5F−9

関数型言語の並列実行環境

三浦 雅弘　高橋 英一　谷口 倫一郎　雨宮 真人
九州大学

1 はじめに

数学的な関数の概念に基づいた関数型言語は，その参照透明性という性質から，プログラムに内在する並列性を利用して並列処理プログラムを容易に導出することができる．また，セマンティクスが単純明快であることから，プログラムの機械的な変換や検証を行うのに適しており，プログラム記述も簡潔であるなど，数多くの有利な性質を備えている．

関数型言語はこれまで数多く提案されてきたが，現在のところ従来型言語ほど普及していない．その一因として，現行の計算機上に実装した処理系ではプログラムが文字列やグラフの形で保持されており，それを実行時に複写したり部分的に更新したりしながら解釈して計算を進めていくため，実行効率が良くない，という点を挙げることができる．したがって，関数型言語を実用的に使っていくためには，効率のよい処理系が必要である．

我々の研究室では，関数型言語 Valid を密結合型マルチプロセッサ・システムで効率よく並列実行するための技術を開発している[2]．本方式ではコンパイル段階において，ソースプログラムからデータフロー・グラフを作成し，データ依存関係がなく並列実行可能な関数適用を抽出する．さらに，この関数適用を単位として，fork/join 型のプロセス生成を行うようなコードを生成する．本稿では，このように生成されたコードを並列実行するための実行環境について述べる．

2 関数型言語 Valid

Valid は，NTT 武蔵野電気通信研究所で開発されたデータフロー計算機 DFM をターゲットマシンとして開発された高級言語である[1]．Valid は再帰的定義を含む関数型言語を基本としており，タイプ付き言語を指向し，Algol 系シンタックスを持つ．また，並列処理/逐次処理を制御するためのプリミティヴを備え，ループを再帰式で記述しやすくしているなど，データフロー計算機での並列実行を意識した言語設計になっている．

3 ターゲットマシン

本研究でターゲットとする計算機は Sequent S2000 で，20個の Intel i80486 が密結合されている[3]．商用の並列機では一般に，タスクの粒度を細かくしすぎると，タスク管理のオーバヘッドが大きくなって実行効率が落ちる．このため本方式では，関数適用を単位とし，fork/join 型プロセス生成の概念に従ってタスクを並列実行する．

4 並列実行環境

並列実行環境は，C言語のライブラリとして実装されている[4][5]．Valid をコンパイルして生成されるコードは，Cの関数の形で並列実行環境に渡される．

4.1 構成

並列実行環境の構成を図 1に示す．

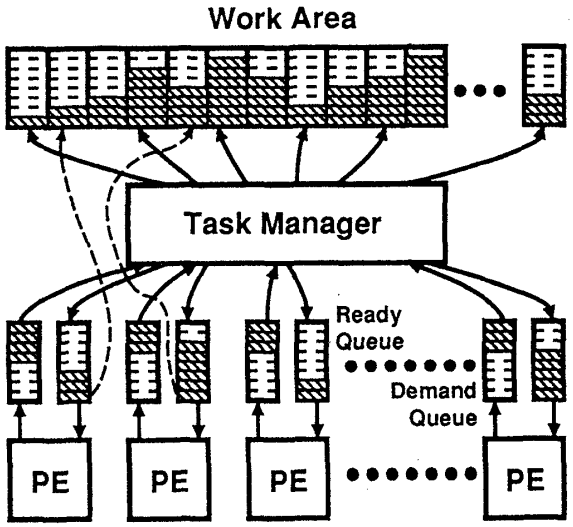

図 1: 並列実行環境の構造

Work Area は，各タスクを実行するのに必要なスタックなどの作業領域である．PE が実行中のタスクを中断して別のタスクに切替わるときは，スタックなどの情報はこの領域に残しておく．

Ready Queue には，各 PE に割当てられたタスクが入る．

あるタスクがプロセス生成を要求すると，その要求は Demand Queue に入る．Task Manager は，各 Demand Queue に入ってきたプロセス生成要求を Ready Queue に振り分ける．

4.2 並列実行の仕組み

まず，トップレベルの関数に相当するタスク（親タスクを持たない唯一のタスク）をどれかの PE の Ready Queue

Parallel Execution Environment for Functional Languages

MIURA Masahiro, Eiichi TAKAHASHI, Rin-ichiro TANIGUCHI, Makoto AMAMIYA
Kyushu University

に詰めておいてから, 各 PE を起動する. 各 PE は各自の Ready Queue に入っているタスクを順次実行する.

あるタスクがプロセス生成を要求すると, PE はその要求を Demand Queue に詰め, 元のタスクを続行する. この要求は Task Manager によって, 負荷の軽い PE の Ready Queue へと移される.

あるタスクが終了すると, PE はそのタスクを生成した親タスクの状態を調べ, 実行可能なら実行する. そうでなければ, 新たなタスクを Ready Queue から取出して実行する.

タスクが自分の生成した子タスクの終了を待つ場合, 既に子タスクがすべて終了していれば, PE はそのまま残りを実行する. そうでなければ, PE は別のタスクに切替わり, 元のタスクの残りは子タスクに任せる.

親を持たないタスクが終了すると, トップレベルの関数の評価が終ったことになるので, プログラムの実行を終了する.

4.3 プログラム例とその動作

Fibonacci 関数を計算するタスクの例を示す.

```
#include "par_env.h"

void
fib(n, pnResult)
    int n, *pnResult;
{
    BARRIER br = 1;
    int nRes1, nRes2;

    if (n < 2) {
        *pnResult = 1;
        return;
    }
    forkTask(n-1, &nRes1, fib, &br);
    forkTask(n-2, &nRes2, fib, &br);
    barrierSync(&br);
    *pnResult = nRes1 + nRes2;
}
```

関数の結果は, 必ずアドレス参照によって親へと返される.

`forkTask(n-1, &nRes1, fib, &br);` により, `n-1` と `nRes1` のアドレスとを引数として, タスク `fib` の生成を要求する. この子タスクの実行が終了すると, その結果が `nRes1` に入る. `n-2` についても同様である.

`nRes1` と `nRes2` とを使って計算を行うためには, 生成した 2 つのタスクが終了していなければならない. `barrierSync(&br)` により, `forkTask` のときに `br` を指定したタスクが全て終了するのを待つ.

5 性能評価

256 × 256 の正方行列の積の計算における台数効果を図 2 に示す.

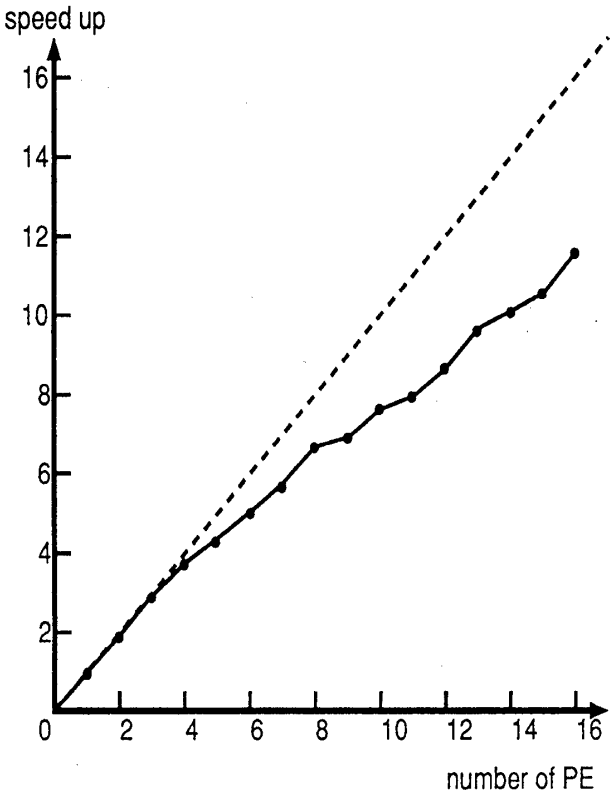

図 2: 台数効果

6 おわりに

本稿では, 関数型言語を商用の並列計算機で並列に実行するための実行環境について述べた.

現在の実行環境におけるスケジューリングは, 各タスクの実行時間の期待値が全て等しいものとして, Ready Queue の短い PE にタスクを割り振るだけのものである. 今後は, Valid をコンパイルする段階で各タスクにこの期待値の情報を持たせ, 実行環境側でこの情報を参考にして動的スケジューリングを行うように改良する.

参考文献

[1] 長谷川: “データフロー計算機方式に関する研究”
[2] 高橋, 谷口, 雨宮: “データフロー解析による関数型言語の自動並列化コンパイラ”, 九州大学第 9 回計算機科学研究報告
[3] Sequent Systems: “Symmetry Technical Summary”
[4] Sequent Systems: “Sequent Guide to Parallel Programming”
[5] Sequent Systems: “Sequent C Compiler User's Guide”