データフロー解析に基づくCommitted Choice型言語のスケジューリング

3D−6

中田秀基, 小池汎平, 田中英彦
{nakada,koike,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学工学部

1 はじめに

Committed Choice型言語を高速に実行する上で障害になるものの一つに、サスペンドがある。サスペンドは、Committed Choice型言語の実行に本質的なものであるが、そのコストは大きく、これを減らすことが高速な実行には不可欠である。本稿では、データフロー解析によって各ゴールがリダクション可能になるまでの時間を静的に見つもり、その値に基づいて実行時のスケジューリングを行なうことで、サスペンドの回数を減少させる手法について述べる。

2 同期オーバヘッド

2.1 Committed Choice型言語の実行

Committed Choice型言語はすべての変数が単一代入であり、変数に値が代入されているかどうかによって、同期をとる。Committed Choice型言語の実行単位はゴールと呼ばれる。ゴールは同時に同期の単位でもある。ゴールは、その実行に必要な値が、変数に代入されているかを調べる。値が代入されていなかった場合にはそのゴールはサスペンドし、実行の対象から外される。変数への値の代入によってゴールはアクティベイトされ、再び実行の対象となる。同期の単位がゴールという細粒度のものであるため、高並列なデータ駆動型計算を自然に記述できる。一方、あらゆる場所に同期ポイントが存在するため、同期のオーバヘッドが大きく高速な実行を妨げる。高速な実行のためにはこれらのオーバーヘッドを極力減らす必要がある。

2.2 同期オーバヘッドの削減

サスペンド/アクティベイトに注目してみよう。サスペンド/アクティベイトは、実行時のスケジューリングによって減少させることが可能である。以下のプログラムを見てみよう。

```
test:-
   test1(A),test2(A).
test1(![a]).
test2([a]).
```

testをリダクションした場合、test1のリダクションがtest2のサスペンドチェックに先行すれば、サスペンドは生じない。しかしこれが逆になると、サスペンドが生じてしまう。つまりゴールのリダクションされる時刻を制御することによりサスペンドを減少できるできる。あるゴールがリダクション可能までの時間を実行開始猶予時間と呼ぶ。

3 実行開始猶予時間とスケジューリング

3.1 実行開始猶予時間

実行開始猶予時間を、以下のように定義する。

- 対象となるゴールがゴールキューに入れられてから、そのゴールがサスペンドすることなく実行可能になるまでの時間。

この時間はすべてのゴールが実行可能になると同時に実行されると仮定すれば以下の時間と等価である。

- トップゴールから親ゴールまでのクリティカルパスに存在するゴールの実行時間の和と対象ゴールへのクリティカルパスに存在するすべてのゴールの実行時間の和の差

実際には、各ゴールが実行可能になってからスケジュールされるまでの時間の和がこれに加わる(図1)。

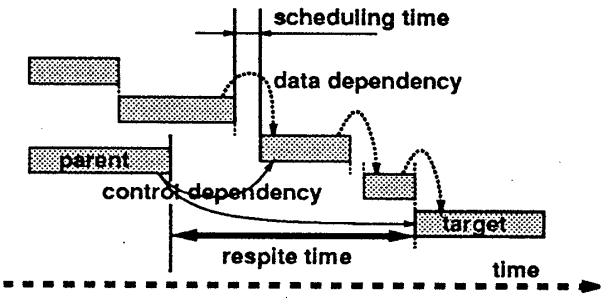

図1: 実行開始猶予時間

この猶予時間を静的に定めることは、一般には不可能である。まず、クリティカルパスは何らかのスケジューリングを仮定しないと定まらない。また、ゴールが実行可能になってからスケジュールの対象になるまでにかかる時間は、その時の並列度やIUの状態に依存し静的には定まらない。また、プライオリティ制御などの他のスケジューリングファクタの影響も考えられる。ゴールの実行速度も、外部メモリアクセスの頻度(アロケーションは完全には静的に求まらない)や、そのときのネットワークの状態などに影響される。

A Schedule method for Committed Choice Language based on data flow analysis
Hidemoto NAKADA, Hanpei KOIKE,
Hidehiko TANAKA
Faculty of Engineering, University of Tokyo

3.2 世代数による実行開始猶予時間

このように猶予時間を静的に正確に求めることはできない。そこで、実行開始猶予時間に世代という概念を導入する [日高 92]。スケジュールに用いるゴールキューを 1 つでなく複数用意し、実行開始猶予時間の各世代に割り振る 2。実行開始猶予時間を世代数で得、ゴールをその世代数だけ後ろのゴールキューに入れてやるのである。最も古いキューのゴールをすべて処理し終ったら、キューをずらし、次のキューをスケジューリングの対象とする。このようにするとある世代で生じた、実行開始猶予時間が 3 のゴールは、2 のゴールがすべてリダクションされるまでリダクションされず、その結果サスペンドを避けられると考えられる。

この方法は、スケジューリングとしては breadth first に近いと考えられ、実行開始猶予時間は、breadth first を前提として世代数で求める。

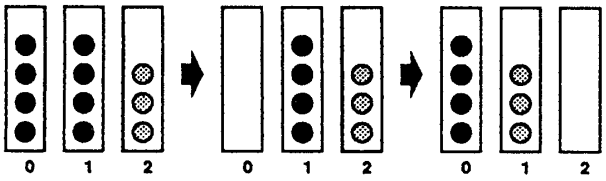

図 2: 世代別ゴールキュー

3.3 ゴールの実行開始猶予時間による分類

プログラム中に出現する各ゴールは以下のように分類できる。

1. 常に 1 世代後にリダクション可能
2. 1 より大きい一定世代後にリダクション可能
3. 1 を含む分散した時間後にリダクション可能
4. 2 世代以降の不定世代後にリダクション可能

Committed Choice 型言語 fleng による fleng 自身のコンパイラでナイーブリバースをコンパイルした際の結果を分類したものを表 1 に示す。このデータは fleng 自身で記述された世代ごとに実行するメタインタプリタを用いて得た。

表 1: nrev コンパイル時の fc

type	種類	種類 (%)	総量	総量 (%)
1	312	64.1	10943	65.5
2	51	10.5	1410	8.4
3	54	11.1	2910	17.4
4	70	14.4	1435	8.6

2 に分類されるゴールに関しては、その一定世代数後のゴールキューに入れることでサスペンドを避けることができると考えられる。1 に分類されるゴールに関しては、サスペンドのチェックの必要がなく、4 に分類されるゴールは始めからサスペンドさせてしまうことによって、それぞれサスペンドチェックのオーバヘッドを避けることができる。

ゴールの分類に関して十分な知識が得られていれば、1, 2, 4 の 82.6% のゴールに関しては同期のオーバーヘッドを削減することができる。

4 静的解析による実行開始猶予時間の導出

前章で示した分類はプロファイルの結果得られたものである。しかし、コンパイル時にプロファイルを行なうことは得策であるとはいえない。プロファイルの実行には時間がかかるだけでなく、プログラムのすべてを実行するような入力データを作成することは難しいからである。プロファイルにかわる手法として静的解析が考えられる。

4.1 実行開始猶予世代数が定まる条件

以下の場合には一般には、test1 の実行開始猶予時間は定まらない。

```
test(A):- test1(A).
test1(a).
```

これは、test からみて、test1 が必要とするデータがいつ用意できるかが、不確定だからである。

しかし、test の実行される環境を特定すれば求まる可能性がある。例えば、test(a) というように呼び出すことがわかっていれば、test1 の猶予時間は 0 であることがすぐにわかる。

このように一般には定まらないケースでも、ゴールの環境を特定することで定まる場合もある。複数の箇所から呼び出されている場合にはそれぞれを異なる述語であると考えることで、環境を特定することもできるだろう。

ある計算木の中でデータのフローが完全に閉じていれば、その計算木の内部の猶予時間は求まる。データのフローが閉じていなくても求まる場合がある。外部からのデータが、内部でコミットに影響しない場合がそうである。また、当然であるが、出力のデータフローが存在することは問題ないと考えられる。

このような計算木を仮想的に作成し、それぞれの内部で猶予時間を決定する。

5 おわりに

Committed Choice 型言語の実行においてゴールの性質に従ってスケジューリングすることによってサスペンドを減少させることができることを示した。また、データフロー解析によってを用いてゴールの性質を静的に見つもる方法に関して述べた。現在、静的解析手法の詳細を検討中である。

参考文献

[日高 92] 日高康雄, 小池汎平, 田中英彦. Pie64 の並列処理管理カーネルのアーキテクチャ. 情報処理学会論文誌, Vol. 33, No. 3, pp. 338–348, 1992.