2P－7

Committed Choice型言語Flengにおけるサスペンドコストの削減

馬場 恒彦, 荒木 拓也, 日高 康雄, 小池 汎平, 田中 英彦
{tbaba,araki,hidaka,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 Flengは単一代入変数とサスペンド・アクティベイトの機構により，非定型的な問題に対しても並列度を最大限抽出し実行可能な言語である．しかし，現在並列推論マシンPIE64上で実装されているFleng処理系はサスペンドによるオーバーヘッドが大きい．その理由は，サスペンドする場合にはゴールがフォークされてからサスペンドするまでの処理，すなわちゴールキューハンドリング・ディスパッチ・ユニフィケーションが全て無意味な処理となってしまうからである．

本稿では，ゴール生成時にゴールキューにエンキューするのではなく，いきなりサスペンド処理を実行することによって，オーバーヘッドを削減する手法について述べる．

2 研究の背景

2.1 細粒度並列処理言語 Fleng

Flengのプログラムは以下のような定義節を並べたものである．

$$Head : -Goal_1, Goal_2, \ldots, Goal_n$$

:- の左側をヘッド部，右側をボディ部という．Flengの計算単位はゴールと呼ばれるものであり，実行可能なゴールが与えられるとゴールと同じヘッド部を持つ定義節が選択される（ヘッドユニフィケーション）．選択されたゴールはボディ部の各ゴールに展開される（リダクション）．展開されたゴールはそれぞれ並列にヘッドユニフィケーション，リダクションが行なわれる．

変数は単一変数であり，書き換えることはできない．変数は未定義か，値を持つかの2つの状態しか持たず，未定義な変数を参照しようとするとサスペンドする．サスペンドしたゴールは未定義な変数がバインドされた時点でアクティベイトされる．Flengではこの性質を利用して並列計算の同期を実現している．

2.2 並列推論マシン PIE64

本研究室ではFlengを高速実行するように設計された並列推論マシンPIE64が稼働している．PIE64は64台のIU(Inference Unit, 推論ユニット)によって構成され，各IUは「計算」を行なう推論処理プロセッサ-UNIRED (Unifier/Reducer)，「通信と同期」を行なうプロセッサ-NIP (Network Interface Processor)，「管理」を行なう管理プロセッサ-MP (Managemend Processor) から構成される．MPとしてSPARCを用いている．

3 サスペンド機構

3.1 従来機構での問題点

従来のFleng処理系の流れは図1のようになる．Fleng処理系では全てのゴールに対して，ゴールスケジューリング・ヘッドユニフィケーション処理を行なわれ，ヘッドユニフィケーションに失敗した場合にはサスペンドが起こる．そのため，ゴールがフォークされてからサスペンドに至るまでの処理，すなわちゴールキューハンドリング・ディスパッチ・ユニフィケーション（図1の白抜矢印）が全て無駄な処理になり，これによるオーバーヘッドは大きなものとなる．次に示すプログラム例で考える．

```
foo(true,_,false) :- .....
foo(true,false,_) :- .....
bar(A,B,C) :-
      bar2(C,D),
      foo(A,B,D),
      .....
```

この例では，bar(A,B,C)のボディゴールfoo(A,B,D)はヘッドユニフィケーションされる対象が二つある．しかし，変数Aと，変数B又は変数Cのどちらか一方が未定義である場合，ヘッドユニフィケーションすることができないため，サスペンドすることになる．

そこでサスペンドを起こすゴールにサスペンドアノテーションを付加することで，ゴールに対していきなりサスペンド処理（Immediate Suspend，図1の斜線矢印）を行い，サスペンドコストを低減させる手法を用いる．

---

*The reduction of suspend costs for Committed-Choice Language Fleng
Tsunehiko BABA, Takuya ARAKI, Yasuo HIDAKA, Hanpei KOIKE, Hidehiko TANAKA
Faculty of Engineering, University of Tokyo
7-3-1 Hongou, Bunkyo-ku, Tokyo 113, Japan

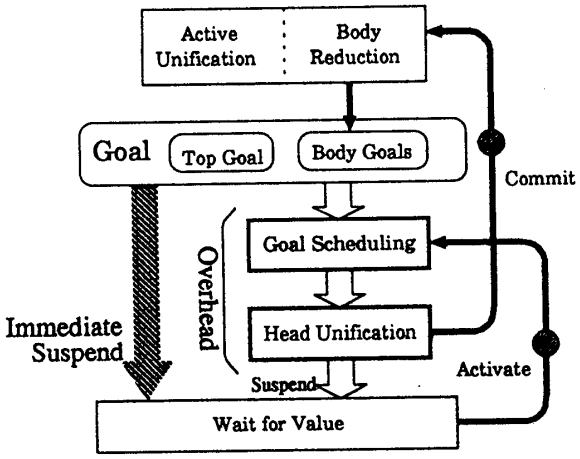

図 1: Suspend を中心とした処理の流れ

3.2 サスペンドアノテーション

サスペンドアノテーションはサスペンドを起こすゴールに対して機械的に付加され，これによりコンパイラに情報を与えることができる．

サスペンドアノテーションを付加する場合には，その変数がゴールに対して AND SUSPEND なのか OR SUSPEND なのかを記述する必要がある．AND SUSPEND とはヘッドユニフィケーションを行なうのに必須な変数に対するサスペンドのことで，OR SUSPEND はいくつかの変数のうちある変数が定義されれば実行されるような場合にその変数に対して行なわれるサスペンドのことである．

この AND SUSPEND と OR SUSPEND を考慮すると，サスペンドアノテーションの標準形は次のように表わすことができる．

Goal @ [suspend(or(and(...),and(...),...))]

実際に前述の例における bar(A,B,C) と foo(A,B,D) のプログラム例で考えると，foo(A,B,D) は，変数 A,B，又は変数 A,D が未定義の場合にはサスペンドするので，

foo(A,B,C) @ [suspend(or(and(A,B),and(A,D)))]

という形のサスペンドアノテーションを付加することになる．

3.3 処理の流れ

サスペンドアノテーションにより AND SUSPEND と OR SUSPEND を持つサスペンションレコードの構造は Fleng 処理系においては図 2のように表される．このサスペンションレコードのアロケーションは以下のような手順で実現する．

1. 対象ゴールフレームのアロケーション．
   従来のコンパイラで行なっているアロケーションと同様の処理を行なう．

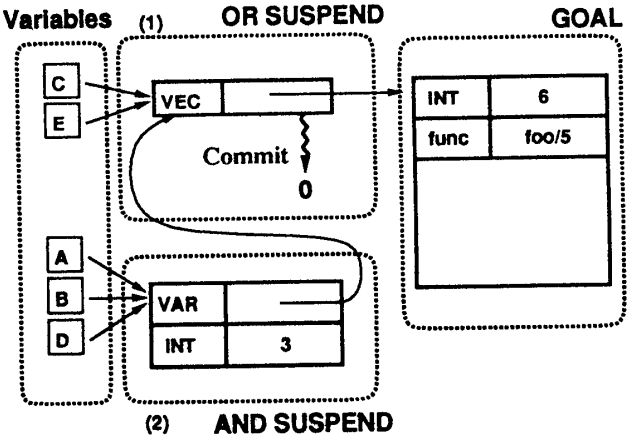

図 2: foo(A,B,C,D,E)@[suspend(or(and(A,B,D),C,E))]

2. OR SUSPEND 部 (1) のアロケーション．
   アノテーションが AND SUSPEND のみである場合も，構造上必要となるので必ずこのセルのアロケーションを行なう．OR SUSPEND の場合，ある一つの変数がバインドされればゴールがアクティベイトされる．そのため，他の変数によるアクティベイトを避けるために，0 が代入される (Commit Flag).
3. OR SUSPEND 変数のチェック．
   もし存在するのであれば，プロセス 2 でアロケートした部分に対し各変数のフックを行なった後，Suspend させる．
4. AND SUSPEND 変数のチェック．
   存在するのであれば，AND SUSPEND 部 (2) のアロケーションと各変数のフックを行ない，Suspend させる．リスト中の int 型は AND SUSPEND している変数の数を表わし，変数がバインドされるごとに値を減らしてゆく．全ての変数がバインドされた時点で値が 0 になり，ゴールがアクティベイトされる．

4 おわりに

本稿では，ゴール生成時にゴールキューにエンキューする従来方法に代えて，サスペンドアノテーションを用いることで，いきなりサスペンド処理を行なう手法を提案し，これによりサスペンドが起こる場合に生じていたオーバーヘッドが削減できることを示した．

現在，サスペンドアノテーションによるコストの削減を定量的に評価するため，実装を行なっている．

参考文献

[Araki 94] T.Araki,Y.Hidaka,H.Nakada,H.Koike,H.Tanaka: System Integration of the Parallel Inference Engine PIE64, FGCS '94 Workshop 1, p64-76, 1994.