研究報告 ARC-134-14, Vol. 99, No. 67, pp. 79–84 (1999).