VIOS IV 並列処理記述用言語 VPE-p 仕様

平成 14 年 3 月 22 日

1 概要

本文章では，並列処理実行部（以下モジュール）を作成するための言語 VPE-p の仕様を示す．

内容としては，C++用ソースコードを出力するプリプロセッサを通るための制限，および様々な煩雑な処理（通信など）を隠蔽するための外部プロセスコールや拡張仕様で利用している予約語，などを示す．

2 VPE-p 基本構成

VPE-p は，C 言語をベースとし，これに並列処理構文，部分大域変数およびそのアクセス方式，特殊関数を加えた言語として構成される．そのため，拡張表現を伴う構文（e.x. *parallel* 構文内，iImage 変数からの読み出しなど）以外では，通常 C 言語への制限はない．また，C++言語における，変数の途中宣言，および//によるコメントも使用することができる．

2.1 モジュール基本構成

VIOS では，複数のユーザにより利用されることを想定しているため，各モジュールファイルの統合は行わず，個別に識別するスタイルを取っている．そのため，1VPE-p プログラムファイル辺り，1 つのモジュールが記述されていることを想定している．そのため，以下のような制限が存在する．

- コンパイル時指定するファイルに，指定する形式で *vs_module* 構文が 1 つ宣言されていること．

なおこれは，あくまでコンパイル時に指定するファイルに対する制限であり，このメインファイルから複数のファイルをインクルードすることに制限はないが，VPE-p は，インクルードファイルを辿り解析することはしないため，指定するファイル以外（#include などで引用するファイル）では VPE-p の拡張表現，変数，などは利用できない．

また *vs_module* 構文は次の様に定義される．

```
vs_module モジュール名( VIOS型 変数名, ...) {

        /********      本文      *********/

}
```

- **vs_module** は typedef された予約型である．
- 引数は，メインフローから自動分割され渡されるため，ivalue, fvalue, iImage, fImage, iCube, fCube のいずれか（またはその最適化変数：**??**）をとる必要がある．
- 引数 0 でも問題ないが，その場合 *parallel* 構文は利用できない（vios_child を実行するホスト数での並列処理になる）．

注意）
VIOS では実行効率向上を図るため，**モジュール実行開始時点において引数データの受け渡し（メインプロセスからの分割・転送）が終了していない．**引数データの到着状況は *local_heigh* 関数などにより，随時確認することが可能であるが，次の関数以降の構文では完全にデータがそろっていることが保証される．

- **vsDataWait2D**() などの明示的データ待ち関数．
- **parallel, parallel_ie** 構文（構文内の処理に関しても全てのデータに対し処理されることが保証される）．

3 プログラミングモデル

VPE-p では，“ワーキングセット”と定義する基本単位に対する SIMD 型プログラミングモデルを採用している．

3.1 ワーキングセット

データ並列処理は一般に，ある特定データの小領域（pixel など）単位に対して並列性を有することが多い．例えば，Laplacian フィルタの場合，処理対象画像の各 pixel，ボロノイ分割の場合，母点を格納した配列の各要素，単位で並列処理することが可能となる．

そこで VIOS では，並列処理の軸とするデータに対し，

1. 各処理の中心となる注目要素
2. 並列処理の間，参照のみを行うキャッシュ半径
3. データ並列の依存関係

の 3 種類の情報を有する，ワーキングセットと呼ぶ単位を，それぞれ無限に生成可能な論理スレッドと 1 対 1 に対応させ並列処理を行うモデルを採用している．また 1 ワーキングセットを構成するデータ構造は，**??**章で述べる並列構文，およびメインフロー部でのセット関数により様々なタイプから選択することが可能である．

このようにして定義したワーキングセットに対し，以降に示す構文を用いて処理を記述することが，VIOS における基本的な並列処理スタイルとなる．

3.2 並列処理構文

VIOS では，任意のワーキングセットに対する処理を，以下に示す parallel 構文を利用し記述することにより，全ワーキングセットに対する処理を並列に行うプログラミング，および実行スタイルを従来より定義している．

1. **各ワーキングセット間に順序関係の無い場合**：

parallel< *WS* サイズ, ...> (*index* 変数, ...) {
　　/******** 本文 *********/
}

まず引数について，< および > に囲まれた，第 1 引数となる *WS* サイズは，1 ワーキングセットに含まれる要素の数を示す．これは，あるデータ “ブロック” を 1 ワーキングセットとして扱う場合に利用する．
次に，(および) に囲まれた，第 2 引数となる *index* 変数は，並列処理中，各ワーキングセットが自身を特定するための変数名を指定するもので，任意の名前（事前に宣言されている必要はない）が利用可能である．
また並列処理のベースとなる変数は，以下に従い決定される．

- 各次元に対応した *parallel* 文を利用するためには，その次元の変数がモジュール構文の引数に有ることが前提となる．
- モジュール引数に ，同次元の変数が複数ある場合は，引数順に最初の変数がベース変数となる．

2. **各ワーキングセット間に順序関係がある場合**
（または通信により各処理単位の処理時間に大きな差が存在する場合）：

parallel_ie< *WS* サイズ, ...>(*index* 変数, ...) {
　　/******** 本文 *********/
}

こちらの並列処理構文は，全てのワーキングセットを同等に扱うことが出来ない場合に利用する．具体的には，

- user_comm 命令などにより，ワーキングセット間に実行依存関係がある．
- プログラムの性質上，実行時までワーキングセットのキャッシュ半径を特定出来ない．
- 入力データにより，ワーキングセットのキャッシュ半径が変更される．

場合に利用する．
なお詳しい書式に関しては，以下の制約は加わるが，基本的な書式は**parallel** 構文のものと同様である．

- *parallel_ie* 構文内では，変数の新規宣言を行うことは出来ない．

example:

```
vs_module sample(iImage data1, iImage data2) {
    parallel<2,3>(lx, ly) {
        if((lx == 0) && (ly == 0)) ....

     } ....
```

この例では，2 次元変数に対する *parallel* 構文を利用している．*parallel* 構文内では，モジュール sample() の最初の引数である VIOS 変数 data1 をベース変数とし，各次元の識別変数には lx, ly という名前を利用している．また，各ワーキングセットの大きさは 2×3 ピクセルとなる．

3.3 VIOS 変数

1. **種類と宣言**
   モジュール引数として実行フローより引き継がれる変数は，ivalue, fvalue, iImage, fImage, iCube, fCube の何れかの型をとる．なお，各型の扱いは C 言語の通常変数の場合と同様である．
   注意）
   これら VIOS 変数に対するアクセスは，計算機外データへのアクセスを検出するため，通常配列アクセスに比べ負荷が大きい．
   そのため，明示的な通信（vsSyncCache 等）以外に計算機外データへのアクセスを行わない場合，最適化 VIOS 変数を利用することが**推奨**される．

   最適化 VIOS 変数を利用するためには，各変数宣言時に，
   vs_module snake(iImage_opt data, ...)
   のように各宣言の後に _opt を付けることで可能となる．
   また，以下のようにモジュール中で宣言することにより，各計算機独立のローカル VIOS 変数を宣言することも可能となる．

   VIOS 変数型 “ローカル変数名”(“VIOS 変数”);
   “VIOS 変数” と同じサイズ（ローカルサイズで）の “ローカル変数” を宣言する．
   e.x）iImage temp(data);

   VIOS 変数型 “ローカル変数名”(サイズ 1，...);
   サイズ 1,... に定義されるサイズの “ローカル変数” を宣言する．
   e.x） iImage temp(30, 40);

2. **アクセス方式**
   従来の VIOS と同様に，[ ] 演算子を利用することで，自ワーキングセットからの相対位置で，[[ ]] 演算子を利用することで，全体での絶対位置によるアクセスを行う．
   例えば，data[3][] との指定は，自ワーキングセットから x 軸方向に 3，y 軸方向に 0 移動したポイントの値を指すことになり，data[[3]][[0]] は，変数 data における (3,0) の要素を指すことになる．

- 相対アクセス方式は，その性質上，並列処理構文中のみでの指定となる．
- 全ての型の実体はポインタ形式となっているため，参照渡しのみ可能となる．そのため，実体の複製が必要な場合は，ローカル宣言を利用した後，ImgCopy 関数などを利用する必要がある．
- VIOS 変数のアドレス指定の中に，VIOS 変数を利用することは，相対アクセスにおいて 1 段まで，絶対アクセスにおいては使用出来ない．

3.4 特殊関数

各ワーキングセットにおいて，境界条件などの例外処理，およびインデックス情報などを得るために使用する関数群となる．

1. **アドレス変換関数：**
   各計算機が持っている分割データ中での *index* と，統合された全データにおける *index* の変換を行うための関数群となる．アドレスの絶対指定などの場合に利用する．

(a) int l2g_x(*VIOS* 変数, *num*)

VIOS 変数x 軸における，ローカル *index num* を，全体データでの *index* に変換する（local to global）.

(b) int l2g_y(*VIOS* 変数, *num*)

上記関数の y 軸版．vsImage, vsCube 変数のみ利用可能.

(c) int l2g_z(*VIOS* 変数, *num*)

上記関数の z 軸版．vsCube 変数のみ利用可能.

(d) int g2l_x(*VIOS* 変数, *num*)

VIOS 変数x 軸における，グローバル *index num* を，ローカルデータでの *index* に変換する（global to local）.

(e) int g2l_y(*VIOS* 変数, *num*)

上記関数の y 軸版．vsImage, vsCube 変数のみ利用可能.

(f) int g2l_z(*VIOS* 変数, *num*)

上記関数の z 軸版．vsCube 変数のみ利用可能.

2. **VIOS 変数からの情報取得関数：**

VIOS 変数における，幅や高さなどの全体の情報を得るための関数群.

(a) int orig_width(*VIOS* 変数)

VIOS 変数全体における，x 軸方向のローカル最大値を得る.

(b) int orig_heigh(*VIOS* 変数)

VIOS 変数全体における，y 軸方向の最大値を得る.

(c) int orig_depth(*VIOS* 変数)

VIOS 変数全体における，z 軸方向の最大値を得る.

(d) int local_width(*VIOS* 変数)

ローカル計算機が保有する，*VIOS* 変数x 軸方向の最大値を得る（ただしキャッシュ領域を含む).

(e) int local_heigh(*VIOS* 変数)

ローカル計算機が保有する，*VIOS* 変数y 軸方向の最大値を得る（ただしキャッシュ領域を含む).

(f) int local_depth(*VIOS* 変数)

ローカル計算機が保有する，*VIOS* 変数z 軸方向の最大値を得る（ただしキャッシュ領域を含む).

(g) int local_min_x(*VIOS* 変数)

ローカル計算機が保有する，*VIOS* 変数に対して，“**処理を行う**” ワーキングセットのローカル最小 x 値を得る.

(h) int local_max_x(*VIOS* 変数)

ローカル計算機が保有する，*VIOS* 変数に対して，“**処理を行う**” ワーキングセットのローカル最大 x 値を得る．

(i) int local_min_y(*VIOS* 変数)

ローカル計算機が保有する，*VIOS* 変数に対して，“**処理を行う**” ワーキングセットのローカル最小 y 値を得る．

(j) int local_max_y(*VIOS* 変数)

ローカル計算機が保有する，*VIOS* 変数に対して，“**処理を行う**” ワーキングセットのローカル最大 y 値を得る．

(k) int local_min_z(*VIOS* 変数)

ローカル計算機が保有する，*VIOS* 変数に対して，“**処理を行う**” ワーキングセットのローカル最小 z 値を得る．

(l) int local_max_z(*VIOS* 変数)

ローカル計算機が保有する，*VIOS* 変数に対して，“**処理を行う**” ワーキングセットのローカル最大 z 値を得る．

3. **VIOS 変数のコピー：**
型全体のデータを**同型式，同サイズ**の変数へコピーするための関数（ただし，キャッシュ領域のデータも上書きされる）．

(a) void ValCopy(*VIOS* 変数 *1*,*VIOS* 変数 *2*)

vsvalue 型 (ivalue, fvalue) の変数 2 から変数 1 へ全データ領域をコピーする．

(b) void ImgCopy(*VIOS* 変数 *1*,*VIOS* 変数 *2*)

vsImage 型 (iImage, fImage) の変数 2 から変数 1 へ全データ領域をコピーする．

(c) void CubeCopy(*VIOS* 変数 *1*,*VIOS* 変数 *2*)

vsCube 型 (iCube, fCube) の変数 2 から変数 1 へ全データ領域をコピーする．

4. **全体での処理や通信を制御する関数：**
特定ワーキングセットに対する処理の抑制や，キャッシュ領域の更新などを行う関数.

(a) void vsSetBoundary1D(*num*)

1 次元 *parallel* 構文により並列処理を行う際，全体データでの外周 *num* をその処理対象から外します.

(b) void vsSetBoundary2D(*num*)

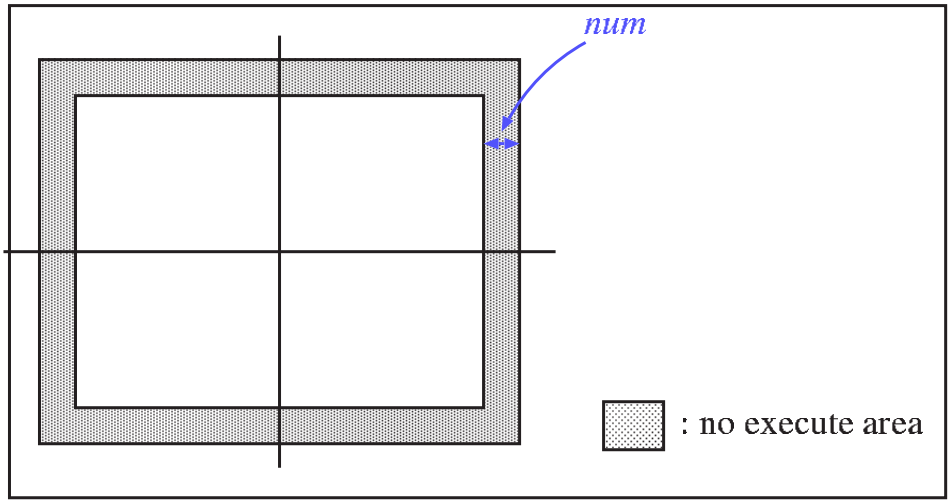
上記関数の 2 次元版

(c) void vsSetBoundary3D(*num*)

上記関数の 3 次元版

example:

上記図は，2 次元変数を 4 分割した場合のイメージしている．本来，各計算機が担当する処



理領域は，図中心の十文字で区切られた各領域全てであるが，vsSetBoundary により，それぞれ，L 字状に処理を行わない領域が出来ている．

(d) void vsSync()

全計算機間において同期が取られる．

(e) void vsSyncCache(*VIOS* 変数)

VIOS 変数におけるキャッシュ領域を最新のものに更新する．またこの際，全計算機間において同期が取られる．

(f) vsUnion(*VIOS* 変数, *TYPE*)

指定した *TYPE* により *VIOS* 変数の統合を行う．詳しくはチュートリアル → モジュール特殊関数 を参照．

なお，*VIOS* 変数は分割タイプとして ALL が指定されていることを前提とする．

(g) vsVote(*VIOS* 変数, int pos, value)

VIOS 変数の index pos に値 val を代入する（ただし、VIOS 変数は 1 次元, かつ分割タイプは ALL である必要あり）。 また，vsUnion(VIOS 変数,vsAGGRIGATE) を行うまで値 val の反映はない．

5. **user_comm 命令：**

ワーキングセット間で値の交換を行うための関数．

(a) comm_in(相対アドレス 1, ...， &入力変数)

自身から相対アドレス指定されるワーキングセットよりデータを入力変数に受信する．また処理はデータの入力があるまでブロックされる．

e.x） comm_in(-1, 0, &value); 相対座標 (-1, 0) のワーキングセットから値を入力し，変数 value へ代入する．

(b) comm_out(相対アドレス 1, ...， val)

自身から相対アドレス指定されるワーキングセットへデータ val を送信する．

なおこれら命令の使用例は，サンプル dp_matching を参照すること．

4 制限

4.1 予約語

並列処理部は，外部モジュールとして起動されるため，システム側で利用しているいくつかの変数名，定義語の影響を引き継ぐ．そのため *module* 内において同名の単語を利用することにより，思わぬ誤動作を招く恐れがある．

よって，以下に上げる名前は使用を禁止する．

- 上記拡張変数，および追加関数名
- vios_define.h において定義されている名前
- _vs，および vs で始まる名前
- “module_tab”（グローバル管理クラス名として使用）

4.2 実装上の制限

- {，および } について
  VPE-p では，その性質上，通常の C++プリプロセッサの処理前に処理を行う．また処理速度向上のため厳密な構文解析を行っていない．そのため，{，} の対応付けを崩すような#define 定義を利用することは出来ない．また特に，parallel_ie 構文中においては条件式における {，} の省略も禁止される．
- *module* の引数の数は 256 まで．
- *parallel_ie* を利用する場合は特に，*parallel_ie* 構文内が 65536 文字までに制限される．
- C 言語による記述部に関して
  基本的に MT-Safe な関数ならば利用可能．ただしヘッダをインクルードする必要と，vs_makemodule スクリプトに，ライブラリをリンクするよう指示する必要はある．