# Lagrangian Particle Tracking Library

**Design & Implementation**

**Ver. 0.9**

**Advanced Visualization Research Team**
**Advanced Institute for Computational Science**
**RIKEN**

7-1-26, Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo, 650-0047, Japan

http://aics.riken.jp/

June 2013

**Release**

| | | |
|---|---|---|
| Version 0.9 | 15 June | 2013 |
| Version 0.8 | 13 June | 2013 |
| Version 0.7 | 10 June | 2013 |

# 目次

# 第 1 章

# イントロダクション

本章では，大規模な粒子の並列粒子追跡計算ライブラリについて概要を示す．

## 1.1　本ドキュメントの目的

本ドキュメントは粒子追跡機能を提供する LPT ライブラリの内部構造の説明を目的としている．

LPT を使用した粒子-流体連成計算を行うために必要な情報はユーザガイドにまとめられているのでライブラリのコンパイル方法やソルバへの組込み方法等はユーザガイドを参照のこと．

## 1.2　LPT のクラス構成

LPT のクラスは LPT, PPlib, DSlib という 3 つの namespace に分かれている．各 namespace に含まれる主なクラスを表 1.1 および図 1.1 に示す．各クラスの詳細についてはリファレンスマニュアルを参照のこと．

表 1.1　LPT ライブラリに含まれる主なクラス

| namaspace | class | 機能 |
|---|---|---|
| LPT | LPT | 粒子計算を行うためのインターフェースルーチンを提供するクラス．Singleton パターンを適用. |
| | FileOutput | 粒子データの出力を行うための抽象クラス |
| | LPT_ParticleInput | 粒子データの読み込みを行うためのクラス．ライブラリ本体では (現時点では) 使っておらず，ファイル変換プログラム内でのみ使われている |
| | FileManager | ファイル名および入/出力ストリームを生成，管理するクラス．Singleton パターンを適用. |
| PPlib | PPlib | 開始点，粒子データクラスを保持し，粒子の放出や移動を計算するクラス. |
| | Interpolator | 流体の格子点上の物理量から粒子位置での物理量へ補間するクラス. |
| | Integrator | 4 次のルンゲ=クッタ法により速度場の積分を行うクラス. |
| | ParticleData | 粒子データを保持するクラス. |
| | StartPoint | 開始点データを保持するクラス. |
| DSlib | DSlib | 流体データのキャッシュ機構を提供するクラス. |
| | Communicator | 流体データの送受信を行うクラス. |
| | DecompositionManager | 解析領域の座標と，領域分割に関する情報を保持するクラス. |

図 1.1　LPT のクラス構成

## 1.3　LPT の処理フロー概要

図 1.2 に呼び出し元の流体ソルバも含めた処理フローを示します．

LPT の処理は大きく 3 つの部分に分けることができます．1 つ目は流体ソルバの初期化部分から呼び出される初期化処理．2 つ目はタイムステップループ内から呼び出される粒子計算部分．最後の一つはリソースの解放等を行う後処理部分です．個々の具体的な処理内容は 3 章から 5 章に記載しています．

図 1.2　LPT の処理フロー

# 第 2 章

# データ構造

本章では，LPT ライブラリのデータ構造について説明する．

## 2.1　LPT が想定する流体ソルバのデータ構造

LPT ライブラリでは，流体ソルバが直交等間隔の格子を持ちセル中心に物理量を保持していることを想定しています．

また，この解析領域は各軸方向を分割したサブドメインに分けられ MPI の 1 プロセスが 1 サブドメインの計算を担当する形で，並列化されているものとします．分割の際に余りセルが生じる場合は，原点を含むサブドメインに近い側のサブドメインのサイズを 1 セル多く担当させるものとします．さらに，個々のサブドメインには全方向に共通のサイズの袖領域を持つものと想定しています．

これらの解析領域に関する情報は LPT_Initialize の引数構造体 LPT_InitializeArgs を通して DecompositionManager クラスに渡され，このクラスのメンバ変数として必要な情報を保持しています．

解析領域全体に関する情報は，原点座標と x,y,z 各方向のセル数およびセル幅を図 2.1 に示すメンバ変数に保持しています．

図 2.1　解析領域全体の定義

領域分割に関する情報は，x,y,z 各方向の分割数を図 2.2 に示すメンバ変数に保持しています.

各サブドメインの計算を担当する Rank は，Rank0 が原点を含むサブドメインを担当し順に，x,y,z の各方向にブロック分割で割り当てるものとします.

各サブドメインと担当 Rank の関係は図 2.2 をご覧ください.

図 2.2　サブドメイン

LPT ライブラリ内での流体データの管理は，複数のセルをまとめた”ブロック”単位で行います．ブロックのサイズは，1 サブドメイン内に含まれるブロック数 (x,y,z 各軸方向毎に個別に指定) で規定され余りが無ければ，1 ブロックのサイズは (Nx/NPx/NBx)*(Ny/NPy/NBy)*(Nz/NPz/NBz) となります．余りのセルが発生する場合は，それぞれ原点に近い側のサブドメインまたはブロックに 1 セル余分に割り当てています．

DecompositionManager クラス内には，解析領域内の個々の部分がどのサブドメインまたはブロックに属しているかを判定するためにサブドメインおよびブロックの境界となるセルインデックスをそれぞれメンバ変数の SubDomainBoundary{X,Y,Z} および BlockBoundary{X,Y,Z} に保持しています．Nx=10,NPx=3,NBx=2 とした時に SubDomainBoundaryX および BlockBoundaryX に保持される値は図 2.3 のようになります．

図 2.3　Nx=10,NPx=3,NBx=2 とした時の SubDomainBoundaryX および BlockBoundaryX の値

## 2.2　粒子データクラスの構成

粒子計算の対象となる粒子データは，PPlib::ParticleData 構造体のオブジェクトとして保持されています．このオブジェクトが持つメンバ変数を表 2.1 に示します．

表 2.1　ParticleData 構造体のメンバ変数

| 型 | 変数名 | 値の意味 |
|---|---|---|
| int | StartPointID[2] | この粒子を放出した開始点の ID. |
| int | ParticleID | この粒子の ID．放出された順に昇順で付けられる |
| REAL_TYPE | Coord[3] | 粒子座標 |
| REAL_TYPE | ParticleVelocity[3] | 粒子速度 |
| double | StartTime | 粒子が放出された時刻 |
| double | LifeTime | 粒子の寿命 |
| long | BlockID | この粒子を含むブロックの ID |
| double | CurrentTime | どの時刻の粒子情報かを示す変数 |

生成された粒子データオブジェクトへのポインタは PPlib クラスのメンバ変数 Particles に登録して管理されており，Particles は ParticleData クラスへのポインタを保持する std::list として実装されています．

## 2.3　開始点クラスの構成

粒子を放出する開始点のデータは，個々の点毎にオブジェクトとして保持するとメモリ使用量が大きすぎるため，一定の領域毎にまとめて 1 つのオブジェクトとしています．

領域の形状によって，以下の 6 種類のクラスが用意されています．なお，StartPoint クラスは 1 点からなる領域のオブジェクトとして使われるとともに他のクラスの基底クラスとしても使われています．

- StartPoint
- MovingPoints
- Line
- Rectangle
- Cuboid
- Circle

各オブジェクトが共通で持つメンバを表 2.2 に示します．

Coord1 は StartPoint クラスでは，粒子が発生する点の座標を示しますが，Line, Rectangle, Cuboid ではそれぞれの領域の一方の端点の座標を示します．これらのクラスでは派生クラス側でメンバ変数 Coord2 が用意されており，領域のもう一方の端点の座標をこちらに保持しています．

ID は 2 要素の配列となっており，第 1 要素には開始点を保持するプロセスの Rank 番号，第 2 要素にはそのプロセスが保持する開始点に一意につけられる ID 番号を保持しています．

各開始点オブジェクトへのポインタは，PPlib クラスのメンバ変数 StartPoints に登録して管理されています．StartPoints は StartPoint クラスへのポインタを保持する std::vector として実装されています．

## 2.4　流体相のデータ

前述のように，LPT ライブラリ内においては，流体データはブロック単位で管理されています．

表 2.2　StartPoint クラスのメンバ変数

| 型 | 変数名 | 値の意味 |
|---|---|---|
| REAL_TYPE | Coord1[3] | 領域を規定する座標の一点 |
| int | SumStartPoints | その領域に含まれる開始点の数 |
| double | StartTime | 粒子の放出を始める時刻 |
| double | ReleaseTime | 開始点のの寿命 |
| double | TimeSpan | 粒子の放出間隔 |
| double | LatestEmitTime | 直近で粒子を放出した時刻 |
| double | ParticleLifeTime | この開始点から放出される粒子の寿命 |
| int | ID[2] | この開始点の ID |

全 Rank が個々に自 Rank が流体ソルバ内で担当する領域のデータへのポインタを LPT_CalcParticleData() の引数構造体のメンバとして受け取り, DSlib::Communicator::CommF2P() およびそこから呼ばれる DSlib::Communicator::CommPacking() へ引数として渡します．CommPacking() 内部で，自 Rank を含む他の粒子計算プロセスから要求されたデータブロックに相当する流体データを，コピーし MPI 通信を用いて必要とするプロセスへ送信します．

流体ソルバから受けとったポインタはこの用途にのみ使われ，LPT 内部では情報を保持しません．

## 2.5　通信時のデータ構造

通信を行なう際には，データ本体以外に，そのブロックの ID や座標情報などのメタデータが必要になります．これらのメタデータは DSlib::CommDataBlockHeader 構造体に格納されてデータ本体を格納した配列へのポインタや，通信に使う MPI_Request 変数とともに DSlib::CommDataBlockManager 構造体に格納されて管理されます．CommDataBlockManager 構造体は，通信が完了された時点で CommDataBlockHeader 構造体は，受け取ったデータを後述の DSlib::DataBlock 構造体へコピーした時点で破棄されます．また，CommDataBlockHeader 構造体は，そのまま MPI_Type_Vector して，メタデータ通信の単位として使っています．粒子-流体通信のアルゴリズムや実装の詳細は**??** 章をご覧ください．

## 2.6　キャッシュ

通信により他 Rank(又は自分自身) から受けとったデータのうちメタデータは DSlib::DataBlock 構造体へ格納されます．また，データ本体は受信バッファの配列をそのまま利用し，領域へのポインタのみを DSlib::DataBlock 構造体へ格納します．

自 Rank が現在保持している流体データは”キャッシュ済ブロック”として，DSlib のメンバである CachedBlocks に登録されます．キャッシュのアルゴリズムや実装の詳細は**??**章をご覧ください．

# 第 3 章

# 初期化部分

本章では，LPT ライブラリの初期化部分について説明する．

## 3.1　初期化部分概要

LPT の初期化処理は，以下の 3 段階に分けることができます．

1. 開始点の設定
2. LPT_InitializeArgs 構造体の設定
3. LPT_Initialize() 内で行う処理

## 3.2　開始点の設定

開始点の設定は，LPT_SetStartPoint*を通じて，LPT のメンバ変数 StartPoint に格納されます．開始点設定用のインターフェースルーチンの使用方法は，ユーザガイドをご覧ください．

ここで設定された開始点は, LPT_Initialize() 内で PPlib のメンバ変数 StartPoints にコピーされ, PPlib::DistributeStartPoints() が呼び出されることで分割および各プロセスへのデータ分散が行われます．開始点の分割およびデータ分散アルゴリズムについては**??**章をご覧ください．

なお，計算の対象となる開始点は，PPlib のメンバとして保持されているものだけなので LPT_Initialize() の呼び出し後に LPT_SetStartPoint*を呼び出しても開始点を追加することはできません．

## 3.3　LPT_InitializeArgs 構造体の設定

流体ソルバの解析領域に関する情報 (原点座標，セル数，セル幅など) を LPT_Initialize() へ渡すために，引数構造体, LPT_InitializeArgs 構造体にこれらの値を設定します．設定する値の詳細はユーザガイドをご覧ください．

## 3.4　LPT_Initialize() の処理内容

LPT_Initialize() 内では，以下の処理を順次行なっています．

1. PMlib のインスタンス生成，初期化
2. DSlib のインスタンス生成，初期化
3. DecompositionManager のインスタンス生成，初期化
4. PPlib のインスタンス生成，初期化
5. Communicator のインスタンス生成，初期化
6. 粒子データ出力ファイルのストリーム作成，ヘッダ部の出力
7. 開始点の分散処理

# 第 4 章

# 計算部分

本章では，LPT のインターフェースである LPT_Calc() および内部で利用されるメソッドの処理を解説する．

## 4.1　主要な計算処理

LPT_Calc() の処理フローを 4.1 に示します.

図 4.1　LPT_Calc の処理フロー

LPT_Calc() の処理は，大きく分けると，タイムステップ毎の初期処理と粒子計算本体部分に分けることができます．

タイムステップ毎の初期処理部分では，寿命を越えた開始点および粒子のオブジェクトの破棄と新規粒子の放出を行ないます．続いて，各粒子の計算に必要なデータブロックを調べ，送信要求の準備をします．また，データブロックの到着順に粒子計算を行うために，粒子オブジェクトを，粒子が属するデータブロック順に並べ替えます．

粒子計算本体部分では，まずこのタイムステップで必要な通信回数を計算します．これは，(計算に必要なデータブロックの総サイズ)/(通信バッファサイズ) を元に計算され Allreduce 通信を用いて，全 Rank 中最も必要な通信回数が大きい値をそのタイムステップでの通信回数とします．

続いて，計算に必要なデータブロックの転送を行います．詳細は**??**章をご覧ください．

データブロックの受信を確認すると，到着したデータブロックに含まれる粒子をワーク領域にコピーしコピー後のデータを対象に粒子の移動を計算します．データの受信確認は，MPI_Test() 関数を用いて，規定の回数に達するまで行なった後 MPI_Wait 関数に切り替えて，通信を完了させます．通信完了後に，未計算の粒子が残っていた場合は，この粒子の移動を計算し，粒子計算部分の最初に計算した通信回数が 2 以上だった場合は，データブロックの転送以降の処理をその回数分繰り返します．

# 第 5 章

# 後処理

本章では，計算完了後に行う処理について説明する．

## 5.1　LPT_Post() の処理内容

LPT_Post() 内では，LPT ライブラリ内で確保したリソースの解放を行います.

具体的には，FileManager クラスが保持するファイルストリームと LPT_Initialize() 内で生成した，PPlib, DSlib, Communicator クラスのインスタンスを破棄します．また，PMlib による測定機能を有効にしていた場合は，測定結果を出力した上で PMlib のインスタンスも破棄します.

LPT_Post() 内では粒子データの出力処理は行なわないので，最終ステップの計算結果を明示的に出力するためにはソルバ側の後処理部分で，LPT_Post() の呼び出しより前に，LPT_OutputParticleData() を呼び出してください.

# 第 6 章

# 通信処理

本章では，LPT 内部で行う粒子-流体プロセス間の通信について説明する．

## 6.1　前提条件

LPT ライブラリが想定する実行モデルは、あるコミュニケータ内の全 MPI プロセスが流体計算を行い、かつ LPT ライブラリのインタフェースルーチンを呼び出す形の SPMD モデルです。しかし、LPT ライブラリ内では自 Rank が流体計算を担当するデータに関する送信処理も行なっていますので本章では便宜上、これらの処理を行なっているプロセスを流体プロセスと呼び、粒子計算のために、これらのデータを要求するプロセスを粒子プロセスと呼びます。

これらの呼称は、通信処理の解説のためのもので、実際には全プロセスが流体プロセスとしての処理と粒子プロセスとしての処理の両方を行なっていることに、ご留意ください。

## 6.2　通信の処理フロー

粒子-流体プロセス間通信は 3 段階に分けて行なわれます。

第 1 ステップでは、粒子プロセス内で自 Rank の計算に必要なデータブロックの ID を要求先の流体プロセス毎にリストアップし、送信を要求するブロック数を Alltoall 通信によって送受信します。

第 2 ステップでは、流体プロセスは第 1 ステップで受け取った要求ブロック数の値を元に必要な数の受信バッファを用意してブロック ID の受信処理を行ないます。

粒子プロセスは、第 1 ステップでリストアップしたブロック ID をそれぞれのデータブロックを保持している Rank に対して送信します。この時、要求ブロック数が 0 となる組み合わせに関しては、送受信ともに処理を行いません。

第 3 ステップでは、 流体プロセスは、第 2 ステップで受け取ったブロック ID のリストを元に自 Rank が保持する流体データを送信します。この時、流体プロセス側ではデータのパッキング処理を行なうため、送信処理には多少の時間が必要です。したがって、データブロックの到着順は要求先の Rank によって大幅に異なることが予想されます。

粒子プロセスは第 1 ステップでリストアップしたブロック ID のリストを元に受信バッファを用意してデータブロックの受信処理を行います。前述のようにデータブロックの到着順が前後する可能性が高いので、MPI_Test() を用いて Polling を行ない、データが到着したブロックに含まれる粒子データから先に計算を行います。この Polling 処理を LPT.NumPolling 回繰り返した後で、MPI_Wait() に切り替えて通信を完了させます。

ここまでの流れをまとめると図 6.1 のようになります。

図 6.1　粒子-流体間通信の流れ

# 第 7 章

# キャッシュアルゴリズム

本章では，流速場キャッシュ機構のアルゴリズムについて説明する．

## 7.1　キャッシュの構成

LPT ライブラリでは，流速場の状態をブロック毎に”初期状態”,”リクエスト待ちデータ”,”リクエスト済データ”,”キャッシュ済データ”の 4 つの状態に分類しています．このうち，初期状態のブロック ID は管理していませんが，”リクエスト待ち”, ”リクエスト済”の二つについては DSlib のメンバのコンテナにブロック ID を格納することで管理しています．また，キャッシュ済データについては，2.6 章に記載した，Cache 構造体を格納するコンテナを DSlib のメンバとして持ちここに登録することで管理しています．

これら 4 つの状態間の遷移とトリガーを図 7.1 に示します．この状態遷移は一方通行になっています．また，粒子近傍のデータブロックの探索は 1 タイムステップにつき 1 回しか行なわれないためキャッシュ溢れによって追い出されたデータブロックが，同一タイムステップ中に必要になっても再送を要求することはありません．

図 7.1　データブロックの状態遷移

## 7.2　追い出しアルゴリズムとその実装

キャッシュデータが規定のサイズを越えた時に，どのデータをキャッシュから削除するかを決めるアルゴリズムを一般に追い出しアルゴリズムと呼びます．LPT ライブラリが採用している追い出しアルゴリズムは，LRU(Least Recently Used) と呼ばれるアルゴリズムで最後に参照されてからの経過時間が最も長いものをキャッシュから削除します．

LPT ライブラリにおける LRU アルゴリズムの実装は次のようになっています．

データブロックの管理のために，ブロック ID とデータブロック本体へのポインタをまとめた Cache 構造体を用意します．キャッシュ済データリストは，この Cache 構造体へのポインタを格納した deque として実装されています．

流速データを受信すると，受信したデータはキャッシュ済データリストの先頭要素に追加されます．また，キャッシュ済データの探索は，先頭要素から順に線形探索で行い，データが見つかった時にはキャッシュ済データリストの先頭位置に移動させます．

したがって，常に最後に参照されたデータがリストの先頭に位置しており，参照されてからの相対的な時間が長いものほど，後ろに位置していることになります．

キャッシュ済データを削除する時には，deque の後ろ側から順に削除することで，LRU アルゴリズムを実現しています．

# 付録 A

# ファイルフォーマット

本章では，LPT ライブラリのファイルフォーマットについて説明する．

LPT が出力する粒子データのネイティブファイルフォーマットは図 A.1 に示すようにファイルヘッダ部，レコードヘッダ部，データ部の 3 つの部分から構成されています．

ファイルヘッダ部は，ファイル全体を通して 1 回だけ先頭部分に出力されます．レコードヘッダ部は，データ出力を行う毎に 1 回だけ出力され，このタイミングで出力される粒子数が記録されています．データ部は，個々の粒子データを記録する場所で，出力する時点で有効な粒子数回の出力が繰り返されます．

個々の部分に出力されるデータの詳細は表 A.1 をご覧ください．

図 A.1　LPT ネイティブファイルフォーマットの構成

表 A.1　LPT ネイティブファイルフォーマットの詳細

| 型 | 要素数 | 値 (固定値の場合) | 値の意味 | 記録される場所 |
|---|---|---|---|---|
| int | 4 | 0xff, 0xef, 0xff,0xef | Byte Order Mark | ファイルヘッダ部 |
| double | 1 | | データフォーマットのバージョン番号 | |
| size_t | 1 | 4 or 8 | sizeof(REAL_TYPE) | |
| unsigned long | 1 | | 粒子数 | レコードヘッダ部 |
| int | 2 | | 開始点 ID | レコード部 |
| int | 1 | | 粒子 ID | |
| long | 1 | | ブロック ID | |
| REAL_TYPE | 3 | | 粒子座標 | |
| REAL_TYPE | 3 | | 粒子速度 | |
| double | 1 | | StartTime(生成された時刻) | |
| double | 1 | | LifeTime(出力された時点での寿命) | |
| double | 1 | | 現在時刻 | |
| unsigned int | 1 | | 現在のタイムステップ | |

# 付録 B

# アップデート情報

本設計書のアップデート情報について記す.

Version 0.9　　2013/6/15

- ドキュメントの様式整備.
- configure でのコンパイルでパッケージ化.

Version 0.8　　2013/6/15

- $\beta$ version

Version 0.7　　2013/6/10

- $\alpha$ version