関数呼出し文並列化手法に基づいた粗粒度プロセス生成手法*

3E－4

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，プログラムの自動並列化技術として，プログラム中の基本ブロックを単位とした粗粒度プロセス（マクロタスク）並列化手法が提案されている[1, 2]．基本ブロック間のデータ依存関係やプロセス起動時のオーバヘッド時間などを考慮し，粗粒度プロセス群を生成するアルゴリズムが報告されている[3]．しかし，この基本ブロックを用いた粗粒度プロセスは我々が対象とする分散処理環境における並列処理にはうまく適合しないという問題がある．複数台のワークステーションから構成される分散処理環境では，プロセス起動時やプロセス間通信時のオーバヘッドなどが非常に大きい．上記のアルゴリズムでは一つの基本ブロックが一つの粗粒度プロセスとして生成され，オーバヘッドが非常に大きくなり，したがって分散処理環境ではうまく動作しない．

本稿では，分散処理環境において粗粒度プロセス並列処理を効率よく行うための粗粒度プロセス生成手法を提案する．本提案手法ではプログラム中に記述されている関数が粗粒度プロセスの生成単位となる[4]．我々の粗粒度プロセス生成手法ではプログラム中に記述された関数の独立性が積極的に活用され，関数を単位としてプロセスが生成されるので，並列実行性の高い分散プロセス群が得られる．

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，生成されるプロセスの粒度が非常に大きな問題となる．すなわち，分散処理環境下では適切な粒度のプロセスを実行させ，プロセス間通信を極力なくすことが重要である．現在までに提案されているプロセス生成アルゴリズムでは，プロセス起動時やプロセス間通信時に発生するオーバヘッドが基本ブロックの実行時間と同程度かあるいは小さい計算機環境を前提としている[3]．したがって，このアルゴリズムを我々が対象としている分散処理環境に適用すると，生成されるプロセスの粒度が細かく適切に動作しないという問題が生じる．

我々は既に，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並列処理を提案した[4]．この粗粒度プロセス並列処理では，プログラム中に記述された関数の独立性に注目している．関数間でのデータの受渡しは，基本的に関数が呼び出されるときの引数と，関数での計算が終了したときに送り返される戻り値のみであり，データの転送量が少ない．従来のプロセス生成手法では，プログラム中に記述された関数や副プログラムはインライン展開され，基本ブロックの列として扱われていた．つまり，関数や副プログラムの独立性を積極的に利用してはいなかった．それに対し，我々の手法では関数の独立性に注目することにより，分散処理環境に適した粗粒度プロセスを生成することができる．

3 粗粒度プロセス生成アルゴリズム

上で述べたように，我々の粗粒度プロセスはプログラム中の関数を単位として生成される．しかし，プログラム中に記述されたすべての関数を独立なプロセスとして生成すると，逆に実行時間が増大してしまうという状況が発生する可能性がある．プログラムの実行時間を短縮させるという並列処理の目的を考えると，このような状況は避けなければならない．現在までのプロセス生成アルゴリズムでは，基本ブロックの実行時間と OS のプロセス起動オーバヘッドを比較することで，プロセスを生成している．つまり，OS のオーバヘッドよりも基本ブロックの実行時間の方が長ければ，どのような基本ブロックも分散プロセスとして生成されてしまうという問題がある．しかし，分散プロセスを生成する場合は，実際にそのプロセスを並列実行することにより処理効率が向上するかを評価することが不可欠である．

この目的のために，本アルゴリズムでは呼出し関係にある二つの関数に対して，**依存度** と呼ばれる評価尺度を導入する．依存度は，二つの関数が独立なプロセスとして生成・実行される場合，どの程度並列実行が可能であるかを表す尺度であり，0 から 1 までの値を取る．

---

*A Coarse Grain Process Generation Method Based on Remote Procedure Call Means

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

呼び出す関数が f_i, 呼び出される関数が f_j である
関数呼出し文に対して

ET_i := 関数 f_i の推定処理ステップ数.
ET_j := 関数 f_j の推定処理ステップ数.
DV_{ij} := 関数 f_i, f_j 間の依存度.
短縮実行ステップ数 := min[ET_j, $ET_i \times(1\text{-}DV_{ij})$].
if (短縮実行ステップ数 > オーバヘッド) **then**
　関数 f_i, f_j を粗粒度プロセスとして生成する.
else
　関数 f_i, f_j を融合する.
endif
ET_i を再計算する.

図 1: 粗粒度プロセス生成アルゴリズム (概略)

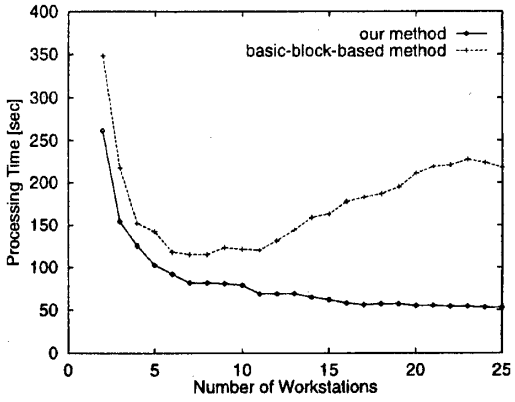

図 2: 実験結果

依存度を計算し，二つの関数の並列実行の有効性が低いと判断された場合，その二つの関数は融合され，一つの粗粒度プロセスとして生成される．すなわち，並列実行により効果が得られると判断された場合のみ分散プロセスが生成される．本アルゴリズムではプロセスの並列実行の有効性が考慮されプロセスが生成されるので，より最適な粗粒度プロセスが得られる．

我々の粗粒度プロセス生成アルゴリズムの概略を図1に示す．プログラム中のすべての関数呼出し文に対してアルゴリズムが適用される．まず，呼び出す側と呼び出される側の関数の実行時間を推定する．実際には，関数内の演算子の個数から実行ステップ数を計算する．次に，データ依存関係解析により二つの関数の並列実行可能性を依存度として計算する．推定実行ステップ数と依存度から，二つの関数を並列に実行させたとき短縮される実行ステップ数を計算する．この値が，実行時のOS によるプロセス起動時のオーバヘッドよりも大きい場合，二つの関数は独立の粗粒度プロセスとして生成され，そうでない場合，二つの関数は融合される．

4 評価実験

我々が提案した粗粒度プロセス生成アルゴリズムにより生成されたプロセス群を，分散処理環境において並列実行させたときの処理時間を図2に示す．横軸が使用したワークステーション台数を，縦軸が処理時間をそれぞれ表している．比較のため，従来の基本ブロック単位の粗粒度プロセス群の処理時間も破線で示す．従来の粗粒度プロセス並列処理では，使用するワークステーションの台数がある程度増加すると処理時間も増大することがわかる．これは，プロセス間通信を頻繁に行う密接に関連するプロセスが異なるワークステーション上で実行されることにより，計算機間の通信量が増大するからである．また，プロセスの粒度が細かいので，プロセス起動時のオーバヘッドが大きくなり，これも処理時間の増大の原因となっている．それに対し，本提案手法で生成された粗粒度プロセス群では，使用するワークステーション台数の増加につれて，処理時間が減少していることがわかる．

5 おわりに

本稿では，分散処理環境において並列処理を達成するための，粗粒度プロセス生成手法について述べた．我々の粗粒度プロセスはプログラム中に記述された関数単位で生成される．また，我々の提案した粗粒度プロセス生成アルゴリズムでは，プロセス間の並列実行可能性を表す，依存度と呼ばれるパラメータを用いてプロセスを生成している．評価実験で得られた結果も，我々の手法が分散処理環境に適していることを示している．

参考文献

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

[2] 本多弘樹, 水野聡, 笠原博徳, 成田誠之助: "OSCAR 上での Fortran プログラム基本ブロックの並列化手法", 電子情報通信学会論文誌, Vol. J73-D-I, No. 9, pp. 756–766 (1990).

[3] 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).

[4] 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).