計算機アーキテクチャ　134－14
（１９９９．８．３）

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

高田　雅美 *　齊藤　哲哉 †　中西　恒夫 ‡　城　和貴 *

takata@ics.nara-wu.ac.jp

* 奈良女子大学　理学部
† 和歌山大学　システム工学部
‡ 奈良先端科学技術大学院大学　情報科学研究科

概要

本稿では，非循環であるタスクグラフを対象としたGirkarのプログラム分割アルゴリズムを見直し，その改良として下限値の１つを提案し，実験によって有効性を示した．下限値を用いた探索の制限により多くの場合，探索時間は短くなり，解についても最適解が返された．稀に解が最適とならない場合もあるが，最適解からの誤差は小さいので，解は近似である．また，Girkarのアルゴリズムではメモリ不足となり実行不可能となるタスクグラフについても，実行結果を得ることを可能とした．

Estimating a Lower Bound for a Branch and Bound Based Program Partitioning Algorithm

Masami Takata *　Tetsuya Saito †　Tsuneo Nakanishi ‡　Kazuki Joe *

* Nara Women's University
† Wakayama University
‡ Nara Institute of Science and Technology

Abstract

In this paper, Girkar's program partitioning algorithm for acyclic and task graphs is investigated, and a lower bound as an improvement is proposed to be validated by experimental results. In many cases, the algorithm restricted by the lower bound requires less execution time for search while obtaining optimal solutions. Although there were a few cases that solutions did not become optimal, the solutions were approximate because the errors to optimal solutions were small. The solutions to the task graphs which Girkar's algorithm could not be applied to for lack of memory could be obtained.

1　はじめに

自動並列化コンパイラは，逐次プログラムを入力として受け取り，様々なコード最適化処理を行った後，並列計算機で動作する並列プログラムに自動的に変換する．

分散メモリ型並列計算機を対象として並列プログラムを開発する場合，データの分割配置を最適化にする必要がある．なぜなら，たとえプログラム分割が最適であっても，各プロセッサが要求するデータが他のプロセッサのメモリに存在するようでは，通信オーバヘッドのために全体の実行時間が延びてしまう場合が多い．従って，データとプログラム両方に対して最適な分割をする必要があるためである．

我々は，現在，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として，Narafrase[1]の実装

に取り組んでいる．Narafraseでは，統一的な中間表現として，各変数へのアクセス状況を内包したDPG（Data Partitioning Graph）[2]を採用している．また，我々はCDP^2（Combined Data and Program Partitioning）アルゴリズムをすでに提案している[3]．

CDP^2アルゴリズムとは，Girkarが提案した分枝限定法を用いたプログラム分割アルゴリズム[4]の自然な拡張である．GirkarのアルゴリズムならびにCDP^2アルゴリズムはアルゴリズムレベルでの妥当性を示してはいるが実装上の問題点が残されている．いずれのアルゴリズムも分枝限定法の各ステップにおいて分割最適化後のプログラムの実行時間の下界を算出する．その下界は最終的な実行時間（アルゴリズムが最適解として与える実行時間）とはかなり差があるため枝刈りがあまり利かず，大きな問題に対しては実行時間を要する．この問題を解決するには，最終的な実行時間に近いよい下限値を見つけ，枝刈りを効率化する必要がある．

本稿では，非循環であるタスクグラフを対象としたGirkarのプログラム分割アルゴリズムを見直し，その改良として下限値の1つを提案する．

以下，第2章においてGirkarのプログラム分割アルゴリズムについて述べ，第3章において同アルゴリズムの改善のために下限値の1つを提案し，第4章において改善されたアルゴリズムと元のアルゴリズムとを，実行時間と実行結果を通して比較する．

2 プログラム分割アルゴリズム

本章では，分枝限定法ならびにそれを利用したGirkarのプログラム分割アルゴリズムに関して説明する．

分割対象のプログラムを表すタスクグラフを図1のような有向非循環グラフとする．ただし，枝は通信を，節点はプログラムのステートメントとする．プログラムを分割するとは，いくつかの節点を同一プロセッサに振り分けていくことである．この振り分けの途中段階において各枝は，同一プロセッサ内の枝（IntraPartitionEdge；以後Intra-PEと略記），あるプロセッサから違うプロセッサへの枝（InterPartitionEdge；以後Inter-PEと略記），Intra-PEにもInter-PEにも振り分けられていない枝（UnexaminedEdge；以後U-Eと略記）のどれかに含まれる．U-Eの枝は，分割終了後，Intra-PE又はInter-PEに変更されている．この変更をする際

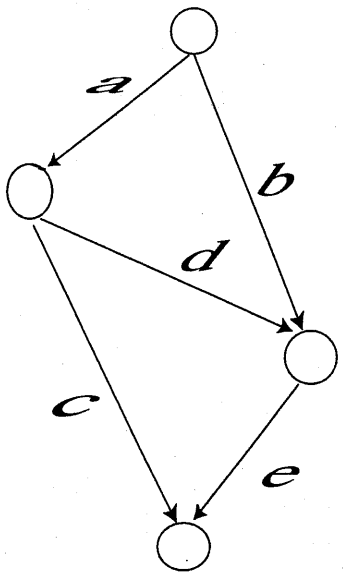

図1: 有向非循環グラフ

の注意点として，U-Eの枝をIntra-PEにする場合Intra-PEの始点n_1から終点n_2について，Inter-PEの枝を含むn_1からn_2へ至る道が存在してはならない．たとえば，図1において枝cをIntra-PEにするのであれば，枝d,eもIntra-PEにしなければならないのである．n_1とn_2が同一プロセッサPに含まれているとき，n_1からn_2へ至るInter-PEの枝を含む道があるならば，プログラム分割後のタスクグラフには循環が生じる．タスクグラフに循環の生じるプログラム分割を行った場合，そのプログラムをデータフローモデルで実行する場合デッドロックが生じるため，上述の制約を設けなければならない．

節点nの実行時間を$t(n)$とし，節点n_1を始点とし節点n_2を終点とする$e =< n_1, n_2 >$の通信時間を$c(e)$とする．枝がIntra-PEの場合，同一プロセッサ内でのデータ移動なので通信時間は$c(e) = 0$となる．

本稿ではパス（path）とはすでに選択されているInter-PEの枝とその枝で結ばれた節点から構成される経道とする．

$path =< n_1, n_2, ..., n_m >$の時の時間を

$$T_\tau = \sum_{i=1}^{m} t(n_i) + \sum_{e=[n_i,n_{i+1}],i=1}^{m-1} c(e)$$

とする．1つの有向非循環グラフに対して path は1本以上存在するので T_r も複数存在する．Inter-PE の枝と Intra-PE の枝との組み合わせで構成された解の評価値は，クリティカルパスの時間，すなわち path の時間 T_r のうち最大のものとする．クリティカルパスの時間はプログラムの（その分割における）最小並列実行時間を与える．

2.1 分枝限定法

タスクグラフの枝を Intra-PE，Inter-PE に割り当てて分割する方法では，枝が n 本存在する場合，2^n 個の解を生成するので最適解を見つけるのは実際的には困難である．そこで，探索時間を短くするために U-E から Intra-PE 又は Inter-PE への変更に制限を与え，総探索数を減少させる，つまり，途中で生成した選択肢節を切り捨てる手法を利用する．この方法を分枝限定法又は分枝制約法という [5]．

例えば最小値を求める場合，すでに得ている暫定解よりも選択肢節の評価値が大きくなった時点でそれ以下の枝を探索する必要はなくなる．

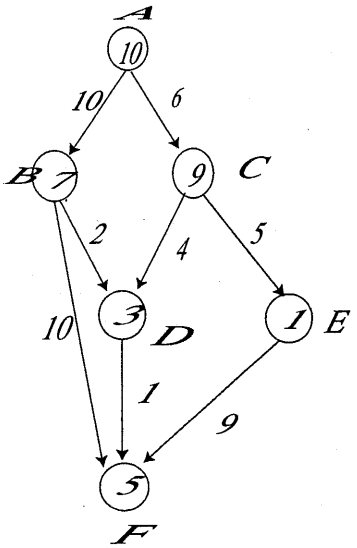

図 2: タスクグラフ

具体例として上界を用いた分枝限定法を挙げる．図2のタスクグラフにおいて各節点の数値を実行時間とし，各枝の数値を通信時間とする．この時，選択肢節の各評価値は path の時間 T_r のうち最大のものとする．まず，全ての枝を Intra-PE にする場合，暫定解は 35 である．これを上界として他の組み合わせについて調べる．最初の選択肢節において枝は全て U-E とすると，評価値は $t(A) = 10$ である．枝 A,B を Inter-PE にする場合，2つ目の選択肢節の評価値は 27，次の選択肢節の値は枝 A,C を Inter-PE にする場合 27，枝 B,D を Inter-PE にする場合 32，枝 B,F を Inter-PE にする場合 42 である．この時点で，選択肢節の評価値が暫定解よりも大きくなるので，これ以下の枝を探索する必要はない．以下同様にして最適解を得る．

分枝限定法を用いれば途中段階で選択肢節の選択を中断することができるので，計算時間が分枝限定法を用いない場合よりも計算時間が少ない．ただし，最悪の場合，分枝限定法を用いない場合と同等の探索時間がかかってしまうので，与えられた問題にあった制限を見つけなければならない．

2.2 Girkar のアルゴリズム

Girkar のアルゴリズムが対象としているのは，図2のような有向非循環タスクグラフである．本節では，このアルゴリズムの説明を行う．

まず，初期状態としてタスクグラフ G に含まれている枝は全て U-E とする．ゆえに，最初の選択肢節における評価値は G の節点の実行時間の最大なものとなる．

1. 末端の選択肢の評価値が最小の構成を検出
2. その構成の中に U-E が含まれているかどうか判定する．
   この時，もし U-E が含まれてなければ探索は終了し，その時の構成より最適分割グラフが得られる．
3. U-E の枝を1つ選ぶ
4. 選んだ枝を Inter-PE にする場合についての新しい構成を生成する次の選択肢節に進む
5. 選んだ枝を Intra-PE にする場合についての新しい構成を生成すし次の選択肢節に進む．
6. 手順1へ戻る．

このアルゴリズムでは，それぞれの選択肢節における最大の T_r が最小の構成を毎回検出して U-E の枝を Inter-PE か Intra-PE のどちらにするかを選択し

ていくことによって，最終的に得られた暫定解よりも，他の全ての選択肢節の評価値は大きいことが保証されている．従って，このアルゴリズムは最適解を与える．

3 Girkarのプログラム分割アルゴリズムに対する新しい下限値

与えられたタスクグラフが複雑な場合，Girkarのアルゴリズムでもメモリの使用量が膨大になり，実行中にメモリ不足となる危険性がある．この問題を解決するために，本章では新しい下限値の1つを提案する．

下限値による分枝限定法とは，節点の暫定解が下限値を超えた場合，それ以下の枝を生成する必要はないという制限をすることである．これは第2章の例として利用した上界を用いた分枝限定法よりも有効である．なぜなら，上界を用いる場合，枝刈りを行う可能性があるのはアルゴリズムの終盤近くである場合が多いのに対して，下限値を用いた場合，枝刈りがアルゴリズムの前半でも行われる可能性があるためである．当然前半で行われる方が，枝刈りの効率としては良い．

数々の実行結果に基づき，我々は下限値として現在注目している節点に対して5つ以上先の子における最小値を下限値として提案する．

本下限値では，下限値による枝刈りが全く起こらずに結果が出力される場合，下限値に対する判断を行う時間分だけ実行が長くなるという問題点がある．しかしながら，このような場合の多くは，与えられたタスクグラフがもともと複雑ではなく，本下限値を使わなくても使用されるメモリはさほど多くない．

4 実験

Girkarのアルゴリズム（original）と我々が第3章で提案した下限値を用いたアルゴリズム（改良版）との実行結果を表1に表す．

節点数と枝数を指定すると指定範囲内で節点間に枝をランダムに張り，節点の実行時間と枝の通信時間を0～1000までの小数点を含むランダム数とするプログラムによって生成されたタスクグラフを使用する．本実験において，節点数の指定は10, 20, 30，枝数の指定は節点数の半数から節点数までとし，30個のタスクグラフに対する結果を比較した．

実験環境は，Sparc SUN Workstation Ultra-2（SunOS 5.6），メモリ容量512MBである．

この実験結果のうちSpeed up率が100％より少ない場合，おそらく下限値による枝刈りが生じていないものと思われる．また仮に生じていたとしても，下限値探索に伴う実行時間の増加よりも短時間分しか探索に要する時間を削減できなかったためにSpeed up率が少なくなったものと思われる．

最適解との誤差は，ほぼ0％である．0％でない場合は，最適解ではなく近似解を出したこととなる．

実行結果より，新しい下限値の1つは，少量の誤差はあるものの，プログラム分割自体は最適に近い解又は最適解そのものを出力することが判明した．また，実行時間は平均約3分の1である．さらに，早い段階で枝刈りを行うためにアルゴリズムのメモリ使用量が激減し，Girkarのアルゴリズムでは実行結果を得ることが不可能だったもの（20 : 18, 30 : 20, 30 : 22, 30 : 25, 30 : 26, 30 : 29, 30 : 30）の結果も改良版では可能である．ゆえに，この下限値は有効である．

5 結論

本稿では，Girkarの分枝限定法を利用したプログラム分割アルゴリズムを見直し，新しい下限値を提案し，実験により，その有効性を示した．特に本手法を利用することで，メモリ使用量が激減し，Girkarのアルゴリズムでは計算できなかったタスクグラフも計算可能となった．

しかし，本稿で提案した下限値を用いても，枝の数が膨大な場合や1つの節点への入出力が多い場合，実行時間は長くなる，あるいはメモリ容量を超えてしまい実行不可能になってしまう．それゆえ，本下限値の改善が今後の研究課題となる．また，本下限値はCDP^2アルゴリズムへの直接利用が期待される．

表 1: 実験結果

節点数：辺数	original(秒)	改良版 (秒)	Speed up 率（%）	最適解との誤差（%）
10:5	0.01	0.01	100	0
10:6	0.01	0.01	100	0
10:7	0.04	0.05	80	0
10:8	0.04	0.04	100	0
10:9	0.11	0.12	92	0
10:10	0.06	0.05	120	0
20:10	0.09	0.08	113	0
20:11	0.63	0.69	91	0
20:12	0.59	0.60	98	0
20:13	3.28	1.02	322	4.3
20:14	0.72	0.42	171	2.7
20:15	4.76	0.63	756	0
20:16	3.05	0.33	924	0
20:17	0.57	0.46	124	0
20:18	-	1.39	-	-
30:15	11.26	2.03	555	0
30:16	1.4	1.25	112	0
30:17	3.77	2.91	130	0
30:18	6.48	4.13	157	0
30:19	75.6	19.05	397	15.3
30:20	-	43.1	-	-
30:22	-	19.34	-	-
30:25	-	41	-	-
30:26	-	15.87	-	-
30:29	-	59.93	-	-
30:30	-	674.32	-	-
平均	-	-	239	1.17

参考文献

[1] Kazuko Kambe, Tsuneo Nakanishi, Kazuki Joe, Yoshitoshi Kunieda, and Fujio Kako, "An Implementation of Loop Transformations witha Universal Intermediate Representation Interface Library" *Proc. of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA '99)* , Vol.IV, pp.1905-1911,Jun. 1999.

[2] Tsuneo Nakanishi, Kazuki Joe, Hideki Saito, Constantine D. Polychronopoulos,Akira Fukuda, and Keijiro Araki, "The Data Partitioning Graph: Extending Data and Control Dependencies for Data Partitioning" *Proc. 7th Int. Workshop on Languages and Compilers for Parallel Computing (LCPC '94)* , pp.170-185, Aug. 1994.

[3] Tsuneo Nakanishi, Kazuki Joe, Hideki Saito, Akira Fukuda, and Keijiro Araki, "The CDP2 Algorithm: A Combined Data and Program Partitioning Algorithm on the Data Partitioning Graph" *Proc. of the 1995 Int. Conf. on Parallel Processing (ICPP '95)* , Vol.II, pp.177-181, Aug. 1995.

[4] Milind Girkar, Constantine Polychronopoulos "Partitioning Programs for Parallel Execution" *Proc. Of the 1988 Int. Conf. on Supercomputing* , pp.216-229, 1988.

[5] A.V.Aho 他 （訳：大野義夫）
情報処理シリーズ *11* データ構造とアルゴリズム，培風館（1987）