PIKG仕様書

Ver 0.1c

野村昴太郎

August 21, 2020

Contents

1 TODO

- FXXmat関連の演算や型推論が実装されていない
- 現状勝手にアップキャストされてしまうが，低精度を維持したい場合，statementの先頭に型宣言ができる必要がある
- 16 bit型の演算をサポートする

2 履歴

2.1 2019/12/23

ファイルを作った.

2.2 2019/12/30

カーネルジェネレータ周りのTODOを追加.

2.3 2020/5/7

カーネルジェネレータの名前をPIKGに変更.

2.4 2020/7/31

0.0版リリース向けに改定

2.5 2020/8/7

変数宣言部に記述を追加.

2.6 2020/8/12

条件分岐の文法に間違いがあったので修正.

2.7 2020/8/14

C-interfaceの記述を追加.

2.8 2020/8/16

Fortran-interfaceの記述を追加.

2.9 2020/8/20

変数宣言部および相互作用記述部の新文法に関するを追加.

3 はじめに

本書は粒子間相互作用カーネル関数のソースコードジェネレータPIKGの仕様書である．

粒子間相互作用とは，分子動力学やN体，SPHなどの粒子系シミュレーションにおいて，粒子の時間発展を差分化して解く際に計算する粒子間に働く相互作用のことである．ここでは，2体間相互作用のみを考える．

粒子系シミュレーションでは，一般的に計算量のほとんどをこの粒子間相互作用の計算が占めている．したがって，多様化しているさまざまな計算機上で十分な実行性能を引き出すためには，それぞれのアーキテクチャ上で相互作用を計算するカーネルの最適化が必要不可欠である．しかしながら，これらの最適化カーネルは特定のアーキテクチャに特化した形で書かれるものが多く，計算機を変える場合には新たにカーネルを書き直すことがほとんどである．さまざまな計算機を利用する研究者にとって，それぞれのアーキテクチャ向けにカーネルを最適化することは，多くの時間を要するだけでなく，複雑なアーキテクチャの特性を理解しないといけないためそもそも最適化が困難であるという問題がある．

相互作用計算カーネルジェネレータPIKGは，ひとつのDSL(ドメイン特化型言語)記述から，複数のアーキテクチャ向けに最適化された相互作用計算カーネルを生成することを目的としている．

4 概要

カーネル関数の最適化ソースコードを様々なアーキテクチャ向けに生成するためのジェネレータを作る．そのための言語仕様及びコンパイラの仕様，簡単な使い方を示す．

5 DSL仕様

PIKGは，PIKG用DSL(Domain Specific Language)で書かれたソースコードを元に，指定したアーキテクチャ向けに最適化されたC++のカーネルソースコードを生成する．

以下では本カーネルジェネレータで用いるDSLの仕様について定義する．

5.1 粒子間相互作用カーネルの一般化

まず，DSL記述について述べる前に，粒子間相互作用カーネルが一般にどのような構造とをとるのかを説明する． C/C++の関数では2粒子間相互作用カーネルは以下のように書ける．

```
void kernel_body(const EPI* epi,const int ni,
                 const EPJ* epj,const int nj,
                 FORCE* force){
  (1) preprocess
  for (int i=0; i < ni; i++){
    (2) load EPI variables
    (3) load FORCE variables
    for(int j=0; j < nj; j++){
      (4) load EPJ variables
      (5) calc interaction for FORCE variable
    }
    (6) store FORCE variables
  }
}
```

入力として，相互作用をうける粒子であるEPIと相互作用を及ぼす粒子であるEPJのArray of Structure(AoS)形式の配列epiとepj(これらをあわせて相互作用リストと呼ぶ)，それぞれの配列長niとnj，計算された相互作用を保持する構造体であるFORCEのAoS形式の配列が与えられる(FORCEの配列の長さはni)．

(1)ではEPIやEPJの変数を一時配列に入れ直し，AoSからSoA形式にする等の前処理が行われる． (2)ではEPI変数及び(1)で作った一時配列のうち，カーネル計算に必要な変数のみがロードされる． (3)では相互

作用のアキュムレートに必要なFORCE変数をロードする．FORCE変数は事前に初期化されていることが前提となっている．これは，複数のカーネルを使ってFORCEを計算する場合を考慮しているためである．(4)ではEPJ変数をロードする．(2)と同様である． (5)ではロードした変数を使って相互作用の計算を行い，FORCE変数にアキュムレートしていく．最後に(6)でアキュムレートしたFORCE変数をもとの配列にストアする．

PIKGでは，DSL記述から(5)相互作用計算と(5)で使われる変数がEPI，EPJとFORCEクラスのどの変数かの情報を読み取り，相互作用計算カーネルを生成する．ここで，EPI，EPJ，FORCEは同じ型でもよく，相互作用カーネルに渡される3つの配列には同じ配列を使用しても良い．

5.2 DSL概要

PIKGのDSLは以下の3つのパートから構成される．

- 変数宣言部
- 関数宣言部
- 相互作用記述部

それぞれのパートは上記の順番で書かれる必要があり，変数宣言部は0個以上の変数宣言からなり，関数宣言も0個以上の関数宣言からなる．相互作用記述部では，変数宣言部と関数宣言部で宣言された変数と関数，予約関数を用いて粒子間の相互作用を記述する．

以下では，DSLの文法とそれぞれのパートについて詳しく見ていく．

5.3 文法等

5.3.1 コメントアウト

二重のスラッシュ(//)以降のその行の文字列はコメントとして無視される．C++における/* */のような行をまたげるコメントアウトは存在しない．

5.3.2 予約語とその説明

- EPI (変数宣言時に相互作用を受ける粒子の変数であることを示す)
- EPJ (変数宣言時に相互作用を及ぼす粒子の変数であることを示す)
- SPJ (変数宣言時に相互作用を及ぼす粒子でTree法で見込角より外のまとめられた粒子でであることを示す，MultiWalkのときのみ使用)
- FORCE (変数宣言時に求める相互作用を保持する変数であることを示す)
- local (変数宣言時にlocalメンバ変数であることを示す)
- function (関数宣言であることを示す)
- (S|U)(16|32|64) ((符号あり|なし)整数スカラー型)
- F(16|32|64)([vec|mat][2|3|4]) (浮動小数点スカラー，ベクトル，マトリックス型．末尾の数字は次元．何もつけない場合は3次元として扱う)
- (sqrt|rsqrt|inv|min|max|table|to_float|to_int|to_uint|madd|msub|nmadd|nmsub) (特殊関数．それぞれ平方根，逆数平方根，逆数，最小値，最大値,テーブル関数，積和算)

5.3.3 利用可能な型

本DSLでは，符号あり/なし整数型および浮動小数点スカラー型，ベクトル型，マトリックス型をサポートする．具体的には，以下の型が使用可能である．(マトリックス型は対称行列のみだが現状サポート外).

- (S|U)(16|32|64)
- F(16|32|64)
- F(16|32|64)vec(2|3|4)?
- F(16|32|64)mat(2|3|4)?

それぞれ，16 bit，32 bit，64 bitの(符号あり｜なし)整数型，浮動小数点スカラー型，浮動小数点ベクトル型，浮動小数点マトリックス型である．浮動小数点ベクトル型とマトリックス型は末尾に2もしくは3，4をつけることにより，次元を表すことができる．省略した場合は3次元である．

仕様に定めているが現状未サポートの機能は以下.

- 現状16 bit型の変数
- すべてのマトリックス型

5.3.4 演算子

使用可能な演算子は以下である.

- + (加算)
- − (減算)
- ∗ (乗算，ベクトル型は内積)
- / (除算)
- && (論理積)
- || (論理和)
- == (等価)
- ! = (不等価)
- < (比較)
- > (比較)
- <= (比較)
- >= (比較)
- + = (加算代入)
- − = (減算代入)
- ∗ = (乗算代入)
- / = (除算代入)

詳細は相互作用関数記述部5.6節で説明する.

5.3.5 条件分岐

条件分岐は以下の文法でおこなえる.

```
if 条件1
  実行1
elsif 条件2
  実行2
else
  実行3
endif
```

elsif以下は省略可能である．将来的にendifはendでも使えるようになる予定(Rubyの文法に合わせる)．後置などはできない.

例

```
dr = ri - rj
r2 = dr*dr
if r2 > 0.0 && r2 < 16.0
  r2i = inv(r2)
  r6i = r2i * r2i * r2i
  f += 24 * r6i * r2i * (2*r6i - 1) * dr
  u += 4 * r6i * (r6i - 1)
endif
```

5.3.6 予約関数

予約されている関数は以下.

- sqrt (平方乗根)
- rsqrt (逆数平方根)
- max (最大値)
- min (最小値)
- madd ($A * B + C$)
- msub ($A * B - C$)
- nmadd ($-A * B - C$)
- nmsub ($-A * B + C$)
- table (テーブル命令)
- to_(float|int|uint) (キャスト)

積和算系の命令に関しては，DSL中で使わなくても，自動的に縮約される.

5.4 変数宣言部

変数宣言では，型の前にEPI，EPJもしくはFORCEをつけることでそれぞれのメンバ変数であることを示す．また，変数宣言では変数名のあとにコロン(:)と実際のEPIもしくはEPJ，FORCE型のメンバ変数名を書かなくてはならない．これにより，変数宣言で宣言された変数名と実際の型のメンバ変数の対応関係を示す.

```
EPI F32vec ri:r
EPJ F32vec rj:r
EPJ F32    mj:m
FORCE F32 fi:f
FORCE F32 ui:u
```

これは仕様ではないが，現状”x”,”y”,”z”,”w”は変数名として使用することができない．

EPI，EPJもしくはFORCEがつかない場合，カーネル関数オブジェクト初期化時に渡す引数として扱われる．DSL上ではグローバル変数のような扱いになる．この場合，コロンによって対応関係を示す必要はない．

```
F32 eps
```

また，local修飾子をつけることでEPI，EPJのローカル変数を宣言できる．ローカル変数にEPIやEPJの変数を使った値を代入する文を記述すると，相互作用計算の2重ループの外側で初期化され，EPI変数やEPJ変数と同じように相互作用計算に利用できる．

後述の–class-fileオプションを付けない場合，EPI，EPJ，FORCEの変数宣言はC++のクラスの宣言と型やメンバ変数の宣言の順番，個数と完全に一致していなくてはならない．したがって，このとき実際には相互作用の計算に使用されないメンバ変数がクラスの定義に含まれている場合でも，変数宣言部では宣言される必要がある．

5.5 関数記述部

本DSLでは，関数を記述し，相互作用関数記述部で利用することができる．関数記述部は，以下の記述になる．

```
function 関数名(引数0, ...)
  /* 計算部分 */
  return 返り値
end
```

関数記述内の計算部分では，予約語および変数宣言部にない任意のローカル変数を使用できる．型推論されるので，特に宣言なく使用できる．

5.6 相互作用関数記述部

相互作用関数記述部では，EPI，EPJ，FORCEクラスの変数および一時変数を使ってFORCEクラスの変数に相互作用計算で求めたい値をアキュムレートしていくように記述を行う．

後述の–class-fileオプションを使う場合は，変数宣言部で宣言した変数名以外に，実際のEPI，EPJ，FORCEクラスのメンバ変数名を使ってEPI.posやEPJ.massのようにメンバ変数を使うことができる．下記の例では，EPJ.massとmjは等価である．

```
EPJ F64 mj:mass
F64 eps2

dr = EPI.pos - EPJ.pos
r2 = dr * dr * eps2
rinv = rsqrt(r2)
mr_inv = mj * rinv
mr3_inv = mr_inv * rinv * rinv
FORCE.acc -= mr3_inv * dr
FORCE.pot -= mr_inv
```

また，EPI.とEPJ.とFORCE.を省略した記法も可能である．ただし，EPIとEPJとFORCEクラスで同じ名前のメンバ変数がある場合，省略したときは，左辺値ではFORCE.が，右辺値ではEPI. > EPJ. > FORCE. の

順番の優先度で省略されたとみなされる．ただし，同じ文において，左辺値でFORCE.が省略された変数があり，右辺値でも同じ変数が出てくる場合に限り，優先度を無視してFORCE.が省略されたとみなす．

たとえば，以下のようなC++でのクラスの定義があるとすると，

```
struct EPI{
  F64vec pos;
};
struct EPJ{
  F64vec pos;
  F64v   mass;
};
struct EPI{
  F64vec acc;
  F64    pot;
};
```

前述の相互作用記述部は以下のようにも記述できる．

```
dr = pos - EPJ.pos
r2 = dr * dr * eps2
rinv = rsqrt(r2)
mr_inv = mass * rinv
mr3_inv = mr_inv * rinv * rinv
acc -= mr3_inv * dr
pot -= mr_inv
```

また，以下のようなクラスParticleがEPI，EPJ，FORCEに使われているとする．

```
struct Particle{
  F64vec pos;
  F64    mass;
  F64vec acc;
  F64    pot;
};
```

このとき，

```
acc = acc - mr3_inv * dr
```

という記述は

```
FORCE.acc = FORCE.acc - mr3_inv * dr
```

と解釈される．

–class-fileオプションを用いない場合，変数宣言部で宣言した変数と一時変数のみを相互作用記述部で使うことができる．

```
EPI F64vec xi:pos

EPJ F64vec xj:pos
EPJ F64    mj:mass

FORCE F64vec acc:acc
FORCE F64    pot:pot

dr = xi - xj
```

```
r2 = dr * dr * eps2
rinv = rsqrt(r2)
mr_inv = mj * rinv
mr3_inv = mr_inv * rinv * rinv
acc -= mr3_inv * dr
pot -= mr_inv
```

相互作用関数記述部では，これまでに型を指定して宣言された変数をもとに型推論が行われるため，ローカルな変数の宣言などを行う必要がなく，これまでに宣言されていない好きな変数名の変数に値を代入できる．

カーネルジェネレータは，EPIとEPJの2重ループを作り，その中でこれらの演算子を用いて書かれた記述をアキュムレートするカーネル関数を生成する．

5.6.1 プラグマ記述

主にARM SVE向けの機能であるが，プラグマをはさみたい場所に#pragmaから始まる行を入れると，生成されたカーネルの対応する箇所に同じプラグマ文が入る．以下のプラグマは，実際に生成されるカーネルにも影響を与える．

- #pragma unroll X (Xはアンロール段数)
- #pragma statement loop_fission_point

6 カーネル生成方法及び利用法

カーネル生成は以下の2ステップで行われる．

- 相互作用関数をDSLで記述
- オプションを指定してカーネルジェネレータにDSLファイルを入力として渡す

カーネルジェネレータの実行方法は以下．$(PIKG)はPIKGをgit cloneしてきたディレクトリである．

```
$(PIKG)/bin/pikg [options]
```

指定可能なオプションは以下の通りである．
必須オプション:

- −−input | −i ファイル名 (入力ファイル名の指定)

省略可能オプション:

- −−output | −o ファイル名 (出力ファイル名の指定．ユーザープログラムにインクルードされることを想定する．単独でコンパイル可能にするためには kernel.cpp のように C++ プログラムの拡張子にする．デフォルトはkernel.hpp)
- −−conversion−type (AVX2|AVX-512|SVE) (アーキテクチャの指定．デフォルトはreference)
- −−kernel−name カーネル関数名 (カーネル関数名の指定．デフォルトはKernel)
- −−epi−name EPI名 (EPIの構造体名の指定．デフォルトはEPI)
- −−epj−name EPJ名 (EPJの構造体名の指定．デフォルトはEPJ)
- −−force−name FORCE名 (FORCEの構造体名の指定，デフォルトはFORCE)

- −−class−file ファイル名0 [ファイル名1 ファイル名2] (EPI/EPJ/FORCEの構造体の宣言がされているファイル名の指定．ファイル名0だけ指定した場合にはEPI，EPJ，FORCEが全て同じファイル名に宣言されていて，ファイル名3つ指定した場合にはファイル名0にEPI，ファイル名1がEPJ,ファイル名2にFORCEが宣言されているとみなす)
- −−strip−mining ストリップマイニングループ数 (アーキテクチャがSVEに指定されている場合にストリップマイニングを行う)
- −−unroll アンロール段数 (−−strip−mining指定時に最内ループをアンロールする段数を指定，デフォルトは1．DSL内にunrollのプラグマが現れるとそれ以降はプラグまで指定されたアンロール段数が使われる)
- −−multiwalk (FDPSにおけるMultiwalk向けにカーネルを生成する場合に指定，現状利用不可)
- −−spj−name SPJ名 (SPJの構造体名の指定．−−multiwalkを指定したときのみ利用される．デフォルトはSPJ)
- −−addtional−text 文字列 (出力ファイルのデフォルトのインクルードファイル指定の直後に追加される文字列．ここで"#include "user_defined.h"" 等を指定することで単独でコンパイル可能なプログラムファイルを生成できる)
- −−c−interface [ファイル名] (ファイル名は省略可能．このオプションがあるとC言語リンケージのための関数実体と関数プロトタイプ宣言を生成する．ファイル名が指定されたらプロトタイプ宣言はそのファイルに出力される．指定がない時にはファイル名は –output もしくは -o で指定されたファイル名の basename (最後の "." の後ろを取り除いたもの)+".h"となる)
- −−fortran−interface module名 (このオプションがあると−−c−interfaceを有効化するとともに，相互作用関数を呼び出すために必要なmoduleを指定したmodule名で生成する．生成されたmoduleはmodule名+".F90"に出力される)
- −−initializer−name イニシャライザ名 (–c-interfaceオプションが有効なときに，C言語において相互作用カーネルのファンクタのメンバ関数を初期化するための関数名を指定．デフォルトは–kernel-nameで指定されるカーネル名+"_initalize")

−−strip−mining及び−−unrollは将来的に廃止される予定であるが，現状ARM SVE向けの最適化には指定が必要なので公開する．現状，最適なARM SVEカーネルを生成するためには，DSL内の適切な箇所にunroll及び，loop_fission_pointのプラグマを挿入する必要がある．

−−class−fileオプションを使用する際，PIKGはEPIクラスやEPJクラスの定義がそのファイルの中で一つだけであることを仮定する．つまり，マクロなどでクラスの定義を切り替える等を想定していない．またテンプレートクラスにおいて，メンバ変数の型名がテンプレートパラメータで与えられていることも想定しない．また，usingなどを使って型名をエイリアスされていることも想定しない．つまり，クラス定義であたえられるクラス名が−−epi−nameなどのオプションで与えられるクラス名と一致している必要がある．EPI，EPJ，FORCEクラスの宣言においてメンバ変数の宣言で型名として使用可能なものを挙げる．()内はPIKG上での型名．

- double (F64)
- float64_t (F64)
- (PS::—PIKG::)F64 (F64)
- float (F32)
- float32_t (F32)
- (PS::—PIKG::)F32 (F32)
- half (F16)

- float16_t (F32)
- (PS::—PIKG::)F16 (F16)
- long long int (S64)
- long long (S64)
- int64_t (S64)
- (PS::—PIKG::)S64 (S64)
- int (S32)
- int32_t (S32)
- (PS::—PIKG::)S32 (S32)
- unsigned long long int (U64)
- unsigned long long (U64)
- uint64_t (U64)
- (PS::—PIKG::)U64 (U64)
- unsigned int (U32)
- unsigned (U32)
- uint32_t (U32)
- (PS::—PIKG::)U32 (U32)
- (PS::—PIKG::)F64vec (F64vec)
- (PS::—PIKG::)F64vec2 (F64vec2)
- (PS::—PIKG::)F64vec3 (F64vec3)
- (PS::—PIKG::)F64vec4 (F64vec4)
- (PS::—PIKG::)F32vec (F32vec)
- (PS::—PIKG::)F32vec2 (F32vec2)
- (PS::—PIKG::)F32vec3 (F32vec3)
- (PS::—PIKG::)F32vec4 (F32vec4)
- (PS::—PIKG::)F64vec (F64vec)
- (PS::—PIKG::)F64vec2 (F64vec2)
- (PS::—PIKG::)F64vec3 (F64vec3)
- (PS::—PIKG::)F64vec4 (F64vec4)
- (PS::—PIKG::)F32vec (F32vec)
- (PS::—PIKG::)F32vec2 (F32vec2)
- (PS::—PIKG::)F32vec3 (F32vec3)
- (PS::—PIKG::)F32vec4 (F32vec4)

using namespaceなどを使ってPS::もしくはPIKG::を省略することは可能.

6.1 生成されるカーネルの形式

カーネルは以下のような構造体として生成される(いずれもデフォルトのEPI,EPJ,FORCE，カーネル名を使う場合).

```
struct Kernel{
  void operator()(const EPI* epi, const int nepi, const EPJ* epj,const int nepj,FORCE* force){
    (カーネル本体)
  }
};
```

生成したカーネルは生成されたファイル(ここではkernel.hpp)をインクルードし，

```
#include "kernel.hpp"
const int N = 1024; // 粒子数
int main{
  const int nepi = N;
  const int nepj = N;
  EPI *epi = new EPI[nepi];
  EPJ *epj = new EPJ[nepj];
  FORCE *force = new FORCE[nepi];
  Kernel kernel; // ファンクタのインスタンス化．メンバ変数の初期化が必要であればここで行う
  (epiやepjに粒子の情報を代入)
  kernel(epi,nepi,epj,nepj,force); // カーネル呼び出し
  (後略)
}
```

のようにして利用できる.

6.2 Cインターフェース

PIKGはC++のstructを利用して，相互作用カーネルのファンクタを生成する．当然このままではC言語およびその他の言語のユーザからはPIKGから生成されたカーネルを利用できないので， Cインターフェースオプションを提供する． −−c−interfaceオプションを有効化すると，生成されたコードの中でファンクタの実体を作り， −−kernel-nameで指定した名前の関数からファンクタを呼び出す関数が生成される．また，各関数のプロトタイプ宣言も別ファイルに生成される(ファイル名の指定などは−−c-interfaceオプションを参照). C言語やその他の言語で書かれたコードからは−−kernel-nameで指定した関数に適切な引数をつけて呼び出せば，相互作用計算が行われる.

生成されたコードが単独のファイルとしてコンパイルされるためにはEPI/EPJ/FORCEクラスもしくは構造体の実体をインクルードされる必要がある．PIKGからはその実体がどのヘッダーファイルにかかれているのかを把握するすべがないため，−−additional-textオプションをもちいて適切にファイルをインクルードする必要がある．詳しくは−−additional-textオプションを参考のこと.

また，EPI/EPJ/FORCE以外の変数を宣言している場合は，ファンクタのメンバ変数を初期化する必要があるのでこれを行う関数も提供される．詳しくは−−initializer-nameオプションを参照のこと.

–class-fileオプションを使用したとき，PIKGは以下のように構造体名が定義されていることを仮定する.

```
typedef struct{
  型名0 メンバ変数名0;
  型名1 メンバ変数名1;
          .
          .
          .
} 構造体名;
```

型名として使用可能なものを挙げる．()内はPIKG上での型名.

- double (F64)
- float (F32)
- long long int (S64)
- long long (S64)
- int (S32)
- unsigned long long int (U64)
- unsigned long long (U64)
- unsigned int (U32)
- unsigned (U32)
- pikg_f64vec (F64vec)
- pikg_f64vec2 (F64vec2)
- pikg_f64vec3 (F64vec3)
- pikg_f64vec4 (F64vec4)
- pikg_f32vec (F32vec)
- pikg_f32vec2 (F32vec2)
- pikg_f32vec3 (F32vec3)
- pikg_f32vec4 (F32vec4)
- fdps_f64vec (F64vec)
- fdps_f64vec2 (F64vec2)
- fdps_f64vec3 (F64vec3)
- fdps_f64vec4 (F64vec4)
- fdps_f32vec (F32vec)
- fdps_f32vec2 (F32vec2)
- fdps_f32vec3 (F32vec3)
- fdps_f32vec4 (F32vec4)

6.3 Fortranインターフェース

PIKGではCインターフェースと同様にFortranインターフェースを提供する． FortranインターフェースではCインターフェースの機能に加えて，Fortranで利用するmoduleを含むFortranソースを生成する必要がある．module名は−−fortran-interfaceに続けてmodule名を指定する．ユーザコードからは，このmoduleを利用して−−kernel-nameで指定した関数に適切な引数をつけて呼び出すことで相互作用計算が行われる．PIKGで定義されているベクトル型を利用する場合は，src/fortran_interface/modules/pikg_vector.F90に定義されているpikg_vectorを利用する必要がある．

Fortranインターフェースを使う際の注意点として，Fortranには符号なし整数が無いため，EPI，EPJ，FORCE型のメンバ変数としてU(16|32|64)は使えない．

–class-fileオプションを使用したとき，PIKGは以下のように構造体名が定義されていることを仮定する．

```
type 構造体名
  型名宣言0 [::] メンバ変数名0
  型名宣言1 [::] メンバ変数名1
          .
          .
          .
end type 構造体名
```

型名宣言として使用可能なものを挙げる．最後の()内はPIKG上での型名.

- real(kind=c_double) (F64)
- real(kind=c_float) (F32)
- integer(kind=c_long_long) (S64)
- integer(kind=c_int) (S32)
- type(pikg_f64vec) (F64vec)
- type(pikg_f64vec2) (F64vec2)
- type(pikg_f64vec3) (F64vec3)
- type(pikg_f64vec4) (F64vec4)
- type(pikg_f32vec) (F32vec)
- type(pikg_f32vec2) (F32vec2)
- type(pikg_f32vec3) (F32vec3)
- type(pikg_f32vec4) (F32vec4)
- type(fdps_f64vec) (F64vec)
- type(fdps_f64vec2) (F64vec2)
- type(fdps_f64vec3) (F64vec3)
- type(fdps_f64vec4) (F64vec4)
- type(fdps_f32vec) (F32vec)
- type(fdps_f32vec2) (F32vec2)
- type(fdps_f32vec3) (F32vec3)
- type(fdps_f32vec4) (F32vec4)

::は省略可能．メンバ変数の宣言のあとにFDPSのFortranインターフェースのマクロをつけることは可能である.