入出力命令を含むプログラムにおける並列性抽出法とその評価

1L－6

朝井 義久　　渋沢 進

茨城大学工学部情報工学科

1　はじめに

これまで、入出力命令を含むプログラムの並列化は、入出力命令の含まれていない部分についてのみ並列化を行い、入出力命令部分については逐次で実行されていた。この方法では、プログラム全域にわたる並列化が行えず、並列化の効率も悪くなってしまう。

そこで、我々は入出力命令を含まないプログラムの並列化で利用されている最早実行開始条件解析法[1][2][3]に、入出力命令間の依存関係としてデバイス依存を導入することにより並列化を行ってきた[4]。

本稿では、デバイス依存を用いた並列化手法ならびに並列計算機上での実行方法について報告する。また、タスクグラフを用いたシミュレーションによる評価を行い、実際の並列計算機上での実行についても述べる。

2　並列化手法

プログラム全域にわたる並列性の抽出は、プログラムの分割により生成したタスクの実行開始条件に基づき、タスクを並列に実行することにより自動的に行われる。ここでは、その方法について述べていく。

2.1　プログラムのグラフへの変換

まず、プログラムをいくつかのタスクに分割する。入出力命令に関しては、実行時間がかかるため１つの入出力命令を１つのタスクとする。

そして、分割したタスク間の制御の流れを有向辺であらわすことにより、制御フローグラフが生成できる。

2.2　デバイス依存の導入

タスク間の依存関係として、従来最早実行開始条件解析法で用いられてきた制御依存、データ依存に加え、新たにデバイス依存を導入する。

デバイス依存は、複数の入出力命令の実行順序を守り、かつ同時に実行しないようにするための入出力命令間の依存関係である。以下では、デバイス依存関係を有向辺で表すことにより生成されるデバイス依存グラフの生成法と、そのグラフからデバイス依存による各々のタスクの実行開始条件を求める方法について述べる。

[デバイス依存グラフの生成法]

デバイス依存グラフ $DevDG(V_{Dev}, E_{Dev})$ は、以下に基づいて制御フローグラフ $CFG(V, E)$ から生成することができる。ただし、V_{Dev}, Vをノードの集合、E_{Dev}, Eを有向辺の集合とする。

- $V_{Dev} = \{x | x \in V, \text{入出力命令} \in x\}$
- x から $y (x, y \in V_{Dev})$ へ到達可能なとき、x から yへの CFG 内でのパスを $P = \langle x, \cdots, y \rangle$ とすると、

$$E_{Dev} = \{(x, y) | \forall z \in P, \quad x, y \notin z, \quad z \notin V_{Dev}\}$$

[デバイス依存による実行開始条件]

タスク x のデバイス依存による実行開始条件

$$= \bigwedge_{\{y | y \delta_{dev} x\}} \left(y \bigvee_{(a-b) \in Neg(y)} (a - b) \right)$$

ただし、$y\delta_{dev}x$ は yが x にデバイス依存することを表し、$Neg(y)$ は yが実行されないことが確定する CFG 内のすべての分岐の集合を表す。

2.3　実行開始条件

制御依存、データ依存、デバイス依存の各々の依存関係による実行開始条件の論理積をとることにより、最終的な各々のタスクの実行開始条件が求められる。

2.4　並列計算機上でのタスクの実行

タスク実行時には、求めた実行開始条件を用いて並列計算機上で動的スケジューリングを行う (図 1)。

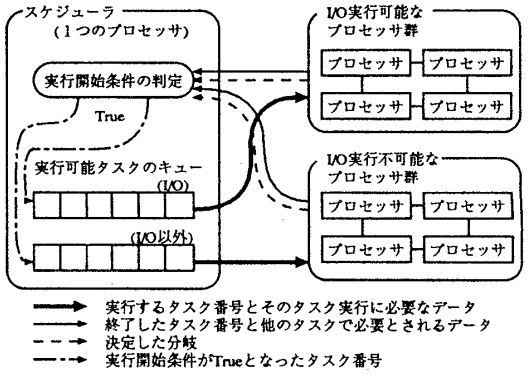

図 1: タスクの実行方式

A Parallelism Extraction Method of Programs with Input/Output Instructions and it's Evaluation

Yoshihisa ASAI
Susumu SHIBUSAWA
Faculty of Engineering, Ibaraki University
Hitachi, Ibaraki 316, Japan

この方法では、すべてのタスクの管理をスケジューラが行っているためスケジューラへの負荷が大きくなるが、入出力命令を割り当てることのできるプロセッサへのデータ転送に時間がかかると考え、データ転送回数をできるだけ減らしている。

3 シミュレーション

この節では、さまざまなプログラムにおけるデバイス依存の有効性を確かめるために、いろいろな種類の制御フローグラフを自動生成し、実行開始条件を求め、いくつかのプロセッサを想定しタスクの割り当てシミュレーションを行う。

3.1 実験の状況

- 生成する制御フローグラフ

  (i) 総タスク数、(ii) 入出力タスクの割合、(iii) データ依存密度の 3 つのパラメータを変更可能として自動生成する。

- 割り当てるプロセッサ

  割り当てるプロセッサ数は 4 つとし、プロセッサの能力はすべて等しいとする。

- 比較対象

  (i) 逐次で実行した場合、(ii) デバイス依存を用いて実行した場合、(iii) デバイス依存を用いないで実行した場合の 3 つの方法でスケジューリングしたものについて比較を行う。

3.2 実験結果

1. 総タスク数を変化させた場合

   入出力タスクの割合を 50 %とし、総タスク数を 10〜100 まで変化させた。その結果、逐次実行時間を 100 %としたとき、デバイス依存を用いた場合の実行時間は 50〜70 %であり、デバイス依存を用いない場合の実行時間は 75〜90 %であった。

2. 入出力タスクの割合を変化させた場合

   総タスク数を 50 とし、入出力タスクの割合を 10 %〜90 %まで変化させた。その結果、逐次実行時間を 100 %としたとき、デバイス依存を用いた場合の実行時間は 30〜85 %であり、デバイス依存を用いない場合の実行時間は 45〜100 %であった。

3. データ依存密度を変化させた場合

   総タスク数を 50、入出力タスクの割合を 50 %とし、データ依存密度を 0 % (データ依存無し) から 100 % (すべてのタスク間にデータ依存が存在) まで変化させた。その結果、逐次実行時間を 100 %としたとき、デバイス依存を用いた場合の実行時間は 55〜100 %であり、デバイス依存を用いない場合の実行時間は 80〜100 %であった。

3.3 考察

総タスク数、入出力タスクの割合、データ依存密度をいろいろと変化させて、デバイス依存を用いた場合、用いない場合、逐次で実行した場合の比較を行った。その結果、これらの可変パラメータに関係なくどの場合にも、デバイス依存を用いた場合が最も実行時間が短くなっていた。また、デバイス依存を用いた場合に常に正しく実行されることも確認された。

4 並列計算機による実行

2節で述べてきた並列化手法を用いて、富士通の並列計算機 AP1000 を用いて実行した。AP1000 では、ホストプロセッサ 1 つとセルプロセッサが複数個あり、入出力命令はホストプロセッサのみで実行可能である。そのため、ホストプロセッサを入出力命令専用プロセッサとし、スケジューラをセルプロセッサ上に配置した。

ファイル入出力、キーボード入力、画面出力を含んだいくつかのプログラム例について実行した結果、正しく実行されることが確認できた。また、デバイス依存を用いた方が用いない場合に比べより高速に実行できたが、プログラムにより逐次実行よりも遅くなる場合もあった。

5 おわりに

本稿では、まず入出力命令を含むプログラムの並列化のために新たにデバイス依存を導入した。そして、デバイス依存の有効性をシミュレーションを行うことにより評価し、実際の並列計算機を用いて正しく動作することを確認した。これにより、プログラム内に複数の入出力命令が含まれる場合においても、プログラム全域にわたる並列化が行えるようになった。

今後は、実際の並列計算機上での実行のさらなる高速化や、デバッガにおける入出力命令の扱いについて考察していく。

参考文献

[1] M.Girkar and C.D.Polychronopoulos: Automatic Extraction of Functional Parallelism from Ordinary Programs, IEEE Trans. Parallel & Distributed Syst., 3, 2, pp.166-178(Mar.1992).

[2] 本多弘樹, 岩田一彦, 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, 信学論 (D-I), J73-D-I, 12, pp.951-960(1990-12).

[3] 笠原博徳 : 並列処理技術, コロナ社 (1991).

[4] 朝井, 渋沢 : 入出力命令を含むプログラムの並列化, 信学技報, CPSY95-78(1995-10).