Lagrangian Particle Tracking Library

User's Guide

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	18 June	2013
Version 0.8	13 June	2013
Version 0.7	10 June	2013

COPYRIGHT

Copyright (c) 2012-2013 Advanced Institute for Computational Science, RIKEN.
All rights reserved.

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します．

1.1　本ドキュメントの目的

本ドキュメントは Lagragian Particle Tracking 法による粒子追跡機能を提供する LPT ライブラリの機能と使用方法を説明します.

1.2　LPT が提供する機能

LPT を用いることで，解析領域内のユーザが指定した開始点から放出される粒子が，流体と相互作用する際の挙動をシミュレーションすることができます.

開始点としては，特定の座標を指定した 1 点，または線分，矩形，直方体および円状の領域中の複数の点を指定することができます．さらに，1 点のみで構成される開始点は時間に応じて移動させることも可能です.

開始点には，粒子の放出を始める時間，粒子を放出する時間間隔，ライフタイムを設定することができます．粒子にも開始点単位でライフタイムを設定することができます．開始点および粒子は，生成されてからライフタイムに設定された時間を経過すると消滅します.

また，本ライブラリが出力した粒子データを可視化するため，FieldView 等のポストプロセッサが読み取り可能な形式に変換するファイルコンバータもパッケージ内に含まれています.

1.3　LPT を使用するために必要なもの

LPT を動作させるためには，直交等間隔格子を採用し，MPI を使用して領域分割型の分散並列化をされた流体計算プログラムに組込む必要があります.

また，本ライブラリに含まれる性能情報採取機能を有効にする場合は PMlib をリンクする必要があります.

第 2 章

コンパイル

本章では，LPT ライブラリのコンパイル方法について説明します.

2.1　事前準備

LPT の性能情報測定機能を使用する場合は，事前に PMlib をコンパイルしておく必要があります．PMlib のコンパイルは基本的に下記の手順でインストールします．詳細は，PMlib のユーザガイドをご覧ください．

```
$ ./configure [options]
$ make
$ make install
```

2.2　LPTlib のコンパイル

基本的に PMlib と同様に configure でビルドします．もし，configure で自動インストールできない場合には，手動コンパイルの方法があります．

2.2.1　configure を使う場合

LPTlib-x.x.tar.gz を展開したディレクトリで下記を実行します．

```
$ ./configure [options]
$ make
$ make install
```

オプションとして，2.1 を指定可能です．上記が実行されると，ライブラリと FileConverter 実行モジュールが指定ディレクトリにインストールされます．
以下に，Intel コンパイラと OpenMPI の場合のパラメータを示します．

```
$./configure --with-pm=/usr/local/PMlib \
             --with-comp=INTEL \
             --with-ompi=/opt/openmpi \
             --with-log=enable \
             CXXFLAGS=-O3 \
             CXX=icpc \
             FC=ifort
```

次の例は，京でのコンパイルで，インストール先を$1 で表しています．

```
$./configure --prefix=$1 \
             --with-pm=/hogehoge \
             --with-comp=FJ \
             --with-log=enable \
             --host=sparc64-unknown-linux-gnu \
             CXX=mpiFCCpx \
             CXXFLAGS=-Kfast \
             FC=
```

2.2.2　手動コンパイルの場合

手動でコンパイルする場合，Makefile_hand を用いて，次のようにコンパイルします．

```
$ make -f Makefile_hand
```

表 2.1　configure option

オプションの指定形式	指定内容
–prefix=INSTALL_DIR	インストール先ディレクトリを指定します.
–with-mpich=MPICH_DIR	MPI ライブラリとして MPICH を用いる場合，MPICH のインストールディレクトリを指定します．mpicxx などコンパイラの MPI ラッパーを用いる場合には，–with-mpich と–with-ompi の両方を省略できます.
–with-ompi=OPENMPI_DIR	MPI ライブラリとして OpenMPI を用い，OpenMPI のインストールディレクトリを指定します.
–with-comp=(INTEL|FJ)	コンパイラを指定する．INTEL と Fujitsu コンパイラが指定できます．その他のコンパイラの場合には指定しません.
–with-pm=PMLIB_DIR	性能測定レポートを生成する場合，Performance Monitor library のインストールディレクトリを指定します.
–with-log=(diable|enable)	LPTlib の実行ログを表示する場合に enable を指定します．デフォルトは disenable です.
–with-realtype=(float|double)	LPTlib の浮動小数点の基準を指定します．デフォルトは float です.
–with-container=(deque|list)	キャッシュアルゴリズムに用いるコンテナを指定します．デフォルトは deque です.
–host	クロスコンパイル時にアーキテクチャを指定します.
CXX=CXX_COMPILER	C++ コンパイラを指定します.
FC=FORTRAN90_COMPILER	Fortran90 コンパイラを指定します.
CXXFLAGS=CXX_OPTIONS	C++ コンパイラのオプションを指定します.

第 3 章

開始点の指定

本章では，粒子の開始点の指定方法について説明する．

3.1　用語の定義

本ドキュメントで用いる開始点に関連する用語を表 3.1 のように定義します.

表 3.1　本ドキュメントで使用する開始点関連の用語定義

開始点	粒子を放出する個々の点
開始点領域 (または領域)	複数の開始点が存在する一定の範囲
開始点設定	ある形状の領域の定義とそこに含まれる開始点の数，寿命等の設定をまとめたもの

3.2　開始点設定の一覧

LPT ライブラリでは以下の 6 種類の開始点設定を指定することができます.

- Point
- MovingPoints
- Line
- Rectangle
- Cuboid
- Circle

3.3　全てのタイプに共通の設定

全てのタイプの開始点設定に共通して次の 4 つのパラメータを指定することができます.

StartTime　粒子の放出を始める時刻
ReleaseTime　開始点の寿命
TimeSpan　1 回放出してから次に放出を行なうまでの時間間隔
ParticleLifeTime　その開始点から放出される粒子の寿命

3.4　Point 型の開始点設定

Point 型の開始点設定は，解析空間中の 1 点を開始点として指定するもので，指定された座標から粒子が放出されます.

3.5　MovingPoints 型の開始点設定

MovingPoints 型の開始点設定は，Point 型と同じく解析空間中の 1 点から粒子が放出されます．さらに，時刻に応じて開始点の位置を移動させることができます.

3.6　Line 型の開始点設定

Line 型の開始点設定は，解析空間内の 2 点を結ぶ線分上の，複数の開始点から粒子が放出されます．図 3.6 に示すようにユーザは線分の両端にあたる点の座標と，この線分上に存在する開始点の数を指定する必要があります.

図 3.1　Line 型の開始点設定

3.7　Rectangle 型の開始点設定

Rectangle 型の開始点設定では，軸と直交する平面内に定義される矩形領域内に格子状に配置された開始点から粒子が放出されます．

図 3.7 に示すように，ユーザは領域を規定する 2 頂点の座標と各軸方向に並ぶ開始点の数を指定する必要があります．なお，開始点数は x,y,z のうちどれか 1 方向に対しては 1，残り 2 方向に対しては 1 以上の値を設定する必要があります．

図 3.2　Rectangle 型の開始点設定

3.8　Cuboid 型の開始点設定

Cuboid 型の開始点設定では，各面が軸と直交している直方体型の領域内に格子状に配置された開始点から粒子が放出されます．

図 3.8 に示すように，ユーザは領域を規定する 2 頂点の座標と各軸方向に並ぶ開始点の数を指定する必要があります．

図 3.3　Cuboid 型の開始点設定

3.9　Circle 型の開始点設定

Circle 型の開始点設定では，解析空間内の任意の円内に同心円状に広がった開始点から粒子が放出されます．ユーザは円を規定するための，中心座標，半径および法線ベクトルを指定しさらに，この円内に存在する開始点数の最大値を指定します．

プログラム内部では指定された開始点数の最大値をもとに，半径方向を n 分割し半径 $r/n * i$ (i=0,1,2…n) の円上に，$a * i$ 個の開始点を配置します．この時の半径方向の分割数 n および円周方向の開始点数を決める係数 a の値は最外周の円上に並ぶ開始点間の距離と，同心円の間隔が，同じ程度になるように決めています．

図 3.4

第 4 章

流体計算プログラムへの組込み

本章では，LPT ライブラリを流体ソルバーへ組み込む処理について説明する．

4.1　流体計算プログラムへの組込み方法概要

本ライブラリを用いて粒子-流体連成計算を行なうためには，流体プログラムに次の 4 つのインターフェースルーチンを組込む必要があります.

- LPT_Initialize()
- LPT_CalcParticleData()
- LPT_OutputParticleData()
- LPT_Post()

また，開始点の設定を行なうためには以下の 5 つの開始点設定ルーチンを組込む必要があります．これらのルーチンは異なる種類のものを混在させたり，同じ種類のものを複数使用しても構いません.

- LPT_SetStartPoint()
- LPT_SetStartPointLine()
- LPT_SetStartPointRectangle()
- LPT_SetStartPointCuboid()
- LPT_SetStartPointCircle()
- LPT_SetStartPointMovingPoints()

図 4.1 に一般的な流体解析ソルバの構造と，そこへ組込む LPT の処理を示します.

図 4.1　LPT の組込み概略図

4.2　粒子計算インターフェースルーチン

本章では，各インターフェースルーチンの処理内容と呼び出しに関する注意事項を説明します．

4.2.1　LPT_Initialize()

LPT_Initialize() では，引数構造体 LPT_InitializeArgs を通して連成計算に必要な流体ソルバの設定情報や LPT の動作パラメータ (キャッシュサイズ等) を受け取り LPT 内部で使用するクラスのインスタンスを生成します．また，開始点を分割して小領域に分け，粒子計算を担当するプロセスに渡す処理も行ないます．本ルーチンは，MPI_Init() の呼び出しより後で呼ばれる必要があります．また，本ルーチンを呼び出す前に開始点設定ルーチンによって開始点を設定しておく必要があります．本ルーチンの引数構造体 LPT_InitializeArgs の内容は表 4.1 および表 4.2 を参照してください．

4.2.2　LPT_CalcParticleData()

LPT_CalcParticleData() では，引数構造体 LPT_CalcArgs を通して現在のタイムステップに関する情報を受けとり，粒子計算を行ないます．本ルーチンによって更新された粒子データは，LPT_OutputParticleData() を用いてユーザ自身で出力する必要があります．本ルーチンの引数構造体 LPT_CalcArgs の内容は表 4.3 を参照してください．

4.2.3　LPT_OutputParticleData()

LPT_OutputParticleData() では，現在有効な粒子データを LPT の独自形式でファイルに出力します．出力されるファイルのファイル名は，{BaseFileName}_{Rank 番号 }.prt という形式になっており BaseFileName の部分は，LPT_Initialize() の引数構造体 LPT_InitializeArgs の内部で任意の文字列を指定することができます．引数構造体内で指定されていない場合は”ParticleData”が使用されます．

4.2.4　LPT_Post()

LPT_Post() では，計算に使用した開始点や粒子データ，LPT_Initialize() 内で生成したインスタンス，LPT_OutputParticleData() 内で使用したファイルストリーム等のリソースを破棄します．また，PMlib による性能測定を有効にしている場合は，Rank0 から測定結果を出力します．

4.3　インターフェースルーチンの引数

本ライブラリのインターフェースルーチンのうち LPT_Initialize() および LPT_CalcParticleData() は必要な引数をそれぞれ LPT_InitializeArgs および LPT_CalcArgs という構造体に格納して渡します．次節以降ではこれらの構造体のメンバに指定する値を説明します．

4.3.1　LPT_InitializeArgs

LPT_Initialize() の引数に指定する構造体 LPT_InitializeArgs のメンバのうち，指定が必須のものを表 4.1 に示します．

オプションのパラメータおよび初期値を表 4.2 に示します．表 4.1 のパラメータは全て指定しなければ粒子計算が行なえませんが，表 4.2 は初期値から変更する必要がなければ指定しなくても構いません．

表 4.1　LPT_Initialize() の引数構造体 (必須)

型	変数名	値の内容
int	Nx	計算領域の x 方向のサイズ (セル数)
int	Ny	計算領域の y 方向のサイズ (セル数)
int	Nz	計算領域の z 方向のサイズ (セル数)
int	NPx	x 軸方向の分割数 (サブドメイン数)
int	NPy	y 軸方向の分割数 (サブドメイン数)
int	NPz	z 軸方向の分割数 (サブドメイン数)
int	NBx	1 サブドメインあたりの x 軸方向のデータブロック数
int	NBy	1 サブドメインあたりの x 軸方向のデータブロック数
int	NBz	1 サブドメインあたりの x 軸方向のデータブロック数
REAL_TYPE	dx	x 方向のセル幅
REAL_TYPE	dy	y 方向のセル幅
REAL_TYPE	dz	z 方向のセル幅
REAL_TYPE	OriginX	解析領域全体の原点座標
REAL_TYPE	OriginY	解析領域全体の原点座標
REAL_TYPE	OriginZ	解析領域全体の原点座標
int	GuideCellSize	袖領域のサイズ (セル数)

表 4.2　LPT_Initialize() の引数構造体 (オプション)

型	変数名	値の内容 (初期値)
MPI_Comm	FluidComm	流体計算で使うコミュニケータ (MPI_COMM_WORLD)
int	NumFluidProc	流体計算に参加するプロセス数 (FluidComm 内のプロセス数)
int	CacheSize	データブロックのキャッシュに使う領域のサイズ．単位は MByte (1024MB)
int	CommBufferSize	キャッシュから 1 度に追い出すサイズ．単位は MByte (512MB)
int	NumParticleProcs	粒子計算に参加するプロセス数の初期値 (NumFluidProc)
int	MaxNumParticleProcs	粒子計算に参加するプロセス数の最大値 (NumFluidProc)
int	MigrationInterval	マイグレーション判定を行なう頻度 (100)
std::string	OutputFileName	出力ファイルの名前 (ParticleData)
std::string	PMlibOutputFileName	PMlib の情報を出力するファイル名 (PMlibOutput.txt)
std::string	PMlibDetailedOutputFileName	PMlib の詳細情報を出力するファイル名 (PMlibDetailedOutput.txt)
int	*d_bcv	FFV 内で使用しているセル情報を保持するビットマスク配列へのポインタ (NULL)

4.3.2　LPT_CalcArgs

LPT_CalcParticle() の引数に指定する構造体 LPT_CalcArgs のメンバを表 4.3 に示します．これらのメンバ変数の指定は全て必須です．正しく設定されていない場合 LPT_CalcParticle() は粒子計算を行ないません．

表 4.3　LPT_CalcParticle() の引数構造体

型	変数名	値の内容
double	CurrentTime	現在時刻
double	deltaT	時間積分幅
double	divT	粒子移動の積分に使う刻み幅の再分割数
REAL_TYPE *	FluidVelocity	流速を保持する配列へのポインタ

4.4　開始点設定ルーチン

開始点設定ルーチンのシグネチャを以下に示します.

```
bool LPT_SetStartPoint(REAL_TYPE Coord1[3], double StartTime,
                       double ReleaseTime, double TimeSpan,
                       double ParticleLifeTime)

bool LPT_SetStartPointLine(REAL_TYPE Coord1[3], REAL_TYPE Coord2[3],
                           int SumStartPoints, double StartTime,
                           double ReleaseTime, double TimeSpan,
                           double ParticleLifeTime)

bool LPT_SetStartPointRectangle(REAL_TYPE Coord1[3], REAL_TYPE Coord2[3],
                                int NumStartPoints[3], double StartTime,
                                double ReleaseTime, double TimeSpan,
                                double ParticleLifeTime)

bool LPT_SetStartPointCuboid(REAL_TYPE Coord1[3], REAL_TYPE Coord2[3],
                             int NumStartPoints[3], double StartTime,
                             double ReleaseTime, double TimeSpan,
                             double ParticleLifeTime)

bool LPT_SetStartPointCircle(REAL_TYPE Coord1[3], int SumStartPoints,
                             REAL_TYPE Radius, REAL_TYPE NormalVector[3],
                             double StartTime, double ReleaseTime,
                             double TimeSpan, double ParticleLifeTime)
```

引数のうち，StartTime から ParticleLifeTime までの 4 つの変数は，3.3 章に記載した，全ての種類の開始点設定に共通のものです．内容を以下に再掲します.

StartTime　粒子の放出を始める時刻
ReleaseTime　開始点の寿命
TimeSpan　1 回放出してから次に放出を行なうまでの時間間隔
ParticleLifeTime　その開始点から放出される粒子の寿命

4.4.1　LPT_SetStartPoint()

LPT_SetStartPoint() は，3.4 章に記載した Point 型の開始点設定を指定するルーチンです．引数 Coord1 には開始点の座標を指定します.

4.4.2　LPT_SetStartPointLine()

LPT_SetStartPointLine() は，3.6 章に記載した Line 型の開始点設定を指定するルーチンです．引数 Coord1, Coord2 には，それぞれ線分の端点の座標を指定します．また，SumStartPoints には，この線分上に存在する開始点の数を指定します．

4.4.3　LPT_SetStartPointRectangle()

LPT_SetStartPointRectangle() は，3.7 章に記載した Rectangle 型の開始点設定を指定するルーチンです．引数 Coord1, Coord2 には，それぞれ領域の対角線上に位置する 2 頂点の座標を指定します．また，配列 NumStartPoints には先頭から順に x 軸方向に並ぶ開始点の数，y 軸方向に並ぶ開始点の数，z 軸方向に並ぶ開始点の数を指定します．

Rectangle 型の開始点設定は軸に直交する平面内にしか設定できないので，Coord1, Coord2 のどれか 1 要素は同じ値を指定する必要があります．また，x 軸に直交する平面内に設定する場合は，NumStartPoints[0] は 1 を指定する必要があります．これらの条件を満たさない引数が渡された場合は，その開始点設定は無効となり本ルーチンは false を返します．

4.4.4　LPT_SetStartPointCuboid()

LPT_SetStartPointCuboid() は，3.8 章に記載した Cuboid 型の開始点設定を指定するルーチンです．引数 Coord1, Coord2 には，それぞれ領域の対角線上に位置する 2 頂点の座標を指定します．また，配列 NumStartPoints には先頭から順に x 軸方向に並ぶ開始点の数，y 軸方向に並ぶ開始点の数，z 軸方向に並ぶ開始点の数を指定します．

4.4.5　LPT_SetStartPointCircle()

LPT_SetStartPointCircle() は，3.9 章に記載した Circle 型の開始点設定を指定するルーチンです．引数 Coord1 には円の中心座標，SumStartPoints には，領域内に存在する開始点数の最大値 Radius には領域の半径，NormalVector には領域が存在する平面の法線ベクトルをそれぞれ指定します．

第 5 章

FFV-C ソルバーへの組み込み

本章では，LPT ライブラリを FFV-C ソルバーへ組み込む処理について説明する．

5.1　ソース修正内容

5.1.1　初期化部分への修正内容

LPT の初期化処理は，FFV_Initialize.C 内に定義されているメンバ関数 FFV::Initialize() 内に追加します．FFV の初期条件の設定を行なっている，setInitialiCondition() の呼び出しの後に以下のコードを追記します．

```
\begin{quote}
  // LPT 初期化処理 start
  cpm_ParaManager::get_instance()->Barrier();
  if ( IsMaster() ) printf("\n\tLPT Initialize start.\n\n");
  LPT::LPT_InitializeArgs init_args;
  init_args.Nx            = G_size[0];
  init_args.Ny            = G_size[1];
  init_args.Nz            = G_size[2];
  const int* divnum = paraMngr->GetDivNum();
  init_args.NPx           = divnum[0];
  init_args.NPy           = divnum[1];
  init_args.NPz           = divnum[2];
  init_args.NBx           = G_size[0]/divnum[0] < 10 ? 1 : G_size[0]/divnum[0]/10;
  init_args.NBy           = G_size[1]/divnum[1] < 10 ? 1 : G_size[1]/divnum[1]/10;
  init_args.NBz           = G_size[2]/divnum[2] < 10 ? 1 : G_size[2]/divnum[2]/10;
  init_args.dx            = pitch[0];
  init_args.dy            = pitch[1];
  init_args.dz            = pitch[2];
  init_args.OriginX       = G_origin[0];
  init_args.OriginY       = G_origin[1];
  init_args.OriginZ       = G_origin[2];
  init_args.GuideCellSize = guide;
  init_args.d_bcv         = d_bcv;
  init_args.FluidComm     = cpm_ParaManager::get_instance()->GetMPI_Comm();
  MPI_Comm_size(init_args.FluidComm, &init_args.NumFluidProc);

  //開始点を定義
  double StartTime=0.0;
  double ReleaseTime=0.1;
  double TimeSpan=0.2;
  double ParticleLifeTime=0;
  REAL_TYPE Coord1[3]={0.0,   0.0,   G_origin[2]+G_region[2]*9/10 };
  LPT::LPT::GetInstance()->LPT_SetStartPoint(Coord1, StartTime,
                           ReleaseTime, TimeSpan, ParticleLifeTime);

  cpm_ParaManager::get_instance()->Barrier();
  if ( IsMaster() ) printf("\n\tcall LPT_Initialize()\n\n");
  //初期化
  LPT::LPT::GetInstance()->LPT_Initialize(init_args);
  cpm_ParaManager::get_instance()->Barrier();
  if ( IsMaster() ) printf("\n\tLPT Initialize end.\n\n");
  // LPT 初期化処理 end
```

5.1.2　タイムステップループ部分への修正内容

粒子計算の呼び出し部分は，FFV_Loop.C 内で定義されているメンバ関数 FFV::Loop() 内に追加します．
引数構造体への値の代入と LPT_CalcParticleData() の呼び出しをまとめて以下のように組込みます

```
//LPT 引数設定 start
static LPT::LPT_CalcArgs calc_args;
calc_args.FluidVelocity=d_v;
calc_args.v00=v00;
calc_args.deltaT=deltaT;
calc_args.divT=1;
calc_args.CurrentTime=CurrentTime;
calc_args.CurrentTimeStep=CurrentStep;
//LPT 引数設定 end
//LPT 粒子計算呼び出し
LPT::LPT::GetInstance()->LPT_CalcParticleData(calc_args);
```

また，粒子データの出力を行う LPT_OutputParticleData() を同ルーチン内に組込みます．Plot3D の出力と同期して粒子データの出力を行う場合は，PLT3D.OutputPlot3D_post() の呼び出し直後に組込みます．

```
LPT::LPT::GetInstance()->LPT_OutputParticleData();
```

5.1.3　後処理部分への修正内容

後処理部分は，FFV_Post.C 内で定義されているメンバ関数 FFV::Post() の末尾に以下のように追加します．

```
LPT::LPT::GetInstance()->LPT_Post();
```

5.2　include 文の追加

LPT のルーチンを組み込んだ 3 つのファイル FFV_Initialize.C, FFV_Loop.C, FFV_Post.C のそれぞれに対して LPT.h のインクルード文を追加します．引数構造体 LPT_InitArgs および LPT_CalcArgs を定義している LPT_Args.h は LPT.h 内でインクルードしているので明示的にインクルードする必要はありません．

5.3　LPT を組み込んだ FFV-C のコンパイル

LPT を組み込んだ FFV-C をコンパイルする際には，インクルードパスにヘッダファイルを置いているディレクトリを指定する必要があります．また，リンク時には LPT ライブラリのファイルを指定する必要があります．
LPT ライブラリをコンパイルしたディレクトリのパスを$LPT_DIR とするとコンパイルオプションには以下を追加します．

```
-I$(LPT_DIR)/include
```

また，リンクオプションには以下を追加します．

```
-L$(LPT_DIR)/lib -lLPT
```

第 6 章

ファイル変換

本章では，ファイル変換について説明する．

6.1　LPT が出力する粒子データファイルの書式

本ライブラリは，ユーザが LPT_OutputParticle() を呼び出したタイミングで有効な粒子の座標および物理量を，独自形式のバイナリファイル (以下ネイティブ形式) として出力します．

LPT の配布物にはネイティブ形式から，FieldView の ParticlePath 形式や csv 形式に変換する変換ツールが含まれています．粒子計算の結果を確認する際には，この変換ツールを用いてファイル形式を変換した上で可視化ソフトに読み込んでください．

ファイルフォーマットの詳細は設計書をご覧ください．

6.2　LPT が提供するファイル変換ツールの使い方

変換ツールは**??**章の手順に従ってビルドを行なうと bin ディレクトリの下に FileConverter という名前で作成されます．

ファイル形式を変換するためには，引数として出力ファイルが存在するディレクトリを指定して変換ツールを実行します．

```
> bin/FileConverter [ディレクトリ名]
```

有効なオプションは表 6.1 のとおりです．

表 6.1　変換ツールの実行時オプション

-csv	カンマ区切りの csv 形式で時刻と座標のみを出力します．
-FVbin	FieldView の ParticlePath 形式のバイナリ版で出力します．
-FV(デフォルト)	FieldView の ParticlePath 形式のアスキー版で時刻と座標のみを出力します．

この変換ツールは，MPI 並列化されており，入力ファイル単位でデータ分割しています．引数に指定したディレクトリに存在する拡張子が prt のファイル全てを粒子データファイルとして扱います．また，引数のディレクトリ名を省略した場合は，カレントディレクトリが指定されたものとして扱います．

入力ファイルはアルファベット順に 1 ファイルづつ Rank0 から順に割り当てられ，各 Rank は自 Rank が担当するファイルのみを open します．

例えば，カレントディレクトリに ParticlePath_0.prt, ParticlePath_1.prt, ParticlePath_2.prt, ParticlePath_3.prt という 4 ファイルが存在する状態で，2 プロセスで変換ツールを実行すると Rank0 が ParticlePath_0.prt と ParticlePath_2.prt を Rank1 が ParticlePath_1.prt と ParticlePath_3.prt の変換を担当します．

したがって，NFS 等の共有ファイルシステムを使わずに複数ノードでこの変換ツールを実行する場合は事前に，各 Rank が動作するノードに対して必要なファイルを転送する必要があります．

ここまでの章で書いていない内容でエンドユーザ向けのものがあれば記載する

付録 A

アップデート情報

本ユーザガイドのアップデート情報について記す.

Version 0.9　　2013/6/18

- 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

- β version

Version 0.7　　2013/6/10

- α version