5U-10

並列論理型言語FLENG−− のデータベース操作に関する考察

吉田 実, 田中 英彦
東京大学工学部*

1 はじめに

我々は、並列推論エンジンPIE64の開発をすすめ、また、一方、PIE64上で走るハードウェアにより密着した低水準言語FLENG−− の設計、実装をおこなっている。PIE64は、データベースマシンと通信をおこないながら一体化して一つの処理を実行する。そのためには、FLENG−− からデータベース操作を行なえるようにする必要がある。ここでは、FLENG−− からのデータベース操作の方法についての検討をおこなう。

2 言語処理システムの全体構成の構想

FLENG−− は実際の処理系の効率を意識した低水準な言語であり、ユーザが直接FLENG−− のプログラムを書くこともできるが、上位言語FLENG++処理系のコンパイルの中間言語としても使われる。FLENG++はユーザ用の高水準言語であり、並列論理型言語にオブジェクト指向パラダイムを採り入れたもので、文法的には、ESPと似た表記になっている。[2] FLENG++のデータベース操作関係のクラスは、FLENG−− を中間言語として使った場合、FLENG−− のデータベース操作コマンドを使ったコードにコンパイルされる。各処理系の関係は、図1のようになっている。PIE64上で走るFLENG−− とデータベースマシン上で走るデータベースOSがあり、共通のプロトコルで通信しあう。FLENG++は、それらを統合して扱うことができる。処理をどのマシンで行なうかと言う判断は、必要な場合、FLENG++がおこなう。このように、FLENG−− はコンパイルのための中間言語の色彩が強いので、ユーザインタフェスよりも、処理系実行時の効率を重視している。このような立場から、FLENG−− のデータベース操作コマンドは、処理をPIE64とデータベースマシンのどちらで行なうかはっきり区別されている。動的な処理ハードウェアの決定は、FLENG−− で書けないことはないが、やや書きにくいものとなる。

3 ハードウェアの概要

ハードウェアについて簡単に説明する。図2のように、全体は大きくPIE64とデータベースマシンの2つのサブシステムに分かれる。サブシステム間は、推論要素プロセッサ(IU)とディスクモジュール(DM)が、複数のパスを通じて通信する。PIE64およびデータベースマシン内部の通信(つまり、IU間通信およびDM間通信)は、PIE64とデータベース間の通信(サブシステム間通信)よりも高速である。

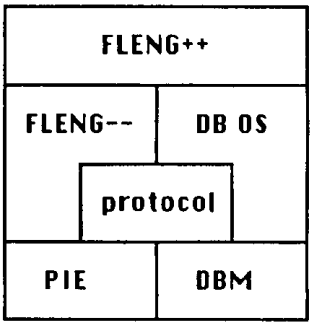

図1: 処理系全体の構成

このことから、サブシステム間通信に関しては、PIE64もデータベースマシンも一つのモジュールとしてとらえられる。通信に関しては、複数のパスに対して同期機構がないので、順序性が保証されない点に留意しなければならない。順序性は適当なプロトコルを選べば実現できるが、その場合のオーバヘッドは大きくなってしまう。

4 FLENG−−

FLENG−− は、並列論理型言語FLENGに処理効率を高めるために各種アノテーションをつけたものである。アノテーションとしては、単一参照[1]、アクティブユニファイ、非変数などがある。これらは、マニュアルで付け加えるとともに、コンパイラ、タイプチェッカなどによって付加される。[2] [3]

5 データベース操作

FLENG−− は、効率を考えた低水準言語なので、データベース操作コマンドもそれに合わせて低い水準のものとする。ただし、データベースOSに送るコマンドは、低水準である必要はなく、むしろ、高水準のものがよい。PIE64が必要とするデータベース上の各データを直接見せるという点では、低水準と言える。一般的に、高水準の操作は、FLENG++上に載せることを予定している。以下に、データベース操作コマンドの特徴をあげる。

1. 処理を推論部でおこなうかデータベース部でおこなうかは、明示的に記述する。
2. データベースは、Prologのサブセットのように見える。

*A Sugestion for Database Manipulations on Parallel Logic Programing Language FLENG−−, YOSHIDA Minoru, TANAKA Hidehiko, University of Tokyo

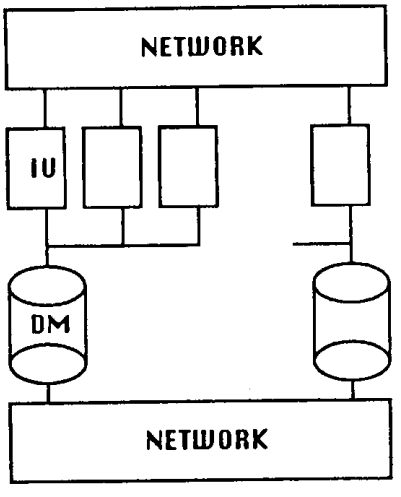

図 2: ネットワークハードウェアのトポロジ

3. 集合は、FLENG-- 上では、原則的には、リストを用いて表す。
4. データを読み出すデータベースコマンドは、原則的には、集合演算のみとする。

処理の単位は、トランザクションであり、述語 transaction/1 でコマンドの列を与える。述語 transaction に与える引数は、コマンドのリストであり、例えば以下のようなものである。

```
transaction([read(Template, Goal, Result),
append(Goal)])
```

read, append, delete は、prolog の setof, assert, retract に相当する。コマンドのリストの中で、データベースないのデータに依存性がある場合は、逐次性が保証される。データの依存性がなければ、ユーザから見ても、コマンドは並列に実行されているように見える。例えば、コマンド A,B,C があって、C は A と B に依存しているとする。このとき、

```
transaction([A, B, C])
```

のように呼び出しておいて、FLENG-- のプログラムでは、B の結果が出るまで、A へのコマンドが実行できないようになっていてもトランザクションは、デッドロックしない。データ依存性のチェックは、データベース OS がおこなう。

ところで、1 つのプログラムで、トランザクションを複数使う場合には、データベース OS の検出できないデッドロックにおちいる可能性がある。これは、複数のトランザクション間で、FLENG-- 内部で依存性のあるプログラムを書いた場合に生じることがある。この対策としては、

1. デッドロック検出機構は、データベース OS 内だけでしか行なわないという方式と、
2. FLENG-- を含めた処理系全体で検出を行なう方式

が考えられるが、プログラムに何の制約も課さずに検出するのは、一般的な、FLENG-- のプログラムのデータ依存関係まで調べなければならないので、オーバヘッドが大き過ぎて実用的ではないと考えられる。2 つ以上のトランザクション処理を同時に行なう場合には、FLENG-- のプログラムの責任において、トランザクションをまたぐデッドロックが起きないようにする。一方、誤ってそのようなプログラムを書いてしまっても、タイムアウトを使って一応は避けることができる。

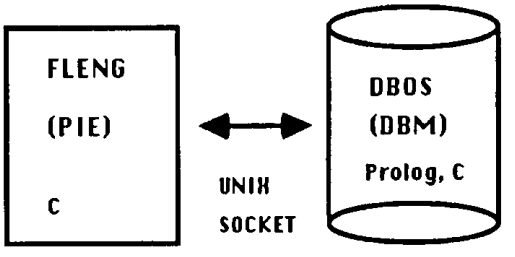

図 3: 試作処理系の構成

6 試作処理系

現在、試作中の処理系は、UNIX 上に実装されている疑似並列版で、図 3 のようになっている。FLENG-- は、C で書かれていて、[4] データベースマシンシミュレータは Prolog と C によって記述されている。それぞれのサブシステムにそれぞれ 1 個ずつのプロセスを割り当て、その間でプロセス間通信を行なっている。試作処理系は、コマンドの実行結果とサブシステム間の通信量などを見ることができ、主としてソフトウェア開発用である。今のところ、シングルユーザ用であり、トランザクション処理もおこなっていない実験的なものである。

7 まとめ

FLENG-- のデータベースコマンドは低水準であり、一般ユーザ向けではないと割り切ることによって、処理系を小さくかつ効率のよいものにすることをねらっている。しかし、一方では、この処理系の論理シミュレータは、現在、開発中であるが、完全なものにするとともに、FLENG++ 処理系のデータベース操作部分の検討および実装もすすめていく予定である。

参考文献

[1] 小池, 田中 “単一参照アノテーションを用いた論理型言語プログラムの最適化コンパイル” 本大会
[2] 中村, 田中 “並列オブジェクト指向言語 FLENG++ の実装” 本大会
[3] 小中, 田中 “Committed-Choice 型言語 FLENG のタイプ / モード推論” 本大会
[4] 島田, 小池, 清水, 田中 “PIE64 上での FLENG 実行方式” 第 37 回全国大会 5N - 6, p.145 - 146