Sushi
5T－4－ プロセスの再配置可能な並列分散プログラミング言語 －

山中英樹，鵜飼孝典，上田晴康，菅野博靖，和田裕二
e-mail: yamanaka@flab.fujitsu.co.jp, {ugai,ueda,suga,wada}@iias.flab.fujitsu.co.jp
〒261 千葉市美浜区中瀬 1-9-3，（株）富士通研究所 情報社会科学研究所

1. はじめに

分散メモリ型の並列コンピュータや，ネットワーク上の計算機群を効率良く利用するソフトウェアの生産性の向上が課題となっている．一般に並列に動作するソフトウェアの記述は，その非決定性などから非常に困難である．さらに CPU などの資源を効率良く利用するために，ある特定の環境においてどの計算をどの CPU で実行させるかといった，資源の割当をアルゴリズムとして明示的に記述しなければならない．そこで我々はプログラムの中でアルゴリズムを記述する計算部分と，アルゴリズムを変更しないアノテーションと呼ぶプロセスの CPU への割り当てや実行の優先度の変更を記述する部分を各々独立に記述するプログラミング言語 Sushi (Smartly User Schedulable HIgh-level language) を設計した．このアノテーションはプロセスのマイグレーションの戦略，実行優先度の動的な変更などを記述する特殊なプログラムで，各プロセスに一つのアノテーションを付加することが可能である．これまでにも CPU の割当などをアルゴリズムと独立に記述するプログラミング言語 PCN [PCN92] は提案されているが，Sushi のように実行中にプロセスのマイグレーションや実行優先度の変更などを行なうことはできない．

計算部分は，複数のプロセスと呼ばれるモジュールとそれらを結ぶストリームで構成され，プログラムを実行すると，各プロセスがストリームを通してデータを流しながら並列に動作する．このような構成で計算部分を記述することによる利点として

a. データの流れにしたがってアルゴリズムが素直に表現できる．
b. 部品となる単純なプロセスを組み上げてより複雑なプログラムを構成することができる．
c. プロセス間のデータの遣り取りがストリームを通して行なわれるため，モジュラリティが高く，プログラムの部品化，再利用に適している．

などがあげられる．ストリームは，これまでデータフロー言語 [ARV78]，関数型言語 [HEN80]，論理型言語 [SHAP83]，Pascal にストリームを導入した言語 [KUS90] など様々な研究がなされて来た．Sushi は，特に次のような設計をしている．

a. 既存の多くのプログラマが容易に利用できるように，各プロセスを C に似た構文で記述する．
b. 簡易にプロトタイプが作成できるように，データ構造を文字列と連想配列だけにした．

2. Sushi のプログラミング

Sushi プログラムは，二つの部分から構成される．一つは，実際に行なう処理のアルゴリズムを記述する計算部分で，一つの main という名のプロセスを含む複数のプロセスとして記述する．他方は，各プロセスのマイグレーションの戦略，実行優先度の動的な変更を記述する特殊なプログラムであるアノテーション部分である．これらは，コンパイル時の処理でも別々で，各々が分割コンパイルの対象になる．

プログラムの計算部分は，複数のプロセスと呼ばれるモジュールとそれらを結ぶストリームの記述で構成される．プロセスが扱うデータ構造は，文字列と連想配列で，その内ストリームに流すことのできるデータ構造は文字列だけである．Sushi のストリームは，複数のプロセスからアクセス可能な，文字列をその要素とする無限長のキューである．したがって，書き込みのブロック状態は起きない．プログラムの計算部分の実行は，プロセスの生成，各プロセスの並列動作，プロセス間のストリームによる通信により行なわれる．ストリーム中のデータの流れる方向は一方向であり，モジュールへ入って来るもの（入力ストリーム），モジュールから出て行くもの（出力ストリーム）がある．プロセスへのストリームの接続は，プロセスの生成時にプロセス生成のパラメータに指定された入出力ストリームに基づき行なわれる．したがって，ストリームは，プロセスの親子関係に従って相続される．

各モジュール（プロセス）は，入力ストリームからデータを読み出し，そのデータに一連の処理を行ない，そして出力ストリームにデータを書き込む処理を c に似た構文で記述される．ストリームへの書き込み，読み出しは，並列動作する複数のプロセスから随時行われる．しかし，書き込みはブロックされることがないが，読み出しは入力ストリームが空の場合ブロックされ，他のプロセスがそれに対応する出力ストリームにデータを書き込むか，その出力ストリームが全て閉じられるまで待たされる．プロセスの終了時には，そのプロセスに接続されている全ての入出力ストリームを自動的閉じるので，必ずしもプログラマが明示的に出力ストリームを閉じる必要がない．これから，例を使ってプログラミングを具体的に説明する．

図 1. は，良く知られたエラトステネスのふるいによる素数生成を Sushi で記述したものである．spawn 文によってプロセスが生成されるので，プロセスの種類は generate と sieve の二つで，sieve プロセスの中で，さらに spawn 文により sieve プロセスが再帰的に生成される．~inp,out~ などの ~ の付いた変数は，ストリームを表し，前に ~ の付いているものが入力ストリームで，後ろに付いたものが出力ストリームである．n << ~inp は，入力ストリーム ~inp からデータを取りだし，変数 n に代入し，n >> out~ は，変数 n の値を出力ストリーム out~ に書き込むことを表す．

Sushi – A Parallel and Distributed Programming Language with Voluntary Process Migration –, Hideki Yamanaka, Takanori Ugai, Haruyasu Ueda, Hiroyasu Sugano and Yuji Wada, Fujitsu Laboratories, 1-9-3 Nakase, Mihama-ku, Chiba 261, Japan

上で述べたように，プロセスの生成は spawn 文を使って記述され，アノテーションと呼ぶ付加的な記述をつけることが可能である．このアノテーションはプログラムのアルゴリズムを変更せず，プロセス生成時にプロセスを割り当てる CPU の決定を行なうための記述とプログラムの実行の単位であるプロセスを物理的に違う多数の CPU 間を移動させるプロセス・マイグレーションの戦略，実行優先度の動的な変更などを記述する特殊なプログラムで，アルゴリズムを記述する言語とは独立になっている．

```
main(stdout~) {
  spawn generate(num~);
  spawn sieve(~num,stdout~);
}

generate(num~) {
  i=1;
  while(1) ++i >> num~;
}

sieve(~inp,out~) {
  n << ~inp;
  n >> out~;
  spawn sieve(~tmp,out~)@go_neighbour();
  while (1) {
    x << ~inp;
    if (x%n!=0) x >> tmp~;
  }
}
```

図 1. 素数生成（エラトステネスのふるい）

Sushi では，アノテーションをプロセスの生成時にのみではなく，プロセスの実行中に何度も計算することにより，プログラムの使われる環境，実行状態を細かく反映したプロセス・マイグレーション，実行優先度の変更を可能にしている．アノテーションは，基本的な計算機環境 (各 CPU の負荷，位置，各ストリームの通信量，各プロセスの計算時間，実時間など) を示す数値をシステム変数として利用でき，その変数と各種算術演算を組み合わせた数式を元にプロセス・マイグレーションと実行優先度を決定する．このアノテーションは，汎用性の高いものを記述し，それを利用することも可能であるが，特定の計算機環境に最適化したい時には，プログラムの中で使われるアノテーションをその計算機独自の新しいものと置き換え，リンクし直すことが簡単にできる．アノテーションを付けずにプロセスが生成された場合は，システムが用意するデフォルトのアノテーションを用いる．

```
/*
システム変数
  MY_CPU     : 現在プロセスが動いている CPU
システム関数
  neighbours(MY_CPU): 自分の隣の CPU の配列
  load(MY_CPU)      : 自分の負荷
*/

annotation go_neighbour() {
  init: {
    if(load(MY_CPU)>load(MY_CPU+1))
       spawn_at(MY_CPU+1);
    else swawn_at(MY_CPU);
  }
  neighbour[]=neighbours(MY_CPU);
  for(i in neighbour[])
    if(load(i)<load(MY_CPU)/2) migrate(i);
}
```

図 2. アノテーションプログラム

ここで，アノテーションの記述例 図 2. を上げる．アノテーションはプロセスの生成時に実行される部分と，プロセスの実行中に時々実行される部分からなる．プロセスの生成時は，`init: {···}` の部分だけが実行され，新しいプロセスが起動されようとしている CPU の負荷の半分より次の CPU の負荷が低い場合は，隣の CPU でそのプロセスが起動される．その後の部分は，次回以降の実行時に実行される．現在プロセスを実行中の CPU の負荷の半分よりその負荷が低い，隣接する CPU があるとき，その CPU にプロセスをマイグレートする．

3. まとめ

本稿では，計算部分と呼ばれるプログラムのアルゴリズムを記述する部分と，アノテーションと呼ばれるプロセス・マイグレーションやプロセスの実行優先度をプロセスの実行途中での動的な変更を記述する部分を独立に記述し，どのアノテーションをどのプロセスで利用するかを，計算部分のプロセスの生成命令 spawn 文で指定可能なプログラミング言語 Sushi の設計について述べた．現在，我々は，Sushi を LAN 上のワークスティション群で走らせるために鋭意実現中 [UGA94,YAM94] である．

また，アノテーションの記述は計算機システムの設置されている環境，すなわちネットワークや各 CPU の性能などを十分熟知して記述する必要があり，そのため通常のプログラマは，既存のアノテーションライブラリを検索して，適当なアノテーションを選んでは試しにリンクすることを繰り返す使い方をすると予想される．そこでグラフィカルな性能評価ツール [WAD94] の開発を別途進めている．

謝辞

本研究を進めるにあたり，Sushi の最初のユーザとして意見を頂いた蓬莱尚幸氏に感謝する．

参考文献

[ARV78] Arvind and J.D. Brock, **Streams and Managers**, LNCS, Vol.143, pp.452–465, 1978.

[HEN80] P. Henderson, **Functional Programming : Application and Implementation**, Prentice-Hall, 1980.

[KUS90] 久世和貿，佐々政孝，中田育男，ストリームによるプログラミングのための言語とその実現方式，情報処理，Vol.31, No.5, pp.673–685, May 1990.

[PCN92] I. Foster, R. Olson and S. Tuecke, **Productive Parallel Programming, Scientific Programming**, Vol.1, pp.51–66, 1992.

[SHAP83] E. Shapiro and A. Takeuchi, **Object Oriented Programming in Concurrent Prolog**, New Generation Computing, Vol.1, No.1,, 1983.

[UGA94] 鵜飼孝典，山中英樹，上田晴康，プロセスの再配置が可能なストリームベース並列プログラミング言語，SWoPP'94, 沖縄, 1994.

[WAD94] 和田裕二，ストリームベースの並列分散プログラミング言語 Sushi における実行状態の可視化の試み，情報処理秋季全国大会，1994.

[YAM94] 山中英樹，並列分散プログラミング言語 Sushi のアノテーションの実現について，情報処理秋季全国大会，1994.