VIOS IV 実行ロー記述用言語 VPE-i 仕様

平成 14 年 3 月 22 日

1 概要

VPE-i は，メインフロー文において，並列処理に伴う初期化や結果の集計，表示処理を行うことを想定しており，あまり複雑な処理をこなすことを目的としていない．また，インタープリタであるための速度の問題も考慮し，プログラムは基本的にモジュール側により実行することを推奨する．

本言語は以下ような特徴を持つインタープリタ言語である．

- 基本文法，各演算子は C 言語に従う．
- ++，?演算子，ポインタなどは無し．
- 関数宣言は無し．
- 扱えるデータ型は整数，実数，文字列の 3 種であり，構造体は扱わない．
- モジュール呼び出し，分割方式の指定，などは所定手続きを呼び出すことにより行う．

2 予約語

以下に挙げる単語は VPE-i の予約語であり，変数名などに利用することは制約される．

ALL, BLOCK, CACHE, COLUMN, depth, DEPTH, DIVISION, else, fCube, fImage, float, for, gather, height, iCube, if, iImage, imgLoad, imgSave, int, module, put, RAND_MAX, random, redist, ROW, set, width, while

3 利用可能な型

- **int** 型：整数型の 1 次元情報を扱う型．通常の C 言語と同様に，単独の変数，または [ ] を利用した配列として宣言を行う (ただし 1 次元変数のみ)．
- **float** 型：実数型の 1 次元情報を扱う型．宣言方式は int と同様となる．
- **iImage** 型：整数型の 2 次元情報を扱う型であり，以下の 3 つの方式何れかにより宣言可能．
  1. **iImage** 変数名 (*width*, *height*);
  2. **imgLoad**(“PGM ファイル名”, 変数名);
  3. **iImage*** 変数名 (*width*, *height*);

  なお，3 番目の宣言はポインタ変数としての宣言ではなく，型の後に “*” を付けた宣言は，初期化の時点で領域を確保しない宣言を表す．つまり，並列モジュールの実行時に，部分領域の大きさに関する情報のみを各実行計算機にし，並列モジュール内においてデータを生成

することが可能となる．この宣言を利用する場合，VPE-i 側において一切データのアクセスが出来なくなることに注意すること．

また，変数へのアクセス方式は，data(3,4) のような形で行う．

- **fImage** 型：実数型の 2 次元情報を扱う型であり，宣言方式は iImage 型と同様となる（ただし，imgLoad による宣言は行うことが出来ない）．
- **iCube** 型：整数型の 3 次元情報を扱う型であり，以下の 2 つの方式何れかにより宣言可能．
  1. **iCube** 変数名 (*width*, *height*, *depth*);
  2. **iCube*** 変数名 (*width*, *height*, *depth*);

  変数へのアクセス方式は，data(3,4, 6) のような形で行う．
- **fCube** 型：実数型の 3 次元情報を扱う型であり，宣言方式は iCube 型と同様となる．

4 通信環境設定用構文

4.1 プラグマによる指定

リモートホストの指定や，通信形態の指定は，以下に示す#pragma 構文を利用する．なお，少なくとも最初のモジュールを呼び出す前までに指定を終了しておく必要がある．

また，現状では一度接続を確立した後での変更には対応していない．

#pragma host_name *host name*

リモートホストの登録．ローカル計算機から一意に特定できる名前なら IP でも名前でも可．

#pragma use_intercomm

リモートホストが相互に通信を行う処理の場合にセットする．具体的には，

- SyncCache 命令を利用する場合
- parallel_ie 構文が必要な場合
- user_comm 命令を利用する場合

の様に，モジュールが各計算機において完全に独立して処理されることが保証出来ない場合にセットする．なお現在の実装では，同期処理，リダクション処理を利用するためにはセットする必要はない．

通常，上記設定のみで問題ないが，以下のようなオプションを設定することも可能である．

#pragma connect_start

サーバとして各計算機で起動している vios_child に接続する．この値をセットしない場合（通常時），vios_run は rsh を利用し必要時，各計算機上に vios_child を自動起動する．
各計算機毎に異なるターミナルに出力を行いたい場合などに利用する．

#pragma div_num *number*

各計算機に送るデータをさらに細かく *number* 個に分割し，データの転送を行いながら parallel 文の実行を行う．デフォルトは 1（各計算機にまとめて転送）．

注）：現状，parallel 実行中に “存在しないデータ” が発生するため，利用する際は十分な注意が必要となる．

4.2 Set 関数による指定

並列処理に利用する各データの分割方式，キャッシュ領域の指定など，データに対する設定は，以下に示す set 構文を利用する．

set(*value_name*, DIVISION, *div_type*);

2,3 次元変数 *value_name* を並列処理する際の，大凡の分割方式を指定する．*div_type* には次のタイプの中から指定する：
{ “ALL”, “BLOCK”, “COLUMN”, “ROW”, }
デフォルトの指定は “BLOCK” となる．

set(*value_name*, DIVISION, *div_type*);

1 次元配列変数数 *value_name* を並列処理する際の，大凡の分割方式を指定する．*div_type* には次のタイプの中から指定する：
{ “ALL”, “BLOCK”}
また，デフォルトの指定は “ALL” となる．

なお分割方式については各次元の分割数を直接指定することも可能 (例えば 2 次元データ “data” を 2×2 に 4 分割する際は，set(data, DIVISION, 2, 2); とする) であるが，分割した結果生成されるデータブロック数と計算機台数が不一致の場合，動作不定となる．

set(*value_name*, CACHE, *cache_size*);

変数 *value_name* を並列処理する際，各ワーキングセットが参照するキャッシュ領域の半径を指定する．

5 VPE-i で利用可能な関数

module(“モジュール名”, *value_name*, ...);

モジュールの実行を行なう．

1 実行フロープログラムを処理する間，通常複数の並列モジュールを実行することが想定され，また連続的に実行される並列モジュールは同じタイプのワーキングセットに対する処理が行われることが多い．

そこで VPE-p では，実行速度を考慮し，各並列モジュールの実行終了後，メインプロセス (vios_run) に変数の値を暗黙的には戻さず，更新された値は各リモートホストが保持し続ける方針を取る．

そのため，全並列モジュールの実行後，値を統合して保存する場合や，先ほどまでの並列モジュールとは異なるワーキングセットに対して並列処理を行う必要がある場合，以下の命令を使用し，明示的に指定する必要がある．

gather(*value_name*);

並列モジュールの実行により更新された変数 *value_name* を，各リモートホストから回収し，実行フロープログラム中において最新の値にする．

redist(*value_name*, *type*);

gather 関数により回収した変数 *value_name* を，以降の並列モジュールにおいて異なるタイプのワーキングセットを用いて並列処理するために，回収（gather），再分割，再転送を行う．
なお再転送のタイミングは次のモジュール呼び出し時となる．

put(*value_name* | “文字列”);

指定された文字列を標準出力に出力する．

random();

C 言語の rand() と同様に，0 ~ RAND_MAX の値を返す関数．

imgSave(“PGM ファイル名”, VIOS 変数名);

1D 変数，Image 変数を PGM ファイルに出力する．

imgLoad(“PGM ファイル名”, VIOS 変数名);

型宣言の項参照．

width(“VIOS 変数名”);

指定変数の width(x 軸方向の長さ) を返す．

height(“VIOS 変数名”);

指定変数の height(y 軸方向の長さ) を返す．2, 3 次元変数のみ有効．

depth(“VIOS 変数名”);

指定変数の depth(z 軸方向の長さ) を返す．3 次元変数のみ有効．

6 文法

以下に VPE-i の構文を示す．

```
program : main_statement_list

main_statement_list
        : statement
        | main_statement_list statement

statement
        : declaration       ";"
        | selection
```

```
        | iteration
        | put_function      ";"
        | set_function      ";"
        | mass_expr         ";"
        | "{" statement_list "}"

statement_list
        : statement
        | statement_list statement

declaration
        : TYPE declarator_list

declarator_list
        : declarator
        | declarator_list "," declarator

declarator
        : NO_DEFINE
        | NO_DEFINE "[" one_expr "]"
        | NO_DEFINE "(" one_expr "," one_expr ")"
        | NO_DEFINE "(" one_expr "," one_expr "," one_expr ")"

selection
        : if.prefix statement
        | if.prefix statement "else" statement

if.prefix
        : "if" "(" cond_expr ")"

iteration
        : for.prefix  statement
        | while.prefix statement

for.prefix
        : "for" "(" mass_expr ";" cond_expr ";" mass_expr ")"

while.prefix
        : "while" "(" cond_expr ")"

put_function
        : "put" "(" put_argment ")"

put_argment
        : STRING
        | one_int_variable
        | one_flt_variable
```

```
        | int_variable
        | flt_variable

set_function
        : "set" "(" set_argment ")"

set_argment
        : address_variable "," PARAMETER "," set_argment_sub
        | address_variable "," PARAMETER "," PARAMETER

set_argment_sub
        : one_expr
        | set_argment_sub "," one_expr

mass_expr
        : one_expr
        | mass_expr "," one_expr
        | addr_int_variable     "=" one_expr
        | addr_flt_variable     "=" one_expr
        | addr_one_int_variable "=" one_expr
        | addr_one_flt_variable "=" one_expr

one_expr
        : value_variable
        | function
        | "(" one_expr ")"
        | "(" TYPE ")" one_expr %prec TOPOP
        | "++" one_expr
        | "--" one_expr
        | "+"  one_expr      %prec TOPOP
        | "-"  one_expr      %prec TOPOP
        | one_expr "*" one_expr
        | one_expr "/" one_expr
        | one_expr "%" one_expr
        | one_expr "+" one_expr
        | one_expr "-" one_expr
        | one_expr "++"
        | one_expr "--"

cond_expr
        : "(" cond_expr ")"
        | one_expr ">"  one_expr
        | one_expr ">=" one_expr
        | one_expr "<"  one_expr
        | one_expr "<=" one_expr
        | one_expr "==" one_expr
        | one_expr "!=" one_expr
```

```
        | one_expr "||" one_expr
        | one_expr "&&" one_expr

address_variable
        : iCUB_VAL
        | fCUB_VAL
        | iIMG_VAL
        | fIMG_VAL
        | addr_one_int_variable
        | addr_one_flt_variable

addr_one_int_variable
        : INT_VAL

addr_one_flt_variable
        : FLT_VAL

addr_int_variable
        : iCUB_VAL "(" one_expr "," one_expr "," one_expr ")"
        | iIMG_VAL "(" one_expr "," one_expr ")"
        | INT_VAL  "[" one_expr "]"

addr_flt_variable
        : fCUB_VAL "(" one_expr "," one_expr "," one_expr ")"
        | fIMG_VAL "(" one_expr "," one_expr ")"
        | FLT_VAL  "[" one_expr "]"

value_variable
        : constant
        | one_int_variable
        | one_flt_variable
        | int_variable
        | flt_variable

constant
        : INTEGER
        | FLOTING
        | STRING

one_int_variable
        : INT_VAL

one_flt_variable
        : FLT_VAL

int_variable
        : iCUB_VAL "(" one_expr "," one_expr "," one_expr ")"
```

```
        | iIMG_VAL "(" one_expr "," one_expr ")"
        | INT_VAL  "[" one_expr "]"

flt_variable
        : fCUB_VAL "(" one_expr "," one_expr "," one_expr ")"
        | fIMG_VAL "(" one_expr "," one_expr ")"
        | FLT_VAL  "[" one_expr "]"

function
        : FUNC "(" argment_list ")"

argment_list
        :
        | argment
        | argment_list "," argment

argment : address_variable
        | constant
        | NO_DEFINE
        | PARAMETER
```