Sphere

Skeleton for PHysical and Engineering REsearch

Ver 2.0.0

並列実行形態・並列制御クラスマニュアル

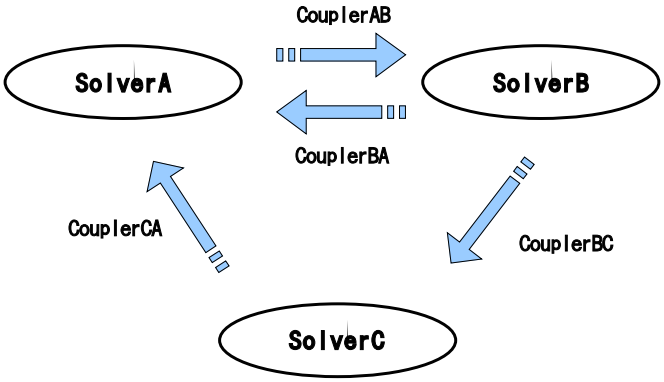
2010 年 8 月 01 日

目次

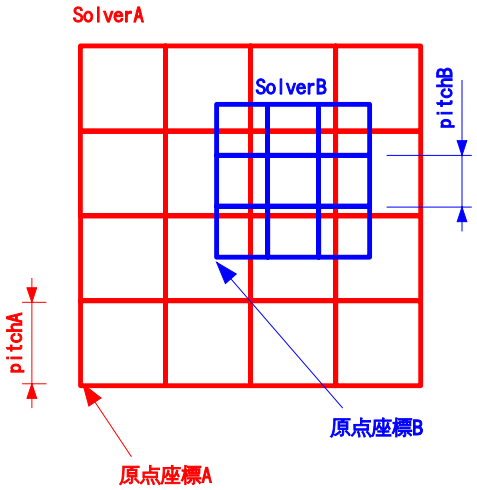
１．　Sphere の並列実行形態概要

Sphere はソルバの実行環境をサポートするミドルウェアライブラリです。
Sphere2.0 の特徴として、複数のソルバと複数のカプラをサポートします。

ソルバ：構造格子計算モデル、非構造格子計算モデルをサポートします。
カプラ：ソルバで計算されたデータをソルバ間でデータコピーをサポートします。
構造格子データと非構造格子データをサポートします。



複数のソルバは単独に計算領域、分割ノードを指定可能で個々のソルバとして動作します。



ソルバは定義された自分の計算領域の解析を行い、カプラによってソルバ間の連成が行われます。

２．　Sphere の並列実行プロセス

ソルバの実行プロセスは、並列実行された全体のプロセス数と分割数、プロセス数、ノ

ードリストによって個々のソルバに割当てられたプロセス数により以下の並列実行形態を持ちます。

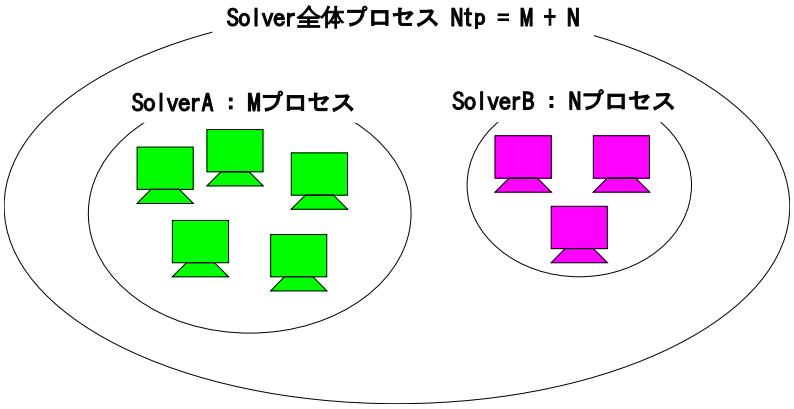
全体プロセス数(Ntp)："mpirun –np Ntp"により起動された全体のプロセス数

ソルバプロセス数(M,N)：<SphVoxelDivision>（分割数）要素、<SphNodeProcess>（プロセス数）要素、<SphNodeList>（ノードリスト）要素によって割当てられたプロセス数

No	並列実行プロセス	プロセス数	ソルバ／プロセス
1	単一ソルバ実行	Ntp = M + N	1 つのプロセスに 1 つだけのソルバ
2	複数ソルバ実行	Ntp = M or Ntp = N	1 つのプロセスに複数のソルバ

（1）単一ソルバ実行（Ntp = M + N）

1 つのプロセスは 1 つだけソルバが割当てられ、1 つのソルバのみを実行します。



Ntp：mpirun –np Ntp"により起動された全体のプロセス数

M：SolverA に割当てられたソルバプロセス数

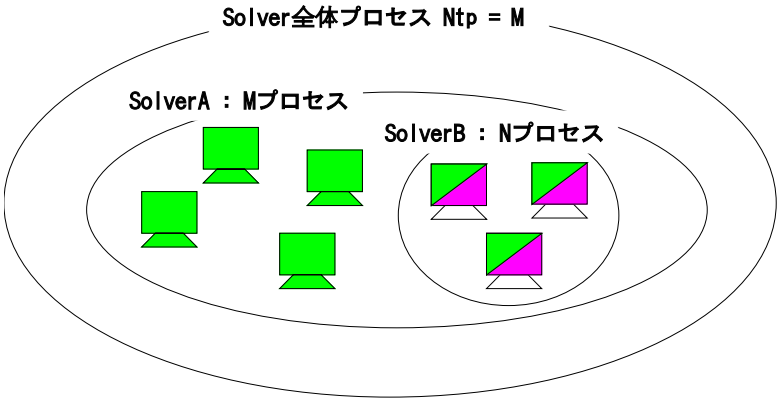
N：SolverB に割当てられたソルバプロセス数

1 つのプロセスに 1 つのソルバしか存在しませんので、1 CPU のハードウェアを占有することができます。

しかし、"SolverA"と"SolverB"間の連成（カプラ）の為に通信コストが多く発生します。

（2）複数ソルバ実行（Ntp = M or Ntp = N）

1 つのプロセスは複数のソルバが割当てられ、複数のソルバを逐次実行します。



Ntp："mpirun –np Ntp"により起動された全体のプロセス数

M：SolverA に割当てられたソルバプロセス数

N：SolverB に割当てられたソルバプロセス数

１つのプロセスに複数のソルバが存在しますので、１CPU のハードウェアを共有することになります。

しかし、"SolverA"と"SolverB"間の連成（カプラ）の為の通信コストは同一プロセス内のメモリコピーで行う一部のプロセスがありますので少なくなります。

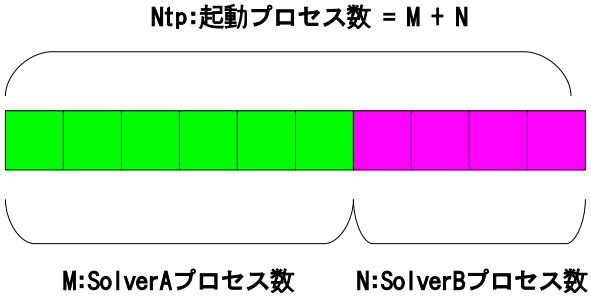
２．１　ソルバプロセスの割当

ソルバプロセスは起動された全体のプロセス数とソルバに定義されたプロセス数により自動的に割り振りを行います。

No	ソルバ割当	プロセス数定義	プロセス数	ソルバ／プロセス
1	単一ソルバ	あり	Ntp = M + N	１つのプロセスに１つのソルバ
2	複数ソルバ	あり	Ntp = M or Ntp = N	１つのプロセスに複数のソルバ
3	複数ソルバ	なし	Ntp = M = N	１つのプロセスにすべてのソルバ
4	空プロセス	あり	Ntp > M + N	ソルバ未割当のプロセス
5	割当エラー	あり	Ntp < M	ソルバ割当エラー

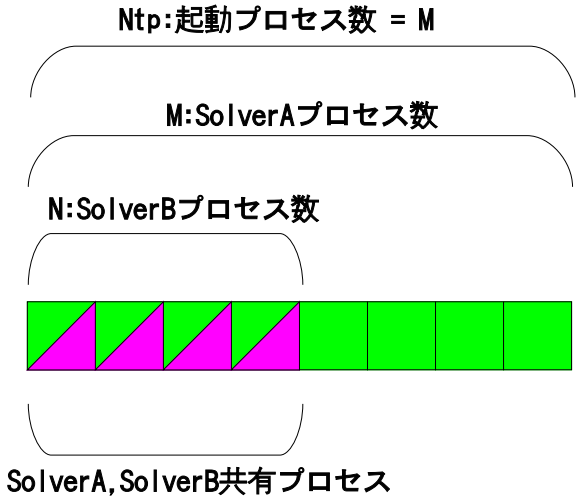
（１）単一ソルバ割当（ソルバプロセス数定義あり）

ソルバに定義されたプロセス数の総和が起動プロセス数と等しい場合、１つのプロセスに１つのソルバが割当てられます。



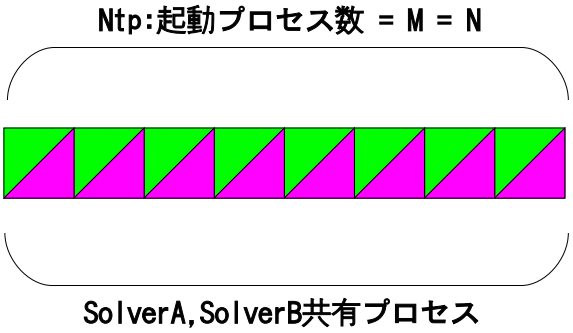
（２）複数ソルバ割当（ソルバプロセス数定義あり）

１つのソルバに定義されたプロセス数が起動プロセス数と等しい場合、１つのプロセスに複数のソルバが割当てられます。



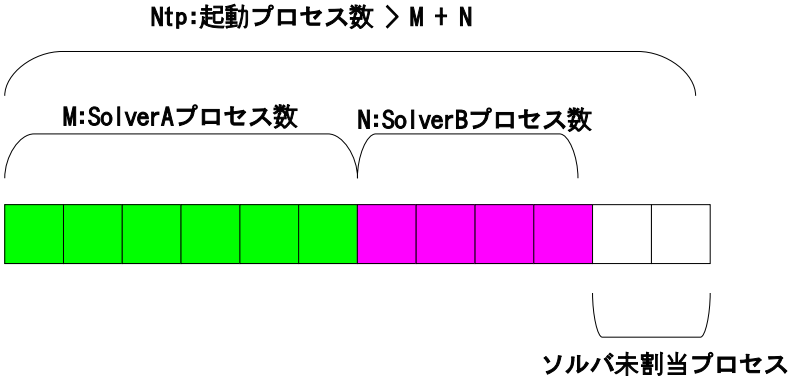
（３）複数ソルバ割当（ソルバプロセス数定義なし）

ソルバにプロセス数が定義されていない場合、起動プロセスすべてにソルバが割当てられます。



（４）空ソルバ割当（ソルバプロセスが存在しない）

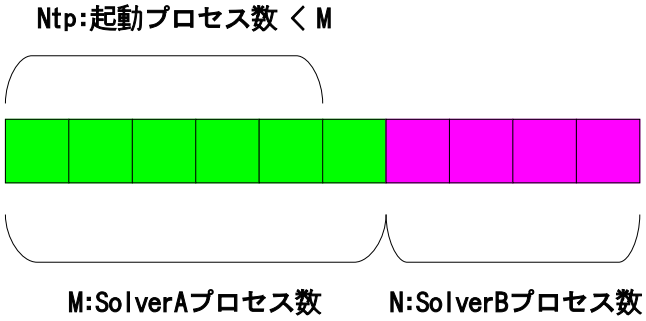
ソルバに定義されたプロセス数の総和が起動プロセス数より大きい場合、ソルバが割当てられないプロセスが存在します。



（５）割当不可能（割当エラー）

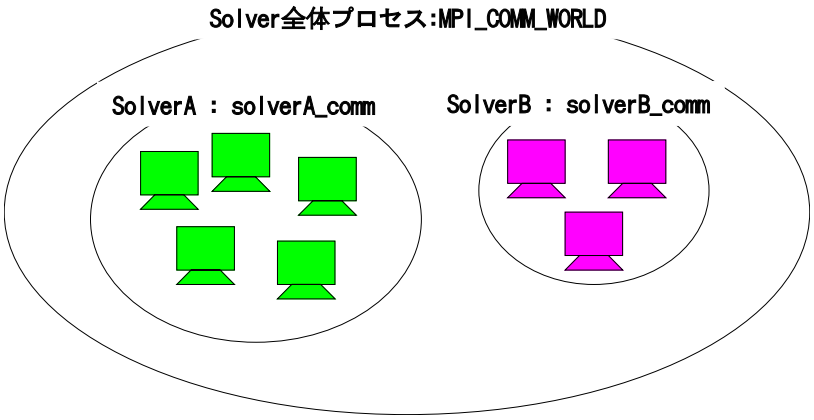
ソルバに定義されたプロセス数が起動プロセス数より小さい場合、ソルバに割当ができませんのでエラーとなります。

(エラー)



２．２　ソルバグループ、コミュニケータ

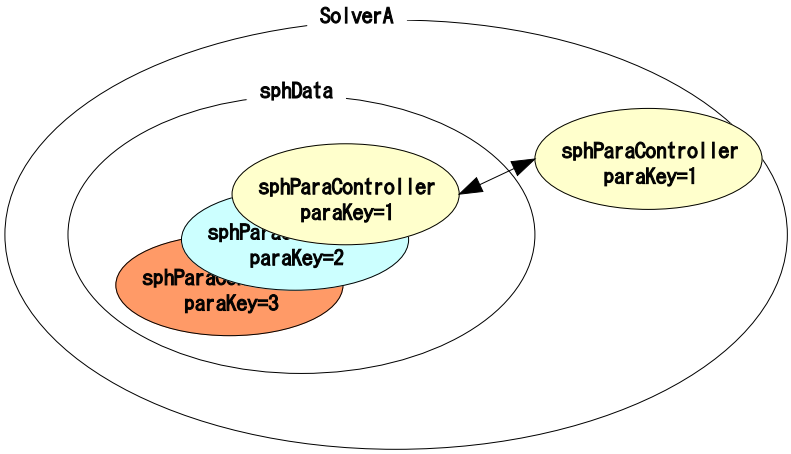
Sphere はソルバプロセスの割当時、MPI グループ、コミュニケータを生成します。



MPI_COMM_WORLD：Sphere 実行プロセス全体が属する MPI コミュニケータ
solverA_comm：SolverA のプロセスが属する MPI コミュニケータ
solverB_comm：SolverB のプロセスが属する MPI コミュニケータ

２．３　並列制御クラス

並列制御クラス(sphParaController)は、ソルバの計算領域の管理、並列実行における通信管理を行います。



並列制御クラスはソルバに１つだけ持ちますが、データクラスには複数持つことができます。

ソルバの並列制御クラスはソルバのすべての実行プロセス（ノード）間で通信を行います。

データクラスに最初に登録される並列制御クラスは、ソルバの並列制御クラスとなりますが、任意のプロセス（ノード）間で MPI コミュニケータ・グループを作成し、そのプロセス間でのみ通信を行うことができます。

３．　Sphere のプログラムフロー

Sphere は連成を行うプログラムフローをソルバ開発者が自由に記述できます。
プログラムフローはソルバ個々に定義可能ですが、１つのプロセスに１つのフローしか定義できません。

１つのプロセス内に複数のソルバが存在する場合、複数のフローは定義できません。

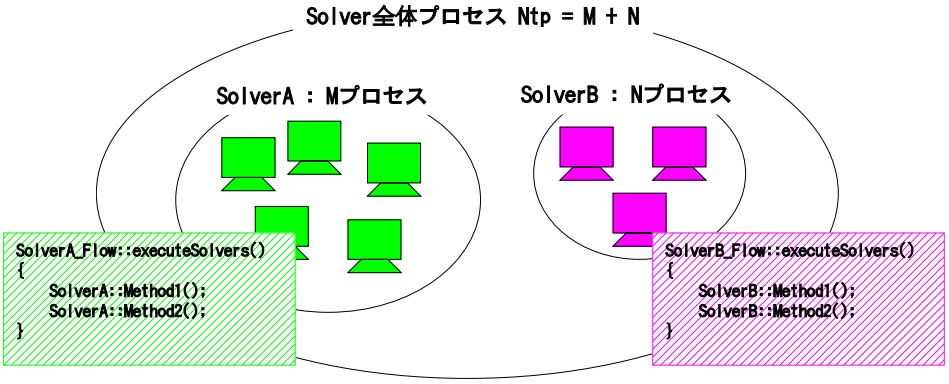
No	プロセス数	プログラムフロー	定義可能フロー
1	Ntp = M + N	ソルバ定義フロー	ソルバに定義されたフロー
2		ソルバ共通フロー	ソルバに共通定義されたフロー
3	Ntp = M or Ntp = N	ソルバ共通フロー	ソルバに共通定義されたフロー

ソルバ定義フロー：ソルバ毎に定義されたフロー。
"SphSolver/SphFlow[@label='フロー名']"要素にて定義されたフロー

ソルバ共通フロー：ソルバ共通に定義されたフロー。
"SphereConfig/SphFlow[@label='フロー名']"要素にて定義されたフロー

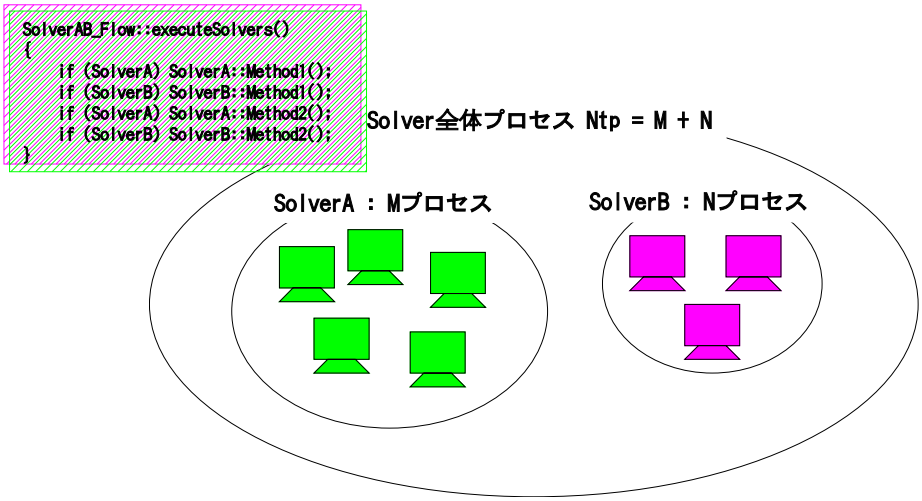
（１）ソルバ定義フロー（Ntp = M + N）

１つのプロセスに１つのソルバが割当われている場合、ソルバに定義されたフローによりプログラムを実行することができます。



（２）ソルバ共通フロー（Ntp = M + N）

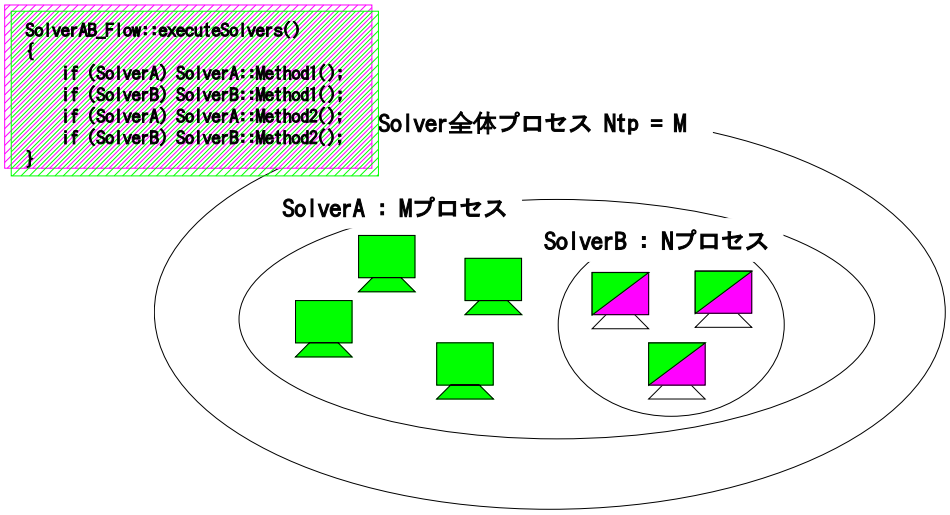
１つのプロセスに１つのソルバが割当われている場合にてもすべてのソルバに共通に定義されたフローによりプログラムを実行することができます。



ソルバ毎にソルバ定義フローが定義されている場合は、ソルバ定義フローは無視されます。ソルバ共通フローが優先されます。

（３）ソルバ共通フロー（Ntp = M or Ntp = N）

１つのプロセスに複数のソルバが割当てられている場合、すべてのソルバに共通に定義されたフローしかプログラムを実行することはできません。

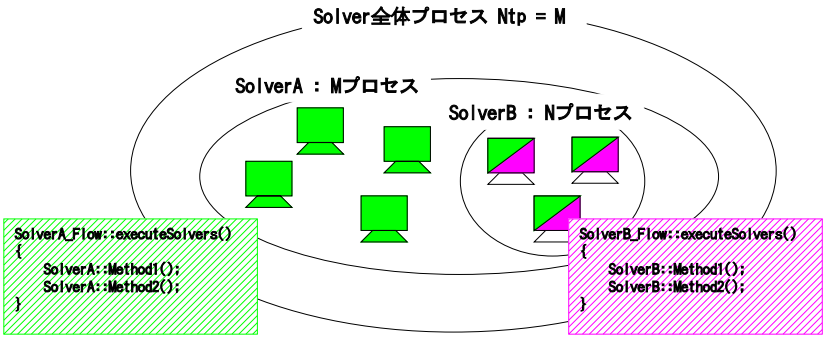


ソルバ毎にソルバ定義フローが定義されている場合は、ソルバ定義フローは無視されます。ソルバ共通フローが優先されます。

（４）フロー定義エラー

１つのプロセスに複数のソルバが存在する場合、２つの異なるフローを定義することはできません。

(エラー)



３．１　デフォルトフロー

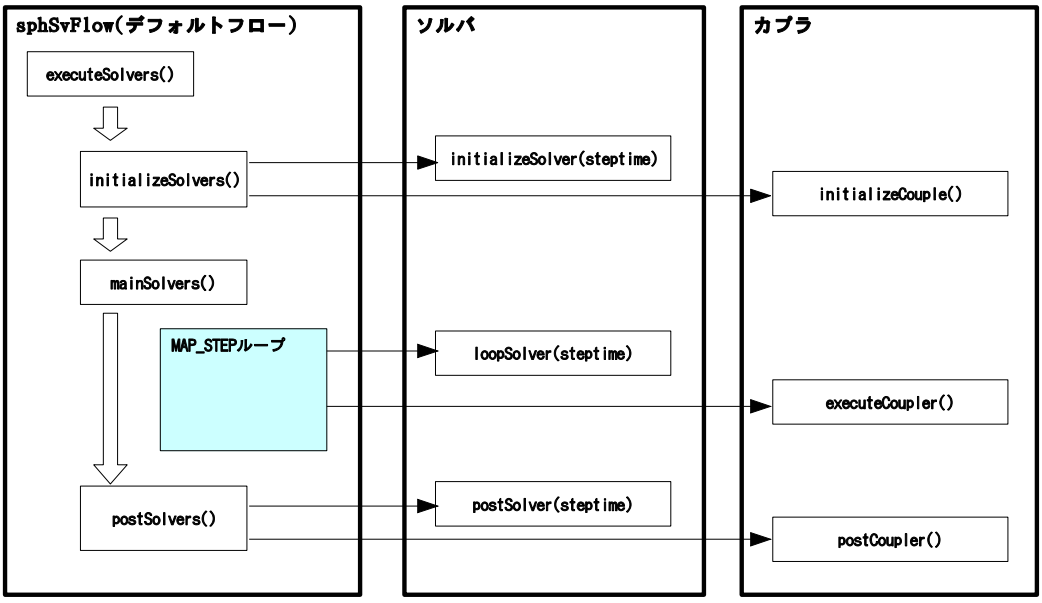
ソルバ定義フロー、ソルバ共通フローが定義されていない場合、Sphere にて作成済みのデフォルトフローにてソルバが実行されます。

（デフォルトフロー実行条件）

ソルバ定義フロー、ソルバ共通フローが定義されていない場合。

"SphFlow[@label='sph_simple_voxel']"要素が定義されている場合。

プロジェクトツールで出力されたユーザフロークラスの変更前。



（１）executeSolver()

ソルバ実行の起点となる関数です。

この関数から以下の関数を呼び出します。

a. initializeSolver()　　：初期化関数

b. mainLoop()　　：メインループ関数
c. postSolver()　　：破棄処置関数

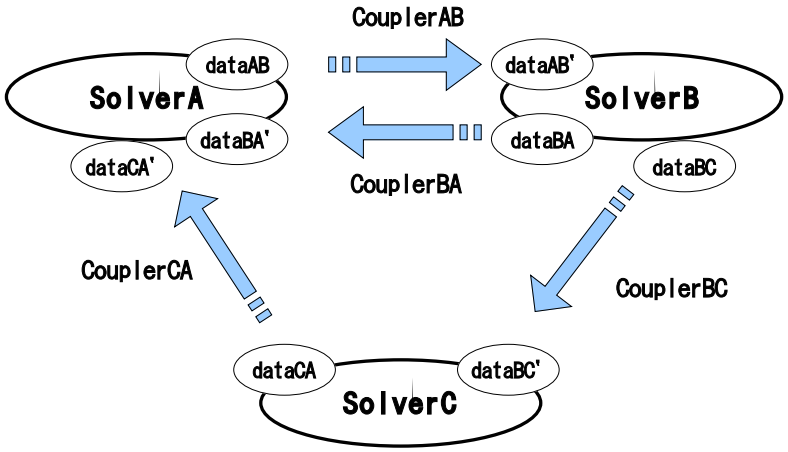
（２）initializeSolver()
ソルバ、カプラの初期化を行います。
ソルバ、カプラの以下の関数を呼び出します。
ソルバ：initializeSolver()
カプラ：initializeCoupler()

（３）mainLoop()
自プロセスに割当済みのソルバに対して MAX_STEP 分ループを繰り返します。
ソルバの呼出順番は、ソルバ ID の小さい順になります。

（４）loopSteps(step, solv_label)
ソルバ、カプラのステップ実行関数を呼び出します。
（ソルバ）
自プロセスに存在するソルバに対して、以下の関数を呼び出します。
ソルバ：loopSolver(step, solv_label)
（カプラ）
直前に実行したソルバが参照ソルバであるカプラに対して以下の関数を呼び出します。
カプラ：loopCoupler(step, solv_label)

（５）postSolver()
ソルバ、カプラの破棄処理を行います。
ソルバ、カプラの以下の関数を呼び出します。
ソルバ：postSolver()
カプラ：postCoupler()

４．　データカプラ
データカプラは複数のソルバの連成を行う為に定義します。



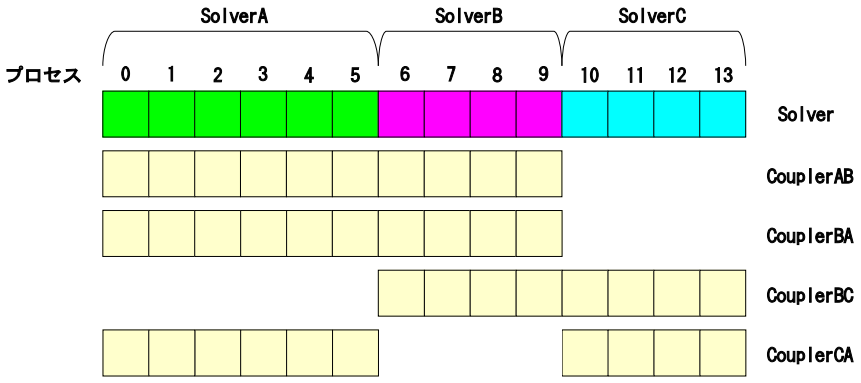
カプラはデータコピーを行う参照情報と更新情報を持っています

参照情報	データの流れ	更新情報
コピー元ソルバ	―＞	コピー先ソルバ
コピー元データ		コピー先データ

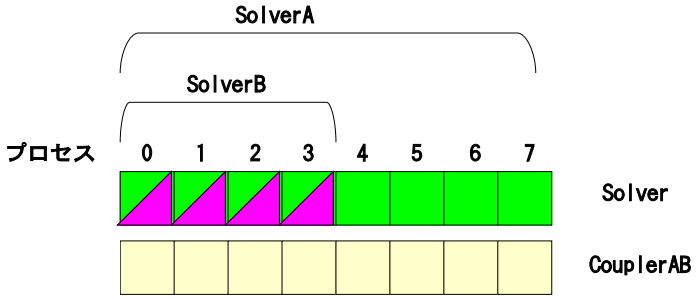
１つのカプラは１つのデータの連成を行いますので、複数のデータ、複数の連成を行う場合は、複数のカプラが必要になります。

参照ソルバ、更新ソルバのプロセスに対してカプラが生成されます。

参照ソルバに生成されたカプラはデータの送信を行い、更新ソルバのカプラはデータの受信を行うことになります。

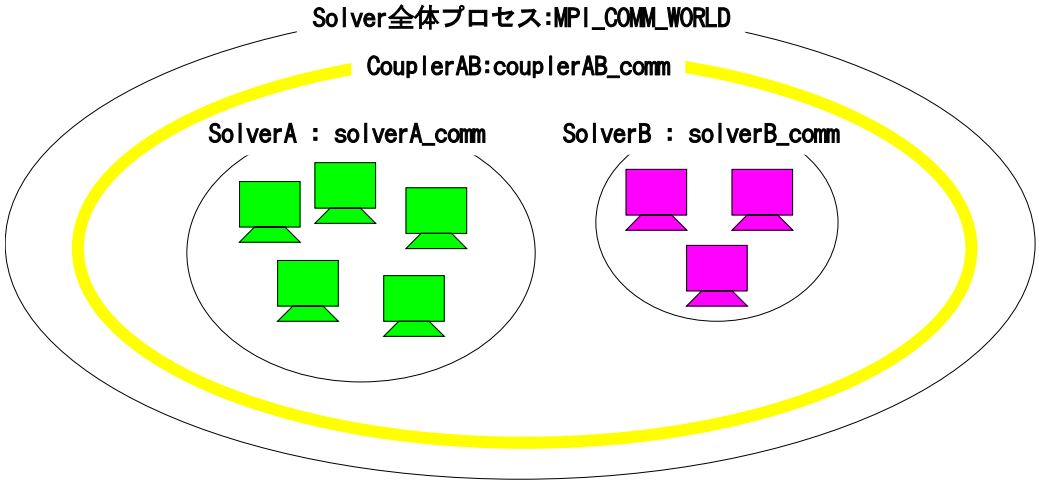


１つのプロセスに参照ソルバ、更新ソルバが共存している場合は、カプラはデータの送受信を１つのカプラで行います。



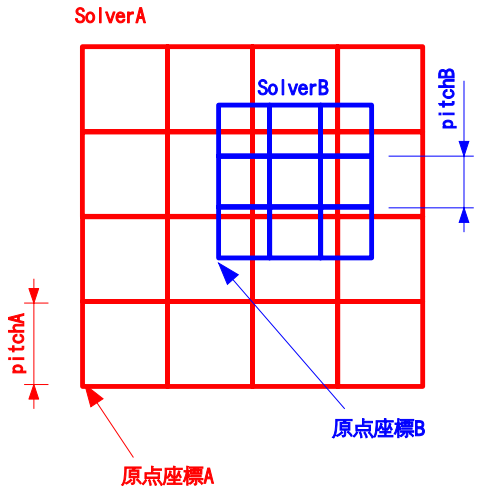
４．１　カプラのグループ、コミュニティー

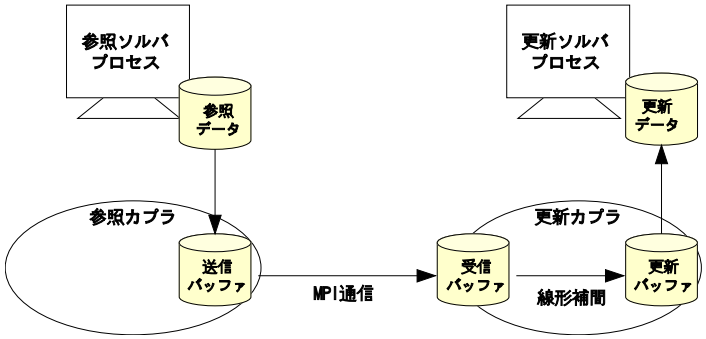
カプラは、参照ソルバ、更新ソルバのプロセスに対してカプラが生成されますので、その参照、更新プロセスを包括する１つのグループ、コミュニティーを作成します。



MPI_COMM_WORLD：Sphere 実行プロセス全体が属する MPI コミュニケータ
solverA_comm：SolverA のプロセスが属する MPI コミュニケータ
solverB_comm：SolverB のプロセスが属する MPI コミュニケータ
couplerAB_comm：SolverA, SolverB のプロセスが属する MPI コミュニケータ

４．２　カプラのデータコピー





（１）参照カプラはソルバの参照データ（コピー元）から送信バッファーを作成します。
送信バッファーは、送信先ノードと重複している領域のサイズを持ち、送信先ノード分、複数作成します。

送信バッファー

送信データ

送信バッファーサイズ

送信データ領域の始点座標

（２）参照カプラから送信バッファーを更新カプラの受信バッファーに送信します。

（３）更新カプラは受信バッファーから線形補間を行い更新バッファーにデータをセットします。

（４）更新バッファーからソルバの更新データ（コピー先）にデータをコピーします。

５．　ソルバの並列実行

並列制御クラス(sphParaController)は、ソルバの計算領域の管理、並列実行における通信管理を行います。
ソルバ毎に自計算領域により生成する必要があります。
SPHERE では並列制御クラスの為に以下のメソッドを提供します。

No	クラス	メソッド名	説明
1	sphSolverSv	virtual int initializeVoxel();	コンフィグレーションに定義された分割方法に従って構造格子用並列制御クラスを生成します。
2		int initializeVoxel(const sphElem* nodelist_elem, const double voxel_origin[3] = NULL, const double voxel_pitch[3] = NULL);	ノードリスト要素によりマルチボックスタイプの領域分割を行った構造格子用並列制御クラスを生成します。
3		int initializeVoxel(const size_t voxelsize[3], const unsigned int division[3], const double voxel_origin[3] = NULL, const double voxel_pitch[3] = NULL);	I,J,K 方向の分割数に応じて均等分割を行い、構造格子用並列制御クラスを生成します。
4		int initializeVoxel(const size_t voxelsize[3], int proc_num = 0, const double voxel_origin[3] = NULL, const double voxel_pitch[3] = NULL);	プロセス数（＝分割数）に応じて自動的に I,J,K 方向に均等分割を行い、構造格子用並列制御クラスを生成します。
5	sphSolverUnst	virtual int initializeUnstMesh();	コンフィグレーションに定義された接点・要素情報に従って非構造格子用並列制御クラスを生成します。

No	クラス	メソッド名	説明
6		virtual int initializeUnstMesh(const char* fn_node, const char* fn_elem, const char* fn_grp = NULL, const char* fn_bnd = NULL, sphUnstMesh::MeshDataType dtype = sphUnstMesh::MESH_DOUBLE);	接点・要素ファイルから接点。要素データを読込み、非構造格子用並列制御クラスを生成します。
7		virtual int initializeUnstMesh(const char* dist_fname, sphUnstMesh::MeshDataType dtype = sphUnstMesh::MESH_DOUBLE);	分散非構造格子ファイルから構造データを読込み、非構造格子用並列制御クラスを生成します。

５．１　コンフィグレーション

SPHERE では、コンフィグレーションに領域・分割情報を定義することにより並列制御クラスを自動生成することが可能です。
以下、領域・分割情報に関するコンフィグレーションについて記述します。
詳細については「コンフィグレーション文法マニュアル」を参照してください。

５．１．１　　SphDomainInfo 要素（領域情報）

記述ルール

要素名	SphDomainInfo	
用　途	計算領域の情報を定義する。	
属　　性		
属性名	値	種別
id	識別 ID（整数）	任意
label	識別名（文字列）	任意
内包可能な要素（子要素として取り得る要素）		
要素名	意味	種別
SphVoxelOrigin	原点座標を定義する。	必須
SphVoxelPitch	ピッチを定義する。	任意 (※1)
SphVoxelSize	計算領域のボクセルサイズを定義する。 （ガイドセルは含まない）	任意
SphStartIndex	始点インデックスを定義する。	任意

	記述がない場合：{０，０，０}	
SphVoxelWidth	計算空間の幅を指定する。	任意 (※1)

（※１）<SphVoxelPitch>, <SphVoxelWidth>要素のどちらか一方は必ず記述しなければなりません。

記述例

```
<SphSolverList>
  <SphSolver>
    <ShpDomainInfo>
      <SphVoxelOrigin ox="-0.5" oy="-0.5" oz="-0.5" />
      < SphVoxelSize   ix="64" jx="64" kx="64" />
      <SphVoxelPitch   dx="1.5625e-02" dy="1.5625e-02" dz="1.5625e-02" />
    </SphDomainInfo>
  </SphSolver>
</SphSolverList>
```

５．１．２　　SphSteer 要素（ソルバの実行パラメータ）

記述ルール

要素名	SphSteer	
用　途	ソルバの実行パラメータを定義する。	
属　　性		
属性名	値	種別
id	識別 ID（整数）	必須
label	識別名（文字列）	任意
内包可能な要素（子要素として取り得る要素）		
要素名	意味	種別
SphDir	ソルバの入出力フォルダの記述	任意
SphNodeList	ノード別のサイズ、始点グローバルインデックスを定義する。	任意
SphVoxelDivision	ノードの分割方法を定義する。	任意
SphNodeProcess	プロセス数	任意
ShpStartCondition	ソルバの開始条件の記述	任意

（注意点）

（１）<SphNodeList>, <SphVoxelDivision>, <SphNodeProcess>によって、プロセス数、

分割方法は決定されるが、優先順位は以下とする。

（高） SphNodeList ＞ SphVoxelDivision ＞ SphNodeProcess （低）

上位の優先順位の要素が記述された場合、下位の要素は無視する。

（２）< SphVoxelDivision>要素、< SphNodeList>要素、<SphNodeProcess>要素が存在しない場合は、ソルバ実行時のプロセス数から均等分割を自動計算する。

５．１．３　　SphNodeProcess 要素（ノードプロセス数）

記述ルール

要素名	SphNodeProcess	
用　途	ソルバの実行プロセス数を定義する。	
属　　性		
属性名	値	種別
value	ソルバの実行プロセス数	必須
内包可能な要素（子要素として取り得る要素）		
要素名	意味	種別
なし		

記述例

```
<SphSolverList>
  <SphSolver>
    <SphSteer>
      < SphNodeProcess   value="8" />
    </SphSteer>
  </SphSolver>
<SphSolverList>
```

５．１．４　　SphVoxelDivision 要素（ノード分割方法）

記述ルール

要素名	SphVoxelDivision	
用　途	ノード分割数を定義する。	
属　　性		
属性名	値	種別
i	I 方向の分割数	必須

j	J 方向の分割数 （２次元モデルの場合は必須）	任意
k	K 方向の分割数 （２次元モデルの場合は必須）	任意
内包可能な要素（子要素として取り得る要素）		
要素名	意味	種別
なし		

記述例

```
<SphSolverList>
  <SphSolver>
    <SphSteer>
      <SphVoxelDivision    i="2" j="1" k="1" />
    </SphSteer>
  </SphSolver>
<SphSolverList>
```

５．１．５　　SphNodeList 要素（ノードリスト）

記述ルール

要素名	SphNodeList
用　途	ノードリストを定義する。
属　　性	
属性名	値	種別
id	識別 ID（整数）	任意
label	識別名（文字列）	任意
内包可能な要素（子要素として取り得る要素）		
要素名	意味	種別
SphNode	ノードの情報を定義する。	任意

記述例

```
<SphSolverList>
  <SphSolver>
    <SphSteer>
```

```
        <SphNodeList>
          <SphNode nodeid="0">
              <ShpSize ix="64" iy="32" iz="32"/>
              <SphStartIndex i="0" j="32" k="32"/>
          </SphNode nodeid>
          <SphNode nodeid="1">
              <ShpSize ix="64" iy="32" iz="32"/>
              <SphStartIndex i="32" j="0" k="0"/>
          </SphNode nodeid>
        </SphNodeList>
      </SphSteer>
    </SphSolver>
<SphSolverList>
```

（TBD）将来的に DFI ファイルによる分割情報の取得に変更予定。

５．２　ソルバ並列制御クラス関係メソッド詳細

５．２．１　ボクセル初期化（コンフィグレーション定義）

virtual int initializeVoxel();	
コンフィグレーションに定義された分割方法に従って構造格子用並列制御クラスを生成します。	
引数	なし
戻り値	-1:error, 0:forced terminated, 1:normality

コンフィグレーションに定義された SphDomainInfo 要素（領域情報）、SphSteer 要素（ソルバの実行パラメータ）から構造格子用の並列制御クラスを生成します。
ソルバ開発者がコンフィグレーションの定義に依存せずに並列制御クラスの生成を行う場合は、"initializeVoxel()"メソッドをソルバ側に定義して、並列制御クラスの生成コードを記述してきださい。

５．２．２　ボクセル初期化（ノードリスト要素指定）

```
int initializeVoxel(
        const sphElem* nodelist_elem,
```

const double voxel_origin[3] = NULL, const double voxel_pitch[3] = NULL);		
ノードリスト要素によりマルチボックスタイプの領域分割を行った構造格子用並列制御クラスを生成します。		
引数	nodelist_elem	ノードリスト要素
	voxel_origin[3]	原点座標
	voxel_pitch[3]	格子ピッチ
戻り値	-1:error, 0:forced terminated, 1:normality	

コンフィグレーションに定義されたノードリスト要素に従ってマルチボックス分割を行い、構造格子用の並列制御クラスを生成します。
（TBD）将来的に DFI ファイルによる分割情報の取得に変更予定。

５．２．３　ボクセル初期化（I,J,K 分割数指定）

int initializeVoxel(const size_t voxelsize[3], const unsigned int division[3], const double voxel_origin[3] = NULL, const double voxel_pitch[3] = NULL);		
I,J,K 方向の分割数に応じて均等分割を行い、構造格子用並列制御クラスを生成します。		
引数	nodelist_elem[3]	ボクセルサイズ
	division[3]	I,J,K 方向分割数
	voxel_origin[3]	原点座標
	voxel_pitch[3]	格子ピッチ
戻り値	-1:error, 0:forced terminated, 1:normality	

I,J,K 方向の分割数に従って均等分割を行い各ノードの計算領域を算出し、構造格子用の並列制御クラスを生成します。

５．２．４　ボクセル初期化（プロセス数指定）

int initializeVoxel(const size_t voxelsize[3],

int proc_num = 0, const double voxel_origin[3] = NULL, const double voxel_pitch[3] = NULL);	
プロセス数（＝分割数）に応じて自動的に I,J,K 方向に均等分割を行い、構造格子用並列制御クラスを生成します。	
引数	nodelist_elem[3] ボクセルサイズ proc_num 分割プロセス数 voxel_origin[3] 原点座標 voxel_pitch[3] 格子ピッチ
戻り値	-1:error, 0:forced terminated, 1:normality

分割プロセス数から I,J,K 方向に計算領域が均等になるように分割を行い、構造格子用の並列制御クラスを生成します。

５．２．５　メッシュ初期化（コンフィグレーション定義）

virtual int initializeUnstMesh();	
コンフィグレーションに定義された接点・要素情報に従って非構造格子用並列制御クラスを生成します。	
引数	なし
戻り値	-1:error, 0:forced terminated, 1:normality

５．２．６　メッシュ初期化（接点・要素ファイル）

virtual int initializeUnstMesh(const char* fn_node, const char* fn_elem, const char* fn_grp = NULL, const char* fn_bnd = NULL, sphUnstMesh::MeshDataType dtype = sphUnstMesh::MESH_DOUBLE);	
接点・要素ファイルから接点。要素データを読込み、非構造格子用並列制御クラスを生成します。	
引数	fn_node 節点ファイル fn_elem 要素ファイル fn_grp グループファイル fn_bnd 境界条件ファイル

	dype　　精度 （sphUnstMesh::MESH_DOUBLE or sphUnstMesh::MESH_FLOAT）
戻り値	-1:error, 0:forced terminated, 1:normality

５．２．７　メッシュ初期化（分散非構造格子ファイル）

virtual int initializeUnstMesh(const char* dist_fname, sphUnstMesh::MeshDataType dtype = sphUnstMesh::MESH_DOUBLE);	
分散非構造格子ファイルから構造データを読込み、非構造格子用並列制御クラスを生成します。	
引数	dist_fname　　分散非構造格子ファイル dype　　精度 （sphUnstMesh::MESH_DOUBLE or sphUnstMesh::MESH_FLOAT）
戻り値	-1:error, 0:forced terminated, 1:normality

５．２．８　並列制御クラス取得

const sphParaController* getParaController() const; sphParaController* getParaController();	
並列制御クラスを取得します	
引数	なし
戻り値	並列制御クラス

５．２．９　並列管理クラス取得

sphParaManager* getParaManager();	
並列管理クラスを取得します	
引数	なし
戻り値	並列管理クラス

５．２．１０MPI コミュニケータ取得

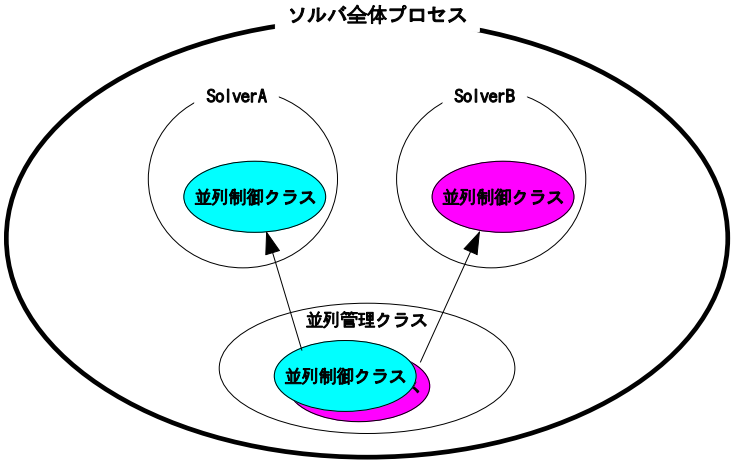
SPL_Comm getSplComm() const;	
ソルバの属する MPI コミュニケータを取得します。	
引数	なし
戻り値	MPI コミュニケータ

５．２．１１MPI グループ取得

SPL_Group getSplGroup() const;	
ソルバの属する MPI グループを取得します。	
引数	なし
戻り値	MPI グループ

６．　並列管理クラス

並列管理クラス(sphParaManager)は並列制御クラス(sphParaController)の生成、管理を行います。
並列管理クラスはソルバ全体プロセスにて１つのみ存在し、ソルバ毎に生成した複数の並列制御クラスを内部にリストにて持ちます。



No	メソッド名	説明

No	メソッド名	説明
1	sphParaController* createSvParaController(const sphElem* solverElem, SPL_Comm comm);	コンフィグレーションの SphSolver 要素を指定して、定義された分割方法に従って構造格子用並列制御クラスを生成します。
2	sphParaController* createSvParaController(int dims, const sphElem* nodelist_elem, const double voxelorigin[3], const double voxelpitch[3], SPL_Comm comm);	ノードリスト要素によりマルチボックスタイプの領域分割を行った構造格子用並列制御クラスを生成します。
3	sphParaController* createSvParaController(const size_t voxelsize[3], const size_t startindex[3], const double voxelorigin[3], const double voxelpitch[3], unsigned int division[3], SPL_Comm comm);	I,J,K 方向の分割数に応じて均等分割を行い、構造格子用並列制御クラスを生成します。
4	sphParaController* createSvParaController(const size_t voxelsize[3], const size_t startindex[3], const double voxelorigin[3], const double voxelpitch[3], unsigned int proc_num, SPL_Comm comm);	プロセス数（＝分割数）に応じて自動的に I,J,K 方向に均等分割を行い、構造格子用並列制御クラスを生成します。
5	sphParaController* createUnstParaController(const sphElem* solverElem, SPL_Comm comm);	コンフィグレーションに定義された接点・要素情報に従って非構造格子用並列制御クラスを生成します。

No	メソッド名	説明
6	sphParaController* createUnstParaController(const sphElem* solverElem, SPL_Comm comm, const char* fn_node, const char* fn_elem, const char* fn_grp = NULL, const char* fn_bnd = NULL, sphUnstMesh::MeshDataType dtype = sphUnstMesh::MESH_DOUBLE);	接点・要素ファイルから接点。要素データを読込み、非構造格子用並列制御クラスを生成します。
7	sphParaController* createUnstParaController(const sphElem* solverElem, SPL_Comm comm, const char* dist_fname, sphUnstMesh::MeshDataType dtype = sphUnstMesh::MESH_DOUBLE);	分散非構造格子ファイルから構造データを読込み、非構造格子用並列制御クラスを生成します。
8	int countOfParaController() const;	登録されている並列制御クラス数を取得します。
9	const sphParaController* getParaControllerByIndex(int id) const;	登録されている並列制御クラスをリスト ID にて取得します。
10	const sphParaController* getParaController(int map_key) const;	登録されている並列制御クラスを登録キー番号にて取得します。

以下、並列管理クラスメソッドの詳細について説明します。

６．１　ボクセル初期化（コンフィグレーション定義）

sphParaController* createSvParaController(const sphElem* solverElem, SPL_Comm comm);
コンフィグレーションの SphSolver 要素を指定して、定義された分割方法に従って構造格子用並列制御クラスを生成します。

引数	configElem　　ソルバコンフィグレーション要素 comm　　コミュニケータ
戻り値	生成構造格子用並列制御クラス

６．２　ボクセル初期化（ノードリスト要素指定）

sphParaController* createSvParaController(int dims, const sphElem* nodelist_elem, const double voxelorigin[3], const double voxelpitch[3], SPL_Comm comm);
ノードリスト要素によりマルチボックスタイプの領域分割を行った構造格子用並列制御クラスを生成します。

引数	dims　　次元数 node_list　　ノードリスト要素 voxelorigin　　原点座標 voxelpitch　　ピッチ comm　　コミュニケータ
戻り値	生成構造格子用並列制御クラス

６．３　ボクセル初期化（ノードリスト要素指定）

sphParaController* createSvParaController(const size_t voxelsize[3], const size_t startindex[3], const double voxelorigin[3], const double voxelpitch[3], unsigned int division[3], SPL_Comm comm);
I,J,K 方向の分割数に応じて均等分割を行い、構造格子用並列制御クラスを生成します。

引数	voxelsize　　ボクセルサイズ startindex　　始点インデックス voxelorigin　　原点座標

	voxelpitch　　ピッチ division　　分割数 comm　　コミュニケータ
戻り値	生成構造格子用並列制御クラス

６．４　ボクセル初期化（ノードリスト要素指定）

sphParaController* createSvParaController(const size_t voxelsize[3], const size_t startindex[3], const double voxelorigin[3], const double voxelpitch[3], unsigned int proc_num, SPL_Comm comm);
プロセス数（＝分割数）に応じて自動的に I,J,K 方向に均等分割を行い、構造格子用並列制御クラスを生成します。

引数	voxelsize　　ボクセルサイズ startindex　　始点インデックス voxelorigin　　原点座標 voxelpitch　　ピッチ proc_num　　プロセス数 comm　　コミュニケータ
戻り値	生成構造格子用並列制御クラス

６．５　メッシュ初期化（コンフィグレーション定義）

sphParaController* createUnstParaController(const sphElem* solverElem, SPL_Comm comm);
コンフィグレーションに定義された接点・要素情報に従って非構造格子用並列制御クラスを生成します。

引数	configElem　　ソルバコンフィグレーション要素 comm　　コミュニケータ
戻り値	生成非構造格子用並列制御クラス

６．６　メッシュ初期化（接点・要素ファイル）

sphParaController* createUnstParaController(const sphElem* solverElem, SPL_Comm comm, const char* fn_node, const char* fn_elem, const char* fn_grp = NULL, const char* fn_bnd = NULL, sphUnstMesh::MeshDataType dtype = sphUnstMesh::MESH_DOUBLE);	
接点・要素ファイルから接点。要素データを読込み、非構造格子用並列制御クラスを生成します。	
引数	configElem　ソルバコンフィグレーション要素 comm　コミュニケータ fn_node　節点ファイル fn_elem　要素ファイル fn_grp　グループファイル fn_bnd　境界条件ファイル dtype　精度
戻り値	生成非構造格子用並列制御クラス

６．７　メッシュ初期化（分散非構造格子ファイル）

sphParaController* createUnstParaController(const sphElem* solverElem, SPL_Comm comm, const char* dist_fname, sphUnstMesh::MeshDataType dtype = sphUnstMesh::MESH_DOUBLE);	
分散非構造格子ファイルから構造データを読込み、非構造格子用並列制御クラスを生成します。	
引数	configElem　ソルバコンフィグレーション要素 comm　コミュニケータ dist_fname　分散非構造格子ファイル dtype　精度
戻り値	生成非構造格子用並列制御クラス

６．８　登録並列制御クラス数取得

int countOfParaController() const;	
登録されている並列制御クラス数を取得します。	
引数	なし
戻り値	登録並列制御クラス数

６．９　登録並列制御クラス数取得

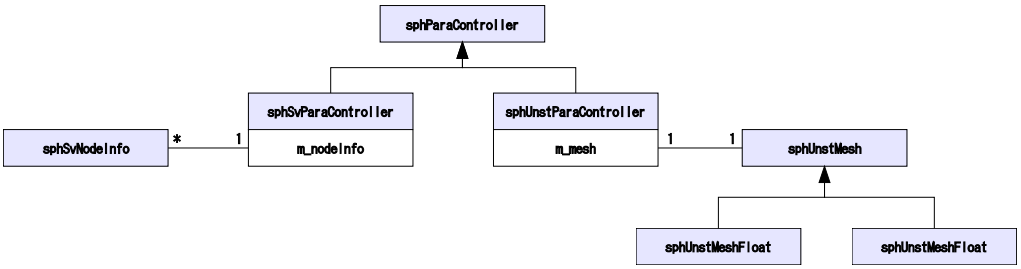
const sphParaController* getParaControllerByIndex(int id) const;	
登録されている並列制御クラスをリスト ID にて取得します。	
引数	id　　リスト ID
戻り値	登録並列制御クラス

６．１０登録並列制御クラス数取得

const sphParaController* getParaController(int map_key) const;	
登録されている並列制御クラスを登録キー番号にて取得します。	
引数	map_key　　登録キー番号（１～）
戻り値	登録並列制御クラス

７．　並列制御クラス

並列制御クラス(sphParaController)は、ソルバの計算領域の管理、並列実行における通信管理を行います。



No	クラス名	説明
1	sphParaController	並列制御クラスの基底クラスです。
2	sphSvParaController	構造格子用の並列制御クラスです。
3	sphUnstParaController	非構造格子用の並列制御クラスです。
4	sphSvNodeInfo	ノード毎のボクセル情報、隣接面情報を管理します。
5	sphUnstMesh	非構造格子のメッシュ情報の管理クラスの基底クラスです。
6	sphUnstMeshFloat	単精度非構造格子のメッシュ情報を管理します。
7	sphUnstMeshDouble	倍精度非構造格子のメッシュ情報を管理します。

以下、並列制御クラスのメソッドについて説明します。

７．１　構造格子用制御クラス

構造格子のソルバにおける計算領域の管理、通信管理を行います。

（メソッド一覧）

No	メソッド名	説明
1	bool getVoxelOrigin(double org[3]) const;	計算領域全体の原点座標を取得する。
2	bool getVoxelPitch(double pitch[3]) const;	ピッチを取得する。
3	bool getNodeOrigin(double org[3]) const;	自ノードの原点座標を取得する。
4	virtual const size_t* getWholeVoxelSize() const;	プロセスグループ内のすべてのノードを結合したサイズを取得する。
5	virtual const size_t* getVoxelSize(int nodeId = -1) const;	ノードサイズを取得する。
6	virtual void getVoxelStartIndex(size_t sta_idx[3]) const;	自ノードの始点インデックスを取得する。
7	const unsigned int* getVoxelDivInfo() const;	分割情報を取得する
8	bool isEv() const; bool isMb() const;	計算領域の分割方法を取得する。
9	int getNumberOfNode() const;	ノード情報クラスリストのノード数を取得する。
10	size_t getVoxelHeadIndex(int nodeId, unsigned int dirId) const;	始点インデックスを取得する。

No	メソッド名	説明
11	size_t getVoxelTailIndex(int nodeId, unsigned int dirId) const;	終点インデックスを取得する。
12	int getCommIDNum(int i, int j, int k) const;	指定方向のノード面に隣接するノード数を取得する。
13	const int* getCommIDs (int i, int j, int k) const;	指定方向のノード面に隣接するノードリストを取得する。
14	int isCommID(int i, int j, int k) const;	指定方向のノード面に隣接するノードの有無・最外郭を取得する。
15	int getCommIDEv (int i, int j, int k) const;	均等分割の場合、指定方向のノード面に隣接するノード番号を取得する。
16	int getCommIDPeriodic (int face, int dir) const;	Periodic通信の最外殻の通信相手ノード番号を取得する。
17	const char* getHostName(int id);	ホスト名を取得する。
18	SPL_Comm getSplComm() const	MPI コミュニケータを取得する。
19	unsigned int getParaKey() const;	並列制御クラス識別キーを取得する。
20	void setSubKey(unsigned int subKey); unsigned int getSubKey() const;	並列制御クラスサブキーを取得・設定する。
21	int getMyID() const;	自ノードの MPI ランク番号を取得する。
22	SPL_Comm createProcessGroup(int idNum, int* idList, SPL_Comm world_comm);	プロセスグループを作成する。
23	unsigned int getNumberOfGroup() const;	プロセスグループ内の実行プロセス数を取得する。
24	unsigned int getNumberOfWorld() const;	すべての実行プロセス数を取得する。
25	int getMyWorldRank() const;	すべてのプロセス内の自ランク番号（ノード番号）を取得する。
26	int getMyRank() const;	プロセスグループ内の自ランク番号（ノード番号）を取得する。
27	bool allreduce(void* sendbuf, void* recvbuf, unsigned int c, SPL_Datatype datatype, SPL_Op operation) const;	ノード間演算を行う。

No	メソッド名	説明
28	bool broadcast(void* data, int c, SPL_Datatype datatype, int id) const;	プロセスグループ内の他のプロセスにデータを送信する。
29	bool barrier() const;	バリア同期を行う。
30	bool send(void* data, int c, SPL_Datatype datatype, int dst) const;	ブロッキングを行い、データ送信を行う。
31	bool recv(void* data, int c, SPL_Datatype datatype, int src) const;	ブロッキングを行い、データ受信を行う。
32	int iSend(void* data, int c, SPL_Datatype datatype, int dst);	ブロッキングを行わず、データ送信を行う。.
33	int iRecv(void* data, int c, SPL_Datatype datatype, int src);	ブロッキングを行わず、データ受信を行う。
34	bool wait(int key);	単一ノンブロッキング通信の完了待ちを行う。
35	bool waitAll(unsigned key_num, int* key_list);	リクエスト登録キーリストのノンブロッキング通信の完了待ちを行う。

No	メソッド名	説明
36	bool waitRequestAll();	完了待ちとなっているすべてのリクエストの完了待ちを行う。
37	void abort(int errcode) const;	MPI 実行環境を終了する。
38	bool gather(void* send_data, int send_cnt, SPL_Datatype send_datatype, void* recv_data, int recv_cnt, SPL_Datatype recv_datatype, int root) const;	１つのプロセスが全プロセスからデータ収集を行う。
39	bool allgather(void* send_data, int send_cnt, SPL_Datatype send_datatype, void* recv_data, int recv_cnt, SPL_Datatype recv_datatype) const;	プロセスグループ内のすべてのプロセスに対しデータ収集を行う。
40	bool createCommGroupByWorld(int idNum, int* idList, SPL_Comm *newComm, SPL_Group *newGrp) const;	新規にプロセスグループを作成する。

No	メソッド名	説明
41	bool createCommGroupByComm(int idNum, int* idList, SPL_Comm oldComm, SPL_Comm *newComm, SPL_Group *newGrp) const;	既存プロセスグループからプロセスグループを作成する。
42	int getMPIDatatypeSize(SPL_Datatype datatype) const;	データタイプのサイズを取得する。

７．１．１　　原点座標の取得

bool getVoxelOrigin(double org[3]) const;	
計算領域全体の原点座標を取得する。	
引数	org　　原点座標
戻り値	成否

計算領域全体の原点座標を取得します。
コンフィグレーションの SphVoxelOrigin 要素に定義されている値となります。

７．１．２　　ピッチの取得

bool getVoxelPitch(double pitch[3]) const;	
ピッチを取得する。	
引数	pitch　　ピッチ
戻り値	成否

格子ピッチを取得します。
コンフィグレーションの SphVoxelPitch 要素に定義されている値となります。

７．１．３　　自ノード原点座標の取得

bool getNodeOrigin(double org[3]) const;

自ノードの原点座標を取得する。	
引数	org　　自ノード原点座標
戻り値	成否

自ノードの原点座標を取得します。

７．１．４　　自ノード始点インデックスの取得

virtual void getVoxelStartIndex(size_t sta_idx[3]) const;	
自ノードの始点インデックスを取得する。	
引数	sta_idx　　自ノード始点インデックス（ガイドセルは含まない）
戻り値	なし

自ノードの始点インデックスを取得します。
"getVoxelHeadIndex()"にて自ノード番号を指定して取得する始点インデックスと同じになります。

７．１．５　　自ノード始点インデックスの取得

const unsigned int* getVoxelDivInfo() const;	
分割情報を取得する。	
引数	なし
戻り値	分割情報

均等分割における I,J,K 方向の分割数を取得します。

戻り値[0]：I 方向分割数

戻り値[1]：J 方向分割数

戻り値[2]：K 方向分割数

コンフィグレーションの SphVoxelDivision 要素(I,J,K 方向分割数)の値と同じになります。
また、プロセス数により計算領域を均等分割した場合も I,J,K 方向の分割数を取得します。

７．１．６　　自ノード始点インデックスの取得

bool isEv() const; bool isMb() const;	
計算領域の分割方法を取得する。	
引数	なし
戻り値	isEv()：true:均等分割 isMb()：true=マルチボックス分割

７．１．７　　ボクセル全体サイズの取得

virtual const size_t* getWholeVoxelSize() const;	
プロセスグループ内のすべてのノードを結合したサイズを取得する。	
引数	なし
戻り値	計算領域全体サイズ（ガイドセルは含まない）

７．１．８　　ノードサイズの取得

virtual const size_t* getVoxelSize(int nodeId = -1) const;	
ノードサイズを取得する。	
引数	nodeid　　ノード ID（省略時自ノード）
戻り値	指定ノードのノードサイズ

７．１．９　　ノード情報クラス数の取得

int getNumberOfNode() const;	
ノード情報クラスリストのノード数を取得する。	
引数	なし
戻り値	ノード情報クラス数

並列制御クラス内のノード情報クラス数を取得します。
"getNumberOfGroup()" によって取得するノード数と同じとなりますが、"getNumberOfGroup()" が "MPI_Comm_size" 関数で取得するノード数に対して、"getNumberOfNode"はノード分割時に作成するノード情報クラスの登録数となります。

７．１．１０　始点インデックスの取得

size_t getVoxelHeadIndex(int nodeId, unsigned int dirId) const;		
始点インデックスを取得する。		
引数	nodeid	ノード ID
	dirId	取得インデックス方向
		0:I 方向, 1:J 方向, 2:K 方向
戻り値	指定ノードの始点インデックス（ガイドセルは含まない）	

７．１．１１　終点インデックスの取得

size_t getVoxelHeadIndex(int nodeId, unsigned int dirId) const;		
終点インデックスを取得する。		
引数	nodeid	ノード ID
	dirId	取得インデックス方向
		0:I 方向, 1:J 方向, 2:K 方向
戻り値	指定ノードの終点インデックス（ガイドセルは含まない）	

７．１．１２　隣接ノード数の取得

int getCommIDNum(int i, int j, int k) const;	
指定方向のノード面に隣接するノード数を取得する。	
引数	i　　I 方向（-1, 0, +1）
	j　　J 方向（-1, 0, +1）
	k　　K 方向（-1, 0, +1）
戻り値	隣接するノード数（０以上）

指定方向の隣接ノード数を取得する。

引数			取得方向
i	j	k	
-1	0	0	-I 方向面
1	0	0	+I 方向面

0	-1	0	-J 方向面
0	1	0	+J 方向面
0	0	-1	-K 方向面
0	0	1	+K 方向面

隣接ノードが存在しない場合は、0 を返します。

７．１．１３　隣接ノードリストの取得

const int* getCommIDs (int i, int j, int k) const;	
指定方向のノード面に隣接するノードリストを取得する。	
引数	i　　I 方向（-1, 0, +1） j　　J 方向（-1, 0, +1） k　　K 方向（-1, 0, +1）
戻り値	隣接するノードリスト

指定方向の隣接ノード ID のリストを取得する。

引数			取得方向
i	j	k	
-1	0	0	-I 方向面
1	0	0	+I 方向面
0	-1	0	-J 方向面
0	1	0	+J 方向面
0	0	-1	-K 方向面
0	0	1	+K 方向面

隣接ノード ID のリスト数は"getCommIDNum"メソッドにより取得する値となります。
隣接ノードが存在しない場合は、NULL を返します。

７．１．１４　隣接ノード有無・最外郭の取得

int　isCommID(int i, int j, int k) const;	
指定方向のノード面に隣接するノードの有無・最外郭を取得する。	
引数	i　　I 方向（-1, 0, +1） j　　J 方向（-1, 0, +1）

	k　　K 方向（-1, 0, +1）
戻り値	隣接ノード ＞0：隣接ノード数 ＝0：隣接ノードなし（ボクセル内側） ＜0：最外殻面

指定方向の隣接ノードの有無・最外郭を取得する。

引数			取得方向
i	j	k	
-1	0	0	-I 方向面
1	0	0	+I 方向面
0	-1	0	-J 方向面
0	1	0	+J 方向面
0	0	-1	-K 方向面
0	0	1	+K 方向面

隣接ノードが存在する場合は"getCommIDNum"メソッドにより取得する値と同じとなります。
指定方向が最外郭の場合は-1 を返します。

７．１．１５　隣接ノード番号の取得（均等分割）

int getCommIDEv (int i, int j, int k) const;	
均等分割の場合、指定方向のノード面に隣接するノード番号を取得する。	
引数	i　　I 方向（-1, 0, +1） j　　J 方向（-1, 0, +1） k　　K 方向（-1, 0, +1）
戻り値	隣接するノード番号 最外殻の場合、-1 を返す。 エラーの場合、-2 を返す。

均等分割の場合、指定方向の隣接ノード番号を取得する。

引数			取得方向
i	j	k	
-1	0	0	-I 方向面

1	0	0	+I 方向面
0	-1	0	-J 方向面
0	1	0	+J 方向面
0	0	-1	-K 方向面
0	0	1	+K 方向面

均等分割でない場合は、-2 を返します。

７．１．１６　Periodic 通信ノード番号の取得

int getCommIDPeriodic (int face, int dir) const;	
Periodic 通信の最外殻の通信相手ノード番号を取得する。	
引数	face　通信面（0:X 面、1:Y 面、2:Z 面） dir　通信方向（-1:マイナス方向、+1:プラス方向）
戻り値	Periodic 通信するノード番号 最外殻ではない場合、-1 を返す。 エラーの場合、-2 を返す。

指定方向の Periodic 通信を行う場合、通信相手のノード番号を取得する。

７．１．１７　ホスト名の取得

const char* getHostName(int　nodeid);	
ホスト名を取得する。	
引数	nodeid　ノード ID
戻り値	ホスト名

７．１．１８　MPI コミュニケータの取得

SPL_Comm getSplComm() const	
MPI コミュニケータを取得する。	
引数	なし
戻り値	MPI コミュニケータ

７．１．１９　識別キーの取得

unsigned int getParaKey()	
並列制御クラス識別キーを取得する。	
引数	なし
戻り値	識別キー

並列管理クラス(sphParaManager)に並列制御クラスの登録時の一意の識別キーを取得する。

７．１．２０　サブキーの取得

void setSubKey(unsigned int subKey); unsigned int getSubKey() const;	
並列制御クラスサブキーを取得・設定する。	
引数	subKey　　サブキー
戻り値	サブキー

サブキーはソルバ開発者によって任意に設定可能な並列制御クラスへ対応させるキー番号です。
媒質 ID 等を設定することを想定しています。

７．１．２１　自ノード番号の取得

int getMyID() const;	
自ノードの MPI ランク番号を取得する。	
引数	なし
戻り値	自ノード番号

"getMyRank()"の取得ランク番号と同じとなります。
"getMyRank()"が"MPI_Comm_rank"関数によって取得しますが、"getMyID"は初期化時に取得したメンバー変数の自ノード番号を返します。

７．１．２２　プロセスグループの作成

SPL_Comm createProcessGroup(int idNum, int* idList, SPL_Comm world_comm);	
プロセスグループを作成する。	
引数	idNum　　プロセスグループノードリスト数 idList　　プロセスグループノードリスト
戻り値	MPI コミュニケータ

７．１．２３　グループ内実行プロセス数の取得

unsigned int getNumberOfGroup() const;	
プロセスグループ内の実行プロセス数を取得する。	
引数	なし
戻り値	プロセス数

実行ソルバのノード数の取得に使用します。
"getNumberOfNode()" によって取得するノード数と同じとなりますが、"getNumberOfGroup()" が "MPI_Comm_size" 関数で取得するノード数に対して、"getNumberOfNode"はノード分割時に作成するノード情報クラスの登録数となります。

７．１．２４　すべての実行プロセス数の取得

unsigned int getNumberOfWorld() const;	
すべての実行プロセス数を取得する	
引数	なし
戻り値	すべての実行プロセス数

"MPI_Comm_size"関数にて"MPI_COMM_WORLD"を指定して実行プロセス数を取得します。

７．１．２５　自 MPI ランク番号の取得（全 MPI グループ）

int getMyWorldRank() const;	
すべてのプロセス内の自ランク番号（ノード番号）を取得する。	
引数	なし
戻り値	自ランク番号（ノード番号） エラー：SPL_PROC_NULL

"MPI_Comm_rank"関数にて"MPI_COMM_WORLD"を指定して自ランク番号を取得します。

７．１．２６　自 MPI ランク番号の取得（MPI グループ）

unsigned int getNumberOfWorld() const;	
プロセスグループ内の自ランク番号（ノード番号）を取得する。	
引数	なし
戻り値	自ランク番号（ノード番号） エラー：SPL_PROC_NULL

"getMyID()"の取得ノード番号と同じとなります。
"getMyRank()"が"MPI_Comm_rank"関数によって取得しますが、"getMyID"は初期化時に取得したメンバー変数の自ノード番号を返します。

７．１．２７　ノード間演算

bool allreduce(void* sendbuf, void* recvbuf,unsigned int c, SPL_Datatype datatype, SPL_Op operation) const;	
ノード間演算を行う。	
引数	sendbuf　送信データ recvbuf　受信データ c　送信データ数 datatype　送信データタイプ operation　演算の種類
戻り値	成否

MPI 実行環境ではない場合、送信データを受信データにコピーする。

７．１．２８　ブロードキャスト

bool broadcast(void* data, int c, SPL_Datatype datatype, int id) const;	
プロセスグループ内の他のプロセスにデータを送信する。	
引数	data　送信/受信データ c　送信/受信データ数 datatype　送信/受信データタイプ id　送信元プロセスのランク
戻り値	成否

７．１．２９　バリア同期

bool barrier() const;	
バリア同期を行う。	
引数	なし
戻り値	成否

７．１．３０　同期データ送信

bool send(void* data, int c, SPL_Datatype datatype, int dst) const;	
ブロッキングを行い、データ送信を行う。	
引数	data　送信データ c　送信データ数 datatype　送信データタイプ dst　送信先ランク番号
戻り値	成否

７．１．３１　同期データ受信

bool recv(void* data, int c, SPL_Datatype datatype, int src) const;
ブロッキングを行い、データ受信を行う。

引数	data　　受信データ c　　受信データ数 datatype　受信データタイプ src　　送信元ランク番号
戻り値	成否

７．１．３２　非同期データ送信

int iSend(void* data, int c, SPL_Datatype datatype, int dst);	
ブロッキングを行わず、データ送信を行う。	
引数	data　　送信データ c　　送信データ数 datatype　送信データタイプ dst　　送信先ランク番号
戻り値	リクエスト登録キー(>0:登録キー, <=0:送信/登録失敗)

戻り値のリクエスト登録キーは並列制御クラス内部で一意に作成した番号である。
MPI_ISEND 関数の MPI_Requset ではなく、MPI_Requset と１対１に対応させる番号である。

７．１．３３　非同期データ受信

bool iRecv(void* data, int c, SPL_Datatype datatype, int src) const;	
ブロッキングを行わず、データ受信を行う。	
引数	data　　受信データ c　　受信データ数 datatype　受信データタイプ src　　送信元ランク番号
戻り値	リクエスト登録キー(>0:登録キー, <=0:送信/登録失敗)

戻り値のリクエスト登録キーは並列制御クラス内部で一意に作成した番号である。
MPI_IRECV 関数の MPI_Requset ではなく、MPI_Requset と１対１に対応させる番号である。

７．１．３４　通信完了待ち（単一）

bool wait(int key);	
単一ノンブロッキング通信の完了待ちを行う	
引数	key　　リクエスト登録キー
戻り値	成否

iSend, iRecv の戻り値のリクエスト登録キーを指定する。

７．１．３５　通信完了待ち（複数）

bool waitAll(unsigned key_num, int* key_list);	
リクエスト登録キーリストのノンブロッキング通信の完了待ちを行う。	
引数	key_num　　キーリスト数 key_list　　リクエスト登録キーリスト
戻り値	成否

iSend, iRecv の戻り値のリクエスト登録キーの数、リストを指定する。

７．１．３６　通信完了待ち（すべて）

bool waitRequestAll();	
リクエスト登録キーリストのノンブロッキング通信の完了待ちを行う。	
引数	なし
戻り値	成否

iSend, iRecv の呼び出し後、完了待ちとなっているすべての通信の完了待ちを行う。

７．１．３７　強制終了

void abort(int errcode) const;
MPI 実行環境を終了する。

引数	errcode　　エラー番号
戻り値	なし

７．１．３８　データ収集（単一プロセス）

bool gather(　　void* send_data, int send_cnt, SPL_Datatype send_datatype, void* recv_data, int recv_cnt, SPL_Datatype recv_datatype, int root) const;	
１つのプロセスが全プロセスからデータ収集を行う。	
引数	send_data　　送信データ send_cnt　　送信データ数 send_datatype　送信データタイプ recv_data　　受信データ recv_cnt　　受信データ数 recv_datatype　受信データタイプ root　　受信ランク番号
戻り値	成否

７．１．３９　データ収集（全プロセス）

bool allgather(　　void* send_data, int send_cnt, SPL_Datatype send_datatype, void* recv_data, int recv_cnt, SPL_Datatype recv_datatype) const;	
プロセスグループ内のすべてのプロセスに対しデータ収集を行う。	
引数	send_data　　送信データ send_cnt　　送信データ数 send_datatype　送信データタイプ

	recv_data　　受信データ recv_cnt　　受信データ数 recv_datatype　受信データタイプ
戻り値	成否

MPI 実行環境ではない場合、送信データを受信データにコピーする。

７．１．４０　MPI プロセスグループ作成（新規作成）

bool createCommGroupByWorld(int idNum, int* idList, SPL_Comm *newComm, SPL_Group *newGrp) const;	
新規にプロセスグループを作成する。	
引数	idNum　　作成ノード数 idList　　作成ノードリスト newComm　　新規作成コミュニケータ newGrp　　新規作成グループ
戻り値	作成 MPI コミュニケータ

ずべてのプロセスグループから新規に MPI プロセスグループを作成する。

７．１．４１　MPI プロセスグループ作成（既存作成）

bool createCommGroupByComm(int idNum, int* idList, SPL_Comm oldComm, SPL_Comm *newComm, SPL_Group *newGrp) const;	
既存プロセスグループからプロセスグループを作成する。	
引数	idNum　　作成ノード数 idList　　作成ノードリスト

	oldComm　　作成済み MPI コミュニケータ newComm　　新規作成コミュニケータ newGrp　　新規作成グループ
戻り値	作成 MPI コミュニケータ

作成済み MPI プロセスグループから MPI プロセスグループを作成する。

７．１．４２　データサイズ取得

int getMPIDatatypeSize(SPL_Datatype datatype) const;	
データタイプのサイズを取得する。	
引数	datatype　　データタイプ
戻り値	データサイズ

MPI 定義済みデータ型のデータサイズを取得する。