プログラミング 14－1
（1997. 8. 21）

Doacross ループにおける並列粒度調整方法の検討

高畠 志泰　大澤 範高　本多 弘樹　弓場 敏嗣
電気通信大学大学院 情報システム学研究科

概要

並列プログラムの効率的な実行では、通信時間を考慮に入れ、分割された部分プログラムをどの要素プロセッサ上で実行させるかが重要な問題である。並列粒度調整とは、適切な大きさに部分プログラムを分割することによって実行時間を最も短くすることを目的とする。

本稿では、1重の Doacross ループを並列に実行する場合を考察する。Doacross ループをイタレーション単位で分割する方法、パイプラインで並列化する方法、さらに、その両方の特徴をもつループの新しい並列実行方法を提案し、この 3 つの並列実行方法を比較する。各々の方法の通信回数や実行時間等が、どのように異なるかを LogP モデルを用いて調べる。最後に今後の課題を示す。

Some Consideration on Parallel Granularity Tuning for Doacross Loops

Motoyasu Takabatake　Noritaka Osawa　Hiroki Honda　Toshitsugu Yuba
Graduate School of Information Systems, The University of Electro-Communications

Abstract

It is important to partition a given parallel program into suitable-size subprograms and schedule them for efficient parallel execution. By tuning parallel granularity to characteristics of a parallel computer, execution time of the program can be shorten.

In this paper, a novel parallel execution method for a doacross loop is proposed, which integrates iteration-based parallelizing and software pipelining. The execution time of each method is analyzed by using the LogP model. This method can be adopted as an optimizing technique of parallelizing compilers. Some future problems are also discussed.

1　はじめに

並列プログラムの効率的な実行を行うには、プログラムを並列実行可能な部分プログラム (タスク) に分割する方法と、分割された部分プログラムをどの要素プロセッサ (PE) に割り当てるかというスケジューリング方法が重要である。本稿では、1重の Doacross 型のループ [2] を対象とし、そのループの分割方法と、分割した部分プログラム (タスク) のスケジューリングを考える。本稿における並列粒度とは、並列に実行可能なタスクの実行時間であり、粒度調整とは、タスクの実行時間をループの分割方法によって変えることである。

Doacross ループの並列実行方法には、従来、よく用いられている方法として、イタレーションを 1 つのタスクとして並列に実行する方法

[2][3] がある。一方、別の方法として、ループボディを分割し、ソフトウェアパイプラインを構成して並列に実行する方法 [9] もある。本稿では、上の 2 つの特徴を合わせ持った新たな並列実行方法を提案する。さらに、LogP モデル [1] を使って、この 3 つの並列実行方法の実行時間と通信回数、使用 PE 数を比較し、最も短い時間で実行する方法を選ぶことで並列粒度を調整する。

2 Doacross ループのモデル

2.1 Doacross ループ

本稿で扱う Doacross 型のループ [2] をモデル化する。ループボディ内のタスクの依存関係を有向グラフで表現する。依存グラフにおける節はタスク、枝は依存に対応する。

タスクは、ループボディ内の 1 つの命令、文、ブロックのいずれかに相当する。データ依存関係には、フロー依存、逆依存、出力依存の 3 種類がある。また、ループに関する依存関係には、以下の 2 種類がある。[4]

- ループ運搬依存 (loop carried dependence) : 同一ループ内の 2 つの異なるイタレーション間に存在する依存関係。
- ループ独立依存 (loop independent dependence) : 同一ループ内の 1 つのイタレーション内に存在する依存関係。

ループ運搬依存において、あるタスクが n 個先のイタレーション内のタスクに依存する場合、これをループ依存距離 n という。

ループにおける依存グラフは、ループ運搬依存によって生じる循環部分と非循環部分の 2 つの種類に分けられる。各々の部分においてループ独立依存の枝にそって依存元タスクと依存先タスクを融合しひとつのタスクとし、タスクの粒度を大きくすることにより、Doacross ループの依存グラフは図 1のようにモデル化することができる。各々をループ運搬依存タスクと非ループ運搬依存タスクと呼ぶことにする。

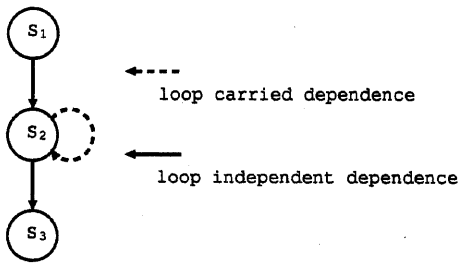

図 1: Doacross loop の依存グラフ

図 1における非ループ運搬依存タスク S_1、S_3 は、2 つの異なるイタレーション間に依存がないので、イタレーション毎に並列実行が可能である。

ループ運搬依存タスク S_2では、複数のループ運搬依存が存在しても構わない。

本稿では、以下の条件に当てはまるループを扱う。

- 1 重の Doacross 型のループである。
- ループ依存距離は基本的に 1 とする。
- ループ運搬依存の数はいくらでもよい。
- 条件分岐が存在しない。

2.2 LogP モデル

LogP モデル [1] は、並列計算機上での並列プログラムの実行時間を評価するためのモデルである。同モデルにおいては、並列計算機の PE 数と PE 間の通信時間を以下の 4 つのパラメータを使って表す。

- L: 1 語のメッセージを送るのに要する時間の遅れの上限 (通信遅延)。
- o: PE においてメッセージの送受信に要する時間 (送受信オーバヘッド)。本稿では、o_sを送信オーバヘッド、o_rを受信オーバヘッドとする。
- g: メッセージを連続して送受信するときの最小時間間隔 (バンド幅)。

- P: PE 数。

3 並列実行の方法

3.1 実行方法の比較

従来から Doacross ループの実行にはよく用いられている方法として、イタレーションを 1 つの単位として PE に割り当てる方法があり、これを従来法と呼ぶことにする (図 2(b))。図では、8 個のイタレーションを 4 台の PE に順番に割り当てている。イタレーションのループ運搬依存を点線の矢印で示す。

従来法は、イタレーションで分割するため、並列に実行させるのが容易である。しかし、ループ運搬依存により実行時間のクリティカルパス上に PE 間通信が現われるので実行時間が長くなる。

ループボディをいつか実行段階に分割し、ソフトウェアでパイプラインを形成し並列化したものをパイプライン法と呼ぶこにする (図 2(c))。パイプライン法では、ループ運搬依存タスクを 1 つのステージとし、それを同一 PE 内で処理することで、ループ運搬依存による通信がなくなり、実行効率がよくなる。また、パイプラインのステージの数を増やすことで並列度を上げることもできる。図 2(c) においては、S_1 を S_{11} と S_{12}、S_3 を S_{31} と S_{32} に分割し、5 台の PE で実行している。しかし、データ依存関係によってパイプラインピッチを揃えるための処理が大変であったり、揃えることができない場合では、最も大きなパイプラインピッチをもつステージによって実行効率が下がる。図 2(c) では、S_{11} が最もパイプラインピッチが長い場合となっている。

ここで、新しい実行方法を提案する。ループ運搬依存タスクと非ループ運搬依存タスクは、異なる PE 上で実行させる。ループ運搬依存タスクより、非ループ運搬依存タスクの方が粒度が大きい場合、異なるイタレーションの非ループ運搬依存タスクを並列に実行する。つまり、ループ運搬依存タスクと非ループ運搬依存タスクの間ではパイプラインを形成し、非ループ運搬依存タスクは、異なるイタレーション間で並列で実行する。この方法を融合法と呼ぶことにする (図 2(d))。

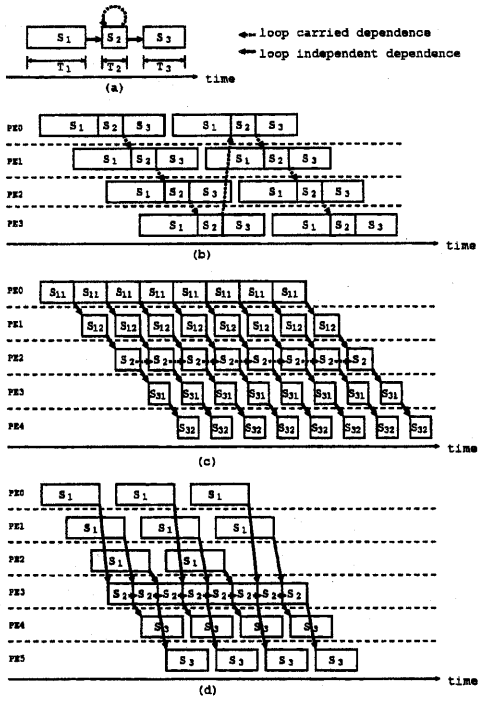

図 2: 並列実行方法

パイプライン法との違いは、非ループ運搬依存タスクを分割してパイプラインで並列に実行するか、異なるイタレーション間で並列に実行するかである。この違いにより、パイプラインピッチを揃える必要がなく、最も大きなパイプラインピッチをもつステージによって実行効率が下がることがなくなる。また、パイプラインによるステージ間の通信が減る。

図 2はループ繰り返し回数 (イタレーション数) を 8 とした場合の実行例である。図 2(a) では、図 1の各タスクの実行時間の長さ (粒度) と依存関係を示している。

表 1: 通信回数の比較

方法	通信回数		
	1イタレーション内	全体	クリティカルパス上
従来法	1	$N-1$	$N-1$
パイプライン法	$S-1$	$N(S-1)$	$S-1$
融合法	2	$2N$	2

3.2 実行時間の比較

Doacross ループにおいて、ループ運搬依存が複数あり、1つのイタレーションで複数のデータが送受信される場合、通信における送受信処理のオーバヘッドを考慮し、1回の通信で送受信すると仮定する。各々の方法におけるループの実行時間を LogP モデルを用いて表す。

ループの繰り返し回数を N、図 2(b) における $S_i(i=1\sim3)$ の実行時間を各々T_iとする。従来法で実行した場合のループの実行時間 T_oは、以下の式で表される。

$$T_o = T_1 + NT_2 + T_3 + (N-1)(o_s + L + o_r) \quad (1)$$

第1、3項は、各々S_1、S_3の実行時間である。第2項は、ループ運搬依存タスクの実行時間である。第4項は、通信にかかる時間である。

パイプライン法で実行した場合を考える。パイプラインのステージ数を S、各ステージの実行時間を $T_i(i=1\sim S)$ とする。パイプライン法でのループの実行時間 T_pは、以下の式で与えられる。

$$T_p = (N-1)(o_s + \max_{i=1,S} T_i + o_r) + \sum_{i=1}^{S} T_i + (S-1)(o_s + L + o_r) \quad (2)$$

第1項は、最もパイプラインピッチの大きいステージを実行する PE において、送受信の通信オーバヘッドを含んだステージの実行時間である。第2項は、1イタレーションを逐次に実行したときの実行時間である。第3項は、ステージ間の通信時間である。

融合法で実行した場合を考える。図 2(d) における $S_i(i=1\sim3)$ の実行時間を各々T_iとする。融合法で実行した場合のループの実行時間 T_hは、以下の式で表される。

$$T_h = (N-1)(o_r + T_2 + o_s) + \sum_{i=1}^{3} T_i + 2(o_s + L + o_r) \quad (3)$$

第1項は、ループ運搬依存タスクの実行時間であり、送受信オーバヘッドを含んでいる。第2項は、1イタレーションを逐次に実行したときの実行時間である。第3項は、通信時間を表している。

式 (1)、(2)、(3) を比較する。従来法では、通信回数がループの繰り返し回数 Nに比例しているため、ループの実行時間が他の方法と比べて長くなる可能性が高い。パイプライン法では、最もパイプラインピッチが大きいステージによって、ループの実行時間が長くなる。そのため、従来法より実行時間が長い場合がある。融合法では、通信回数が2回で、クリティカルパス上で最も時間がかかる部分は、ループ運搬依存が存在する部分なので、他の方法よりも効率がよいことがわかる。パイプライン法において、最もパイプラインピッチが長いステージを S_2、ステージ数 Sを 3 としたときは、融合法と同じ実行時間になる。

実際のプログラムでは、各実行方法でのループの実行時間を調べ、最も短い時間で実行できると思われる方法を選択することで粒度調整を行えばよい。

3.3 通信回数の比較

各方法における通信回数の比較を表1に示す。各々の方法について、1イタレーション内で通信する回数とループ全体での通信回数、クリティカルパス上の通信回数を調べる。1イタレーション内での通信回数は、ループの実行時間における送受信オーバヘッドを考えるためである。全体の通信時間は、ネットワークのトラフィックを考えるためである。クリティカルパス上の通信回数は、通信レイテンシを含む場合を考え、ループの実行時間に大きく影響を与えるので考慮する。

従来法では、1イタレーション内では、1回の受信と送信だけである。しかし、ループを実行する場合、全ての通信がクリティカルパスに現れるために実行には時間がかかる。

パイプライン法では、ステージ数によって通信回数が変わるが、$S > 3$の場合は、他のどの方法よりも通信回数が多くなる。$S = 3$の場合は、融合法と全く同じになる。

融合法は、他のどの方法よりもクリティカルパス上の通信回数が少なくなる可能性が高い。

全体の通信回数が多いと、ネットワークによる遅延が大きくなる可能性があるので、実行時間に影響を及ぼす可能性がある。

3.4 使用PE数の比較

従来法での使用PE数P_oは、以下の式で表される。

$$P_o = \left\lceil \frac{T_1 + o_r + T_2 + o_s + T_3}{o_r + T_2 + o_s + L} \right\rceil \quad (4)$$

この式で求まるPE数は、従来法で実行する場合において無駄な時間を生じない場合のPE数の上限である。求まるPE数より少ないPE数で実行することも可能である。

パイプライン法での使用PE数P_pは、パイプラインのステージ数と等しい。

$$P_p = S \quad (5)$$

この式は、ステージ数Sによって任意のPE数を使うことができることを意味する。

融合法における使用PE数P_hでは、以下の式で与えられる。

$$P_h = \left\lceil \frac{T_1 + o_s}{o_r + T_2 + o_s} \right\rceil + 1 + \left\lceil \frac{o_r + T_3}{o_r + T_2 + o_s} \right\rceil \quad (6)$$

この式で求まるPE数は、融合方で実行するときのPE数で、プログラムによって一意に決まる値である。そのため、他の方法よりもPE数の束縛が強い。第1項は、S_1を実行するためのPE数、第2項は、S_2を実行するためのPE数、第3項は、S_3を実行するためのPE数である。

式(4)、(5)、(6)を比較する。従来法では、割算の分母に通信テイテンシLが入るので、PE数が他の方法よりも少なくなる可能性が高い。パイプライン法では、ステージ数に依存するで、ループボディをいくつに分割するかで変わる。融合法では、パイプライン法を拡張しているので、より多くのPE数を使う。T_1とT_3がT_2以下の場合、パイプライン法のステージ数が3のときと同じになる。

ループ実行時間の比較で求めた実行方法でのPE数が実際の計算機のPE数より多い場合は、よりPE数の少ない実行方法に変えて粒度調整を行えばよい。

4 おわりに

Doacrossループにおいて、イタレーションで分割する従来法とループボディを分割するパイプライン法の両方の特徴をもつ新しい融合法を提案した。また、従来法とパイプライン法、融合法の3つの並列実行方法を実行時間、通信回数、使用PE数の3つの点で比較した。実行時間と使えるPE数によって、実行方法を変えるようにして、粒度調整を行う方法を述べた。

今後の課題としては、以下のものがあげられる。

- ループ依存距離が1より大きい場合について考える。ループ運搬距離が長いと依

存関係のあるタスク間の間にある処理が多くなるので、通信と計算のオーバラップが可能になる。そのためループ運搬依存タスクを並列に実行できる可能性がある。

- 複数のループ運搬依存が存在する場合において、依存のパターンによって最適化を変える。本稿では、1つのタスクとして1つのPEに割り当てたが、並列に実行可能なパターンを探し、並列実行する方法を検討する。
- 複数のループ運搬依存が存在する場合において、各々のループ依存距離が異なる場合について考える。上の2つの複合した場合を考えることが必要である。
- 融合法において、PE数を可変にすることを考える。従来法では、イタレーションで分割しているので、使用するPEを変えることが容易に行える。パイプライン法では、ステージ数を変えることで使用するPE数を変えることができる。融合法でも、使用するPEを変える方法を考える必要がある。
- PE数が足りない場合を考える。現在は、PE数が足りない場合、並列実行方法を変えることによって、使用PE数を変える方針をとっているが、各並列実行方法で任意のPE数を使うことができれば、各々の方法でPE数を決め、そのあと各々のループの実行時間を比較することが可能になる。

参考文献

[1] D. E. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser., E. Santos, R. Subramonian, and T. V. Eicken. LogP: Towards a realistic model of parallel computation. *Proc. the 4th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pp. 1–12, 1993.

[2] R. Cytron. Doacross: Beyond vectorization for multiprocessors. *Proc. of the Int. Conf. on Parallel Processing*, pp. 836–844, 8 1986.

[3] R. Cytron. Limited processor scheduling of doacross loops. *Proc. of Int. Conf. on Parallel Processing*, pp. 226–234, 1987.

[4] H. Zima and B. Chapman. *Supercompilers for Parallel and Vector Computers.* Addison-Wesley Publishing Company, Inc., 1991.

[5] 中西, 城, C. D. Polychronopoulos, 荒木, 福田. ループ最小並列実行時間を算出する一手法. 並列処理シンポジウム JSPP'96, pp. 57–64, 6 1996.

[6] 福田. イタレーション実行時間が異なる1重doacrossループの並列処理における最適プロセッサ数. 信学論 D-I, J75-D-I(7):450–458, 7 1992.

[7] 高畠, 大澤, 弓場, 佐藤, 山口. EM-X用SISALコンパイラにおける並列粒度調整方式. *JSPP'97*, pp. 37–44, 1997.

[8] 山名, 安江, 岡村, 山口. 分散共有メモリ型並列計算機における1重doacross型ループの実行時間算出法. 信学論 D-I, J78-D-I(2):170–178, 2 1995.

[9] 金子, 古関, 小松, 深澤. ループステージング: 共有メモリ型並列計算機を対象としたループ並列化技法とその評価. *JSPP'97*, pp. 197–204, 1997.