PARDS: データフロー同期を用いた並列プログラム用ライブラリ

荒 木 拓 也†

1. はじめに

近年，CPU の周波数向上による性能向上が限界に達しつつあるため，マルチコア CPU を用いた性能向上が必要だと言われている．実際，最近の PC の多くがマルチコア CPU を搭載している．その一方，マルチコア CPU を有効に生かすソフトウェアは普及していない．これは，pthread 等のライブラリを直接用いた並列プログラミングは容易では無いためである．

この問題を解決するため，並列プログラミングをサポートするライブラリ PARDS (A library for PARallel programs with Dataflow Synchronization) を開発した．本ライブラリは UNIX 系 OS 上の C++を対象とする．

本ライブラリでは，“SPAWN” マクロにより，関数等をプロセスとして fork する．これにより，並列動作する部分のアドレス空間が分けられるため，非決定的なバグが起こりにくい．また，関数が「スレッドセーフ」かどうかを気にせずに並列化できるため，既存のプログラムを並列化する際に有効となる．最近の OS では，プロセス生成におけるメモリコピーは Copy-On-Write 機構により遅延されるため，プロセス生成のオーバヘッドはスレッド生成と比べてそれほど大きなものにはならない．

プロセス間で通信および同期を行うためには，`Sync<T>` 等の class template を用いる．`Sync<T>`型の変数はプロセス間で共有され，`read()`, `write()` などの操作を行うことができる．ここで，`read()` は同一変数に対する `write()` が行われるまで待たされる．このようなデータフロー同期を利用することで，プロセス間の通信および同期を容易に実現できる．

本ライブラリを用い，実用的なプログラムとして bzip2 を対象に並列化を行うことで，その記述性と性能を確かめた．

本ライブラリは，Sourceforge において実装およびマニュアルを公開している (http://pards.sourceforge.jp/)．また，Fedora 7 以降では，yum install pards pards-devel でもインストール可能である．

† NEC サービスプラットフォーム研究所

2. PARDS の機能と実装

PARDS を用いた簡単なプログラムの例を示す．

```
int main(){
  pards_init();        // 初期化
  Sync<int> a, b;
  SPAWN(add(1,a,b));  // add を fork
  a.write(1);          // add が実行可能に
  printf("value = %d\n",
         b.read());    // add の終了を待つ
  pards_finalize();}  // 終了処理
void add(int x, Sync<int> y, Sync<int> z){
  z.write(x + y.read());}
```

このプログラムでは，`SPAWN` で fork した `add` の中で 1 + 1 を実行し，その結果を `printf` で出力している．fork した `add` は `y.read()` で第 2 引数の値が決まるまで待つため，`main` で `a.write(1)` が実行されるまで待つ．また，`main` の `b.read()` は，`add` の `z.write()` が実行されるまで待たされる．

ここで，`SPAWN` は cpp マクロとして実現されており，単純化すると，

```
#define SPAWN(f) if(fork()==0){f; exit(0)}
```

のように実現されている．また，`Sync<T>`は System V IPC の共有メモリとセマフォを用いて実現されている．

`read()` や `write()` は共有メモリ領域とのコピーが必要なため，大きなデータを扱う際には，性能が低下する可能性がある．そのため，直接共有メモリ領域を確保するライブラリコールも用意している．

また，生産者・消費者問題を簡単に記述するため，`Sync<T>`をリストにした `SyncList<T>`および，その長さを制限可能とした `SyncQueue<T>`を用意した．

```
SyncList<int> *a = new SyncList<int>;
SPAWN(generator(a));
SPAWN(consumer(a));
```

などのように作成し，`generator` の方では

```
for(i = 0; i < 10; i++){
  current->write(i);
  current = current->create();}
```

のように write() による値の設定と，create() による cdr の作成および設定を行う．create() は省略記法であり，cdr セルの作成および設定を直接行っても良い．consumer の方では，

```
while(1){
  printf("value = %d\n",current->read());
  current = current->release();
  if(current == 0) break;}
```

のように，read() による値の読み込みと，release() による読み込み後不要となったリストセルの解放および次のセルの読み込みを行う．release() も省略記法であり，利用済みのセルの delete と，次のセルの読み込みを直接行っても良い．release() は create() によって cdr が設定されるまで待たされる。

SyncList<T>の代わりに SyncQueue<T>を用いることで，リストの長さを制限することができる．これにより，生産者が先に走り続けることで資源が枯渇することを防ぐことができる．

また，プロセスの生成数をおさえたい場合のため，WorkPool<T1，T2>というクラスも用意している．本クラスを用いることで，プロセスがリストから仕事を取り出し，その結果を仕事が設定されていた順でリストに出力する，という処理を記述することができる．

また，その他の機能として SPMD プログラムの記述もサポートした．また，並列処理だけではなく，並行処理をサポートするため，タイムアウト付きの read() 系関数等や，プロセスに対するインタラプトの送出，SyncList, SyncQueue の非決定的なマージ機能をサポートしている．

3. 評　　価

実用的なプログラムの並列化例として，bzip2 を並列化した．

bzip2 はファイルを一定の大きさのブロックに分割し，それぞれを圧縮する．このため，並列化のための戦略として，

- 生産者・消費者パターンを用い，「ファイルからデータを読み出すプロセス」「バッファの内容を圧縮するプロセス」「圧縮した内容を書き出すプロセス」に分割する．それぞれはパイプライン的に動作する．
- 圧縮するプロセスでは，複数のブロックに分割されたデータに対し，それぞれを圧縮するプロセスを起動する．

とした．この並列化のために要したプログラムの変更は，71 行の削除と，141 行の追加のみと，非常に少ないものとなっている．

表 1 性能評価

マシン	逐次版 (s)	並列版 (s)	速度向上
HP-UX (2CPU)	14.76	8.96	1.64
Fedora 6 (2*2CPU)	3.79	1.43	2.65
Fedora 5 (2CPU)	8.96	5.55	1.61

性能評価の結果を**表 3** に示す．HP-UX のマシンは IA-64, Fedora Core 6 のマシンは DualCore Xeon を 2 台，Fedora Core 5 のマシンは Xeon を 2 台搭載している．圧縮対象のファイルは，10MB 程度，ブロックサイズはデフォルトの 900KB である．

問題の並列度が低かったためか，4CPU のマシンではやや速度向上は低いものの，いずれの場合も十分に並列化されて実行されていることが分かる．

4. 関 連 研 究

データフロー同期を用いる手法は良く知られており，例えば MultiLisp の future や，Id の I-structure，並列論理型言語の論理変数などが相当する[1)]．本ライブラリは，新しい言語ではなく，ライブラリ層として提供している点に特徴がある．これにより，fork によるアドレス空間の分離とともに，特に既存のプログラムを並列化するのに適している．

類似のライブラリとしては，MPC++ MTTL が存在する[2)]．同期変数の機能も類似しているが，SCore を前提としており，またプロセスではなくスレッドを用いる点で本ライブラリとは異なる．

5. お わ り に

本論文では，並列プログラミングをサポートするライブラリ PARDS の開発およびその評価を行った．今後の課題として，Windows 系 OS への移植や，スクリプト系言語等他の言語への移植があげられる．

謝辞 PARDS の rpm 化およびリポジトリへの登録を行って頂いた MASA.H 様に感謝致します．

参 考 文 献

1) Wikipedia: Futures and promises. http://en.wikipedia.org/wiki/Futures_and_promises.
2) PC Cluster Consortium: MPC++ マルチスレッドテンプレートライブラリ. http://www.pccluster.org/score/dist/score/html/ja/ reference /mpcxx/mttl.html.