分散処理環境における並列処理のための粗粒度プロセス実行順序制御手法*

1P－8

朝倉 宏一†　渡邉 豊英†
名古屋大学大学院 工学研究科 情報工学専攻‡

1 はじめに

近年，複数のワークステーションをローカル・エリア・ネットワークで接続した分散処理環境が普及している．分散処理環境を効率よく動作させるためには，粗粒度プロセス並列処理の適用が不可欠である．我々はプログラム中に記述された関数を単位とした粗粒度プロセス [1] 群を効率よく並列実行させるための，粗粒度プロセス実行順序制御手法を提案する．本提案手法では，それぞれの粗粒度プロセスが実行可能となる条件を実行開始条件として表す．今日までに提案された手法 [2, 3] と比較して，本手法はより最適な実行開始条件を計算する．

2 粗粒度プロセス生成手法

分散処理環境における並列処理を考える場合，計算機間での通信処理を減少させるように粗粒度プロセスを生成することが重要である．すなわち，分散処理環境下ではプロセス間の並列実行性を最大限抽出するよりも，並列実行性を多少犠牲にしてもプロセス間通信の少ないプロセスを生成する方が，全体の処理効率が向上する場合が多い．現在までに報告されている粗粒度プロセス並列化処理手法のほとんどは，基本ブロックを単位とした粗粒度プロセスを生成している [4]．しかし，この手法で生成される粗粒度プロセスはデータ依存関係が大きく，プロセス間通信が多く発生する．したがって，分散処理環境での並列処理において基本ブロックを単位とするプロセス生成法は適していない．

我々は，基本ブロックではなく，プログラム中に記述された関数を単位とする粗粒度プロセス生成手法を既に提案している [1]．関数は独立な変数空間を有しているので，関数をそれぞれ独立な粗粒度プロセスとして生成すれば，プロセス間でのデータ共有が極力抑えられる．また，プロセス間の通信は基本的にプロセスの生成時と終了時に関数の引数と戻り値を授受するときのみに発生する．つまり，関数を単位として粗粒度プロセスを生成することで，プロセス間通信の少ない粗粒度プロセス群が得られる．

3 粗粒度プロセス実行順序制御手法

我々の粗粒度プロセス実行順序手法では，各プロセスに対してプログラムの実行中にどのような条件が成立したら実行を開始してもよいかを，実行開始条件と呼ばれる論理式で表現する．そして，実行開始条件をプログラムの実行時に評価し，実行開始条件が満たされた粗粒度プロセスを動的に生成し実行する [2, 3]．しかし，既存の手法が各基本ブロックを粗粒度プロセスとして生成し，並列実行することを目的としているのに対して，我々のアルゴリズムは関数単位の粗粒度プロセスが生成されることを前提として設計されている．したがって，我々の手法において実行順序制御されるのはプログラム中の関数呼出し文のみである．つまり，関数呼出し文が実行開始可能であるか否かが，関数呼出し文に制御流れが到達するまでの基本ブロックの実行結果，すなわちどのブロックからどのブロックへ制御が移動したかの情報により表される．

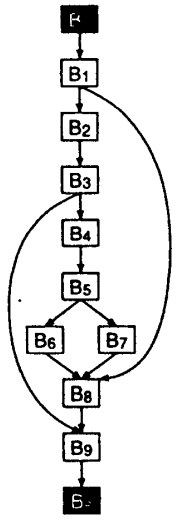
3.1 実行順序制御アルゴリズム

我々の粗粒度プロセス実行順序制御アルゴリズムは，基本ブロック，および関数呼出し文のみから成る関数呼出し文ブロックの二種類のブロックから構成される制御流れグラフに対して適用される．まず，並列実行の対象となっている関数呼出し文ブロックが後方支配 (post-dominate)[2] しているブロックの集合 V_{pd} のみで構成される制御流れサブグラフを生成する．後方支配関係の定義より，プログラムの制御が集合 V_{pd} 中のブロックに到達したとき，関数呼出し文ブロックに制御が到達することが確定する．逆に，集合 V_{pd} 中のどのブロックにも制御が到達しなければ，関数呼出し文ブロックは決して実行されない．したがって，関数呼出し文ブロックの実行開始条件の計算で考慮しなければならないブロック群を，制御流れサブグラフ内のブロックに限定することができる．また，我々の粗粒度プロセス内の基本ブロックは逐次的に実行されるので，集合V_{pd} 内のブロックに制御が到達するとき，それまでに通過したブロックの実行は必ず終了している．したがって，集合 V_{pd} 以前のブロックと関数呼出し文ブロックの間に生じる実行先行制約はすべて満たされ，実行開始条件の計算においてそれらの間の依存

---

*A Control Method of Execution Order for Coarse Grain Distributed Processes

†Koichi ASAKURA and Toyohide WATANABE
‡Department of Information Engineering, Graduate School of Engineering, Nagoya University
{asakura, watanebe}@nuie.nagoya-u.ac.jp



(a) control flow graph

$B_2 \Delta_d B_3$ $B_2 \Delta_d B_4$ $B_2 \Delta_d B_8$

$B_4 \Delta_d B_5$ $B_4 \Delta_d B_8$

$B_5 \Delta_d B_8$

$B_6 \Delta_d B_8$

$B_7 \Delta_d B_8$

(b) data dependency relationship

図 1: プログラム例

関係を考慮する必要がない．制御流れサブグラフを生成したら，サブグラフ中のブロックに対してデータ依存関係を解析し，実行開始条件を計算する．

上で述べたように，我々のアルゴリズムにおける実行開始条件の計算では，その計算範囲を制御流れサブグラフに限定することができる．したがって，従来のアルゴリズムと比較すると，より最適な実行開始条件が得られる．

4 実験結果

図1に示される制御流れグラフとデータ依存関係に対して，ブロック B_8 の実行開始条件を今日までの手法 ([2], [3]), および本手法を用いて計算した結果を表1に示す．また，[2] で提案された最適化処理を施した結果も同様に示す．従来のアルゴリズムと我々のアルゴリズムの計算結果を比較すると，従来手法による計算結果が非常に長いことが分かる．これは，従来のアルゴリズムは制御流れグラフ中の各ブロックを並列に実行することを目的としているからである．これに対して我々のアルゴリズムでは，制御流れグラフ中の各ブロックは逐次実行されることが前提となっているので，データ依存関係の計算をサブグラフ内のブロック ($B_4 \sim B_7$) に限定することができる．したがって，より最適化された実行開始条件が計算される．

また，[2] の最適化された条件と我々の条件では，意味的にはどちらも同じである．しかし，我々の条件式の方が簡潔であり，また論理和の形式で表現されている．したがって，我々の条件式ではどれか一つの条件が成り立てば真となるので，条件式の評価が高速であるという利点がある．

これらの考察により，関数呼出し文を粗粒度プロセスとして生成する場合，[2], [3] のアルゴリズムと比較して我々のアルゴリズムの方がより最適な実行開始条件を与えることが確認された．

表 1: 計算結果

アルゴリズム	B8の実行開始条件
[2], [3]	((B1,B8) ∨(B3,B4)) ∧(B2∨(B1,B8)) ∧ (B4∨(B1,B8) ∨(B3,B9)) ∧ (B5∨(B1,B8) ∨(B3,B9)) ∧ (B6∨(B5,B7) ∨(B1,B8) ∨(B3,B9)) ∧ (B7∨(B5,B6) ∨(B1,B8) ∨(B3,B9))
[2] の最適化法	(B6∨(B5,B7) ∨(B1,B8)) ∧ (B7∨(B5,B6) ∨(B1,B8))
本稿の手法	((B1,B8) ∨(B6,B8) ∨(B7,B9))

Bn … ブロックの実行が終了したら真
(Bi,Bj) … 制御の流れが Bi から Bj に流れたら真

5 おわりに

本稿では，分散処理環境における粗粒度プロセスの実行順序制御手法を提案した．本提案アルゴリズムは関数単位で生成された粗粒度プロセスを前提として設計されているので，従来のアルゴリズムと比較すると，より最適な実行開始条件が得られる．また，我々のアルゴリズムで計算される実行開始条件には論理積が存在せず，すべて論理和の形式で表現されるので，条件式の評価が高速であり，実行時の通信オーバヘッドやプロセス生成オーバヘッドが削減可能である．

参考文献

[1] K. Asakura, T. Watanabe and N. Sugie: "C parallelizing Compiler on Local-network-based Computer Environment", *Proc. of the 7th Int'l Parallel Processing Symp.*, pp. 849–853 (1993).

[2] M. Girkar and C. D. Polychronopoulos: "Automatic Extraction of Functional Parallelism from Ordinary Programs", *IEEE Transaction on Parallel and Distributed System*, Vol. PDS-3, No. 2, pp. 166–178 (1992).

[3] 本多弘樹, 岩田雅彦, 笠原博徳: "Fortran プログラム粗粒度タスク間の並列性検出手法", 電子情報通信学会論文誌, Vol. J73-D-I, No. 12, pp. 951–960 (1990).

[4] 笠原博徳, 合田憲人, 吉田明正, 岡本雅巳, 本多弘樹: "Fortran マクロデータフロー処理のマクロタスク生成手法", 電子情報通信学会論文誌, Vol. J75-D-I, No. 8, pp. 511–525 (1992).