1J−10

データフロートレーサによる並列論理型言語 Fleng のパフォーマンスデバッギング

館村 純一, 小池 汎平, 田中 英彦
{tatemura,koike,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学工学部*

1 はじめに

並列計算機の実用化と普及のためには、実際に用いられるアプリケーションが高速に動かなければならない。このためには、効率のよい並列プログラムのためのプログラミング技術・プログラミング環境の構築が重要であり、パフォーマンス悪化要因を修正するパフォーマンス・デバッギングの支援が必要である。

我々は、並列プログラミングにおけるパフォーマンス・デバッギングとして、プログラマが何をなすべきか、これを支援するためのプログラミング環境はどうあるべきかを考察した [1]。我々の提案するパフォーマンス・デバッギング方式では、パフォーマンスを決定する要因として、プログラミングにおけるコントロール・データ依存関係と、それを実行するためのスケジューリング・負荷分散とを分類して対処する。現在この考え方に基づいて、実行性能の高いプログラムの作成を支援するプログラミング環境の開発をすすめている。

本論文では、並列論理型言語 Fleng について、実行されたプログラムにおけるデータ依存関係を解析・提示し、性能低下の原因となる不必要な逐次性の発見を支援するツールについて述べる。

2 Fleng による並列プログラミング

Concurrent Prolog や GHC などの Committed-Choice 型言語 (CCL) は、通信・同期が記述できるように制御機能を強化した並列論理型言語であり、Fleng もこの CCL の一つである。Fleng プログラムの実行は、パターンマッチによるゴールの集合の書き換えによって進められる。プログラムは定義節と呼ぶ書き換えルールの集合である。書き換え操作をリダクション、マッチング操作をユニフィケーションと呼ぶ。ユニフィケーションは、2 つの種類に分けられる。一つはマッチング時に行なわれるガードつきユニフィケーションで、ゴール側からのデータを待つ時に用いられる。もう一つは定義節選択後に行なわれるアクティブ・ユニフィケーションで、ゴールの持つデータに値を代入する。

Fleng プログラムは、ゴールリダクションによる制御依存関係と、ガードつき/アクティブ・ユニフィケーションによるデータ依存関係を定義節という形で記述するものといえる。

3 パフォーマンス・デバッギング・ツール

大規模高並列プログラムでは、実行情報が大量なものとなり、全体の動作の把握は難しい。パフォーマンス・デバッギングの最初の課題として、大量の実行情報の中から問題となる部分を絞り込み、その部分が全体とどう関わっているかを把握する必要がある。我々の研究室では、プログラム中のパフォーマンスバグの部分を絞り込むためのパフォーマンス視覚化ツール Paf を開発した [2]。

問題部分が抽出された次の段階として、その部分がどのように実行性能に影響しているかを詳しく分析する必要がある。そこで、実行履歴とプログラムコードからデータ・制御依存関係のグラフを作って提示する。これをユーザが分析し、どの依存関係が問題となっているかを発見する。これをもとに、データ依存関係や制御依存関係による不必要な逐次性を取り除く。

以降では、Fleng プログラム実行中のデータ依存関係の解析結果に基づいたパフォーマンス・デバッギング手法と、これをもとに試作されたデータフロー・トレーサについて述べる。

4 データフロー・トレーサによる問題点の解析

Fleng のゴールの実行 (そのサブゴールの実行を含む) は、互いに依存関係を持った入出力の履歴として表現できる。入出力の履歴は、ある出力が行なわれるまでに必要となる入力がわかるような半順序関係を持った入出力の組で表される。データフロー・トレーサは、この入出力依存関係を表示する。

4.1 入出力依存関係

各入力・出力を表現するため、ユニフィケーションを以下のように表す。

- in(Variable, Term) : ガード付きユニフィケーションによるデータの入力
- out(Variable, Term) : アクティブ・ユニフィケーションによる未定義変数へのデータの代入 (データ

*Performance Debugging for Parallel Logic Programs with a Dataflow Tracer
Junichi TATEMURA, Hanpei KOIKE, Hidehiko TANAKA, the University of Tokyo

の出力)

- unify(Variable$_1$, Variable$_2$) : アクティブ・ユニフィケーションによる変数(ポインタ)同士の単一化

一つのゴール(プロセス)Gの入出力依存関係は、入力条件と出力の組を要素とする集合

$$IO(G) = \{io_i | io_i = (ICondition_i \rightarrow O_i)\}$$

で表現される。ここで、出力O_iはアクティブ・ユニフィケーションを意味し、out(V_i, T_i)、またはunify(V_{i1}, V_{i2})と表される。入力条件$ICondition$は、入力のAND関係である。

$$ICondition = (I_1 \wedge \cdots \wedge I_n)$$

$$I_i = \begin{cases} \text{in}(V_i, T_i) \\ ICondition_{i1} \vee \cdots \vee ICondition_{im_i} \end{cases}$$

4.2 タイムスタンプつき入出力依存関係

データフローのネックとなる部分(クリティカル・パス)を表現するため、入出力依存関係中の各入力には、そのデータがガード付きユニフィケーションによって初めて参照された時刻が、各出力には、そのデータがアクティブ・ユニフィケーションによって書き込まれた時刻が付加される。

この時刻には仮想時刻を用いる。これは、プロセッサ無限大で実行可能なゴールは直ちに実行されるような理想的環境における実行時間を意味する。この仮想時刻を導入するのは、スケジューリング・負荷分散の問題を最初は考えずに、プログラムに内在する並列度を対象とするためである。この他に、スケジューリングなども考慮したモデルや、実際の測定値での時刻を用いることも可能である。

4.3 パフォーマンス・デバッギングへの応用

入出力依存関係を用いたパフォーマンス・デバッギングは、必要とされる出力のためのデータフローのクリティカルパス中から、ユーザにとって不必要な依存関係を発見する過程であり、この依存関係がパフォーマンス・バグである。これを探索するためには、以下の処理を繰り返す。

1. ゴールGの入出力依存関係を調べ、ネック(クリティカルパス)となっている出力に着目する。
2. 外部との入出力依存関係で不必要な依存関係のために出力遅延が生じている場合、この依存関係を定義する部分がパフォーマンスバグである。
3. 出力遅延についての不必要な依存関係がない場合、サブゴールを調べる。サブゴールのうち着目した出力を行なっているゴールをGとする。
4. 必要な入力の与えられる時間がネックとなって着目する出力が遅延している場合、この入力を与えているゴールをGとする。

5 グラフィカル・インタフェース

データフロー・トレーサでユーザが把握しなければならない関係には、(1) 入出力間の依存関係、(2) データ間のポインタによる参照関係、(3) 各入出力の時間関係の三種類がある。トレーサは、これらを分かりやすくユーザに示すためにグラフィック表示を行なう。図1は試作版の画面である。左端の画面が入力、中央が出力、右端

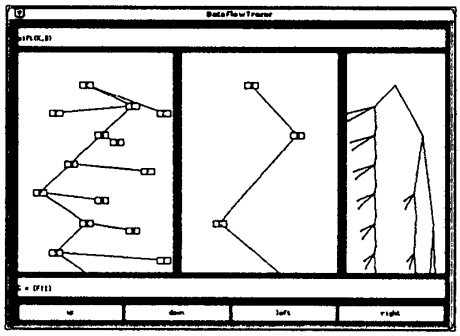

図 1: 入出力依存関係のグラフィック表示

が実行されたゴールを表す。各入出力は矩形で表され、時間関係は縦方向の座標で、データ間の参照関係は直線でつないで表される。マウスカーソルで出力データの矩形を指示しすると、対応する入力データの矩形の色が変わり、入出力間の依存関係を表す。

6 おわりに

本論文では、性能低下の原因となる不必要な逐次性の発見を支援するデータフロー・トレーサについて述べた。現在は、この試作版をもとにより実用的な機能の開発を行なっている。

参考文献

[1] 舘村, 白木, 小池, 田中: 並列論理型言語 Fleng のパフォーマンスデバッギング– Toward a Programming Environment for Efficient Highly Parallel Programs, 情報処理学会研究報告 プログラミング－言語・基礎・実践－, Vol.92, No. 67 (1992).

[2] 白木, 舘村, 小池, 田中 : 高並列プログラムのパフォーマンス・デバッギング・ツール Paf, 情報処理学会第8回プログラミング–言語・基礎・実践–研究会 SWoPP '92, 1992年8月.