

非同期通信

東京大学情報基盤センター 准教授 片桐孝洋

2015年12月22日(火) 10:25-12:10

スパコンプログラミング(1)(I)



東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

講義日程（工学部共通科目）

レポートおよびコンテスト課題
(締切:
2016年2月11日(木)24時 厳守

~~1. 10月6日: ガイダンス~~

~~2. 10月13日~~

- ~~● 並列数値処理の基本演算(座学)~~

~~3. 10月20日: スパコン利用開始~~

- ~~● ログイン作業、テストプログラム実行~~

~~4. 10月27日~~

- ~~● 高性能演算技法1
(ループアンローリング)~~

~~5. 11月10日~~

- ~~● 高性能演算技法2
(キャッシュブロック化)~~

~~6. 11月24日~~

- ~~● 行列ベクトル積の並列化~~

~~7. 12月1日(8:30-10:15)~~

- ~~● ベキ乗法の並列化~~

~~8. 12月1日(10:25-12:10)~~

- ~~● 行列・行列積の並列化(1)~~

~~9. 12月8日~~

- ~~● 行列・行列積の並列化(2)~~

~~10. 12月15日~~

- ~~● LU分解法(1)~~
- ~~● コンテスト課題発表~~

9. 12月22日

- 計算機保守のため座学
- ソフトウェア自動チューニング
- 非同期通信

10. 2016年1月5日

- LU分解法(2)

12. 1月12日

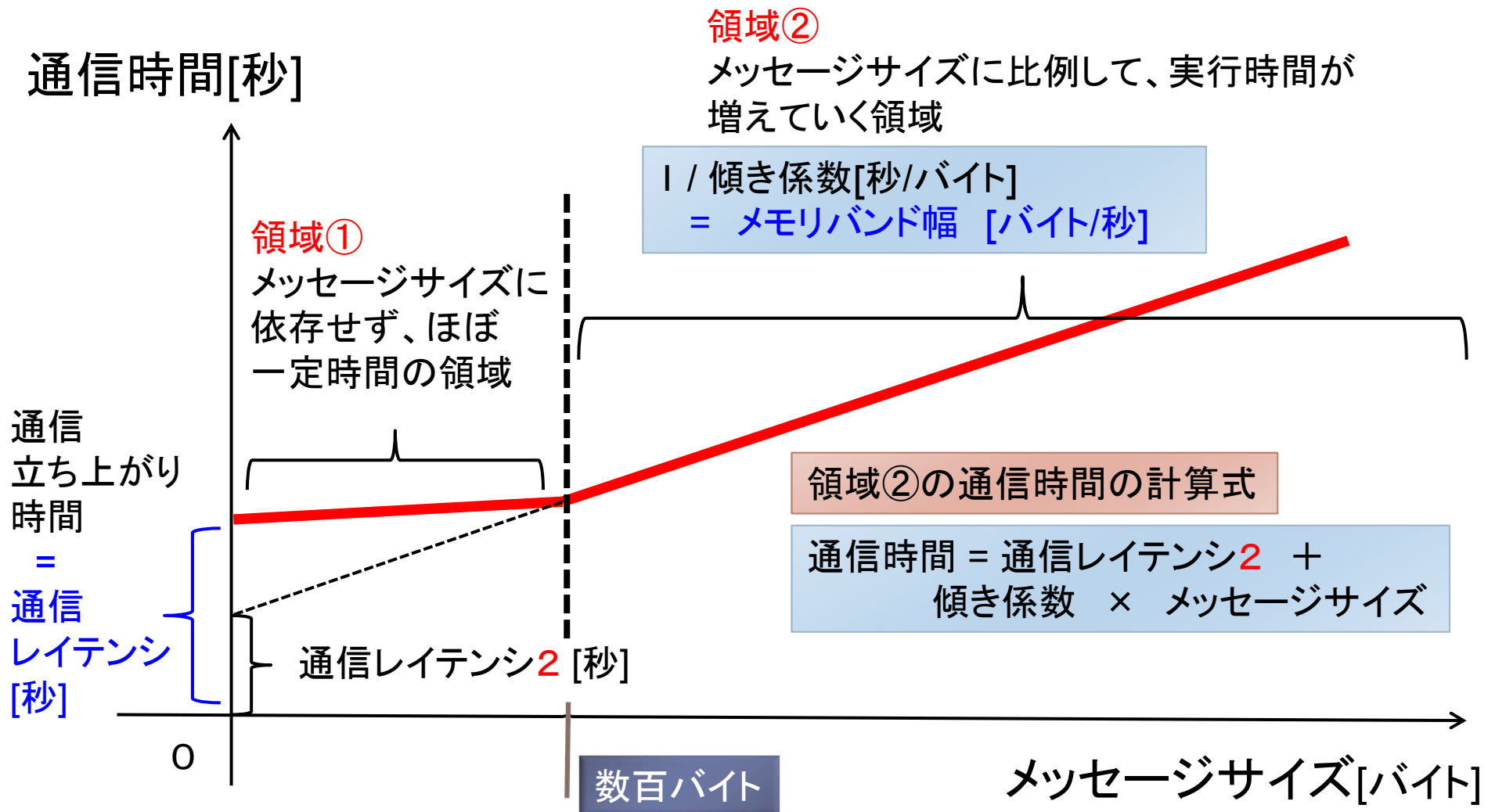
- LU分解法(3)

講義の流れ

1. 1対1通信に関するMPI用語
2. サンプルプログラム(非同期通信)の実行
3. レポート課題

通信最適化の方法

メッセージサイズと通信回数



通信最適化時に注意すること（その1）

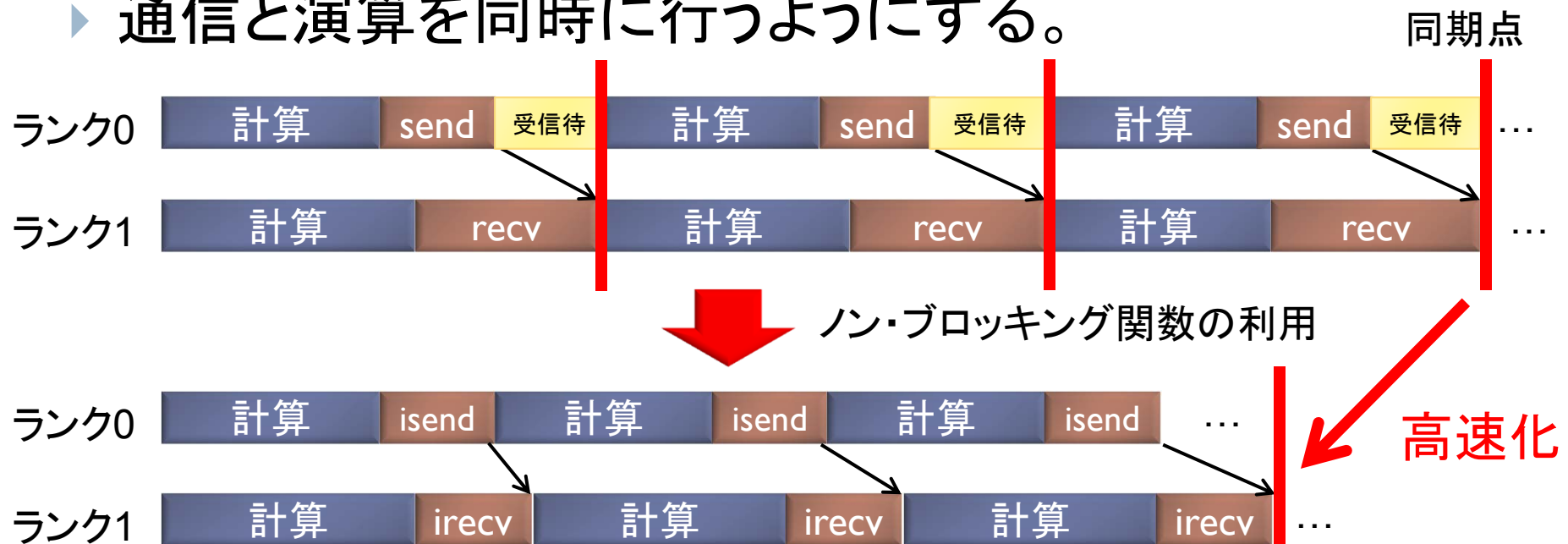
- ▶ 自分のアプリケーションの通信パターンについて、以下の観点を知らないと通信の最適化ができない
 - ▶ <領域①><領域②>のどちらになるのか
 - ▶ 通信の頻度(回数)はどれほどか
- ▶ **領域①の場合**
 - ▶ 「通信レイテンシ」が実行時間のほとんど
 - ▶ 通信回数を削減する
 - ▶ 細切れに送っているデータをまとめて1回にする、など
- ▶ **領域②の場合**
 - ▶ 「メッセージ転送時間」が実行時間のほとんど
 - ▶ メッセージサイズを削減する
 - ▶ 冗長計算をして計算量を増やしてでもメッセージサイズを削減する、など

領域①となる通信の例

- ▶ 内積演算のためのリダクション(MPI_Allreduce)などの送信データは倍精度1個分(8バイト)
- ▶ 8バイトの規模だと、数個分を同時にMPI_Allreduceする時間と、1個分をMPI_Allreduceをする時間は、ほぼ同じ時間となる
 - ▶ ⇒複数回分の内積演算を一度に行うと高速化される可能性あり
- ▶ 例)連立一次方程式の反復解法CG法中の内積演算
 - ▶ 通常の実装だと、1反復に3回の内積演算がある
 - ▶ このため、内積部分は通信レイテンシ律速となる
 - ▶ k反復を1度に行えば、内積に関する通信回数は1/k回に削減
 - ▶ ただし、単純な方法では、丸め誤差の影響で収束しない。
 - ▶ 通信回避CG法(Communication Avoiding CG, CACG)として現在活発に研究されている。

通信最適化時に注意すること（その2）

- ▶ 「同期点」を減らすことも高速化につながる
- ▶ MPI関数の「ノン・ブロッキング関数」を使う
- ▶ 例：ブロッキング関数 `MPI_SEND()`
→ ノン・ブロッキング関数 `MPI_ISEND()`
- ▶ 通信と演算を同時に行うようにする。

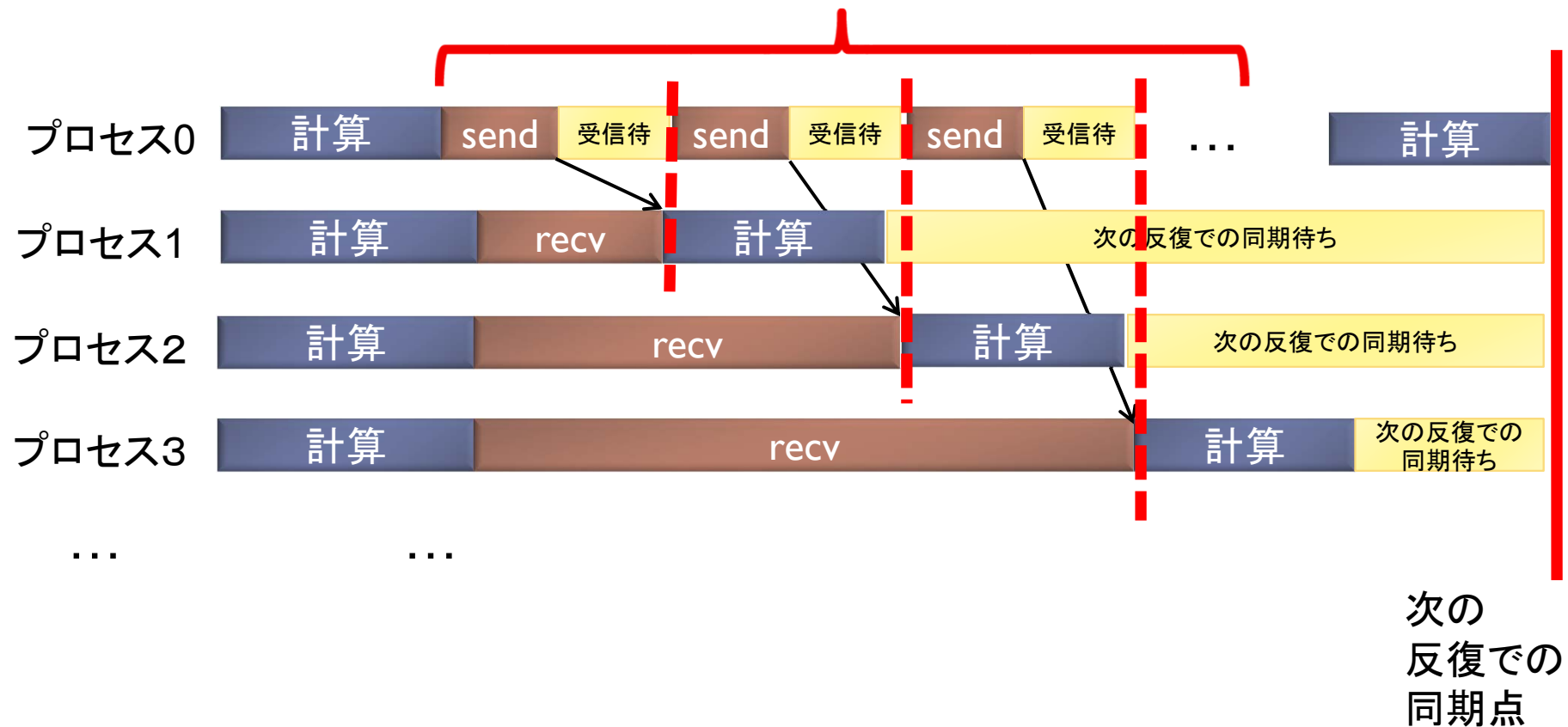


非同期通信： Isend、Irecv、永続的通信関数

ブロッキング通信で効率の悪い例

▶ プロセス0が必要なデータを持っている場合

連続するsendで、効率の悪い受信待ち時間が多発



1 対 1 通信に対する MPI 用語

ブロッキング？ノンブロッキング？

ブロッキング、ノンブロッキング

1. ブロッキング

- ▶ 送信／受信側のバッファ領域にメッセージが格納され、受信／送信側のバッファ領域が自由にアクセス・上書きできるまで、呼び出しが戻らない
- ▶ バッファ領域上のデータの一貫性を保障

2. ノンブロッキング

- ▶ 送信／受信側のバッファ領域のデータを保障せずすぐに呼び出しが戻る
- ▶ バッファ領域上のデータの一貫性を保障せず
 - ▶ 一貫性の保証はユーザの責任

ローカル、ノンローカル

▶ ローカル

- ▶ 手続きの完了が、それを実行しているプロセスのみに依存する。
- ▶ ほかのユーザプロセスとの通信を必要としない処理。

▶ ノンローカル

- ▶ 操作を完了するために、別のプロセスでの何らかのMPI手続きの実行が必要かもしれない。
- ▶ 別のユーザプロセスとの通信を必要とするかもしれない処理。

通信モード（送信発行時の場合）

1. 標準通信モード（ノンローカル）：デフォルト
 - ▶ 送出メッセージのバッファリングはMPIに任せる。
 - ▶ バッファリングされるとき：相手の受信起動前に送信を完了可能；
 - ▶ バッファリングされないとき：送信が完全終了するまで待機；
2. バッファ通信モード（ローカル）
 - ▶ 必ずバッファリングする。バッファ領域がないときはエラー。
3. 同期通信モード（ノンローカル）
 - ▶ バッファ領域が再利用でき、かつ、対応する受信／送信が開始されるまで待つ。
4. レディ通信モード（処理自体はローカル）
 - ▶ 対応する受信が既に発行されている場合のみ実行できる。それ以外はエラー。
 - ▶ ハンドシェイク処理を無くせるため、高い性能を発揮する。

実例－MPI_Send

▶ MPI_Send関数

▶ ブロッキング

▶ 標準通信モード(ノンローカル)

- ▶ バッファ領域が安全な状態になるまで戻らない

- ▶ **バッファ領域がとれる場合:**

メッセージがバッファリングされる。対応する受信が起動する前に、送信を完了できる。

- ▶ **バッファ領域がとれない場合:**

対応する受信が発行されて、かつ、メッセージが受信側に完全にコピーされるまで、送信処理を完了できない。

非同期通信関数

```
▶ ierr = MPI_Isend(sendbuf, icount, datatype,  
    idest, itag,  icomm, irequest);
```

- ▶ **sendbuf** : 送信領域の先頭番地を指定する
- ▶ **icount** : 整数型。送信領域のデータ要素数を指定する
- ▶ **datatype** : 整数型。送信領域のデータの型を指定する
- ▶ **idest** : 整数型。送信したいPEのicomm 内でのランクを指定する
- ▶ **itag** : 整数型。受信したいメッセージに付けられたタグの値を指定する

非同期通信関数

- ▶ **icomm** : 整数型。PE集団を認識する番号であるコミュニケータを指定する。
 - 通常ではMPI_COMM_WORLD を指定すればよい。
- ▶ **irequest** : MPI_Request型 (整数型の配列)。送信を要求したメッセージにつけられた識別子が戻る。
- ▶ **ierr** : 整数型。エラーコードが入る。

同期待ち関数

▶ `ierr = MPI_Wait(irequest, istatus);`

- ▶ `irequest` : `MPI_Request`型(整数型配列)。送信を要求したメッセージにつけられた識別子。
- ▶ `istatus` : `MPI_Status`型(整数型配列)。受信状況に関する情報が入る。
 - ▶ 要素数が`MPI_STATUS_SIZE`の整数配列を宣言して指定する。
 - ▶ 受信したメッセージの送信元のランクが`istatus[MPI_SOURCE]`、タグが`istatus[MPI_TAG]`に代入される。

実例－MPI_Isend

▶ MPI_Isend関数

- ▶ ノンブロッキング
- ▶ 標準通信モード(ノンローカル)
 - ▶ 通信バッファ領域の状態にかかわらず戻る
 - ▶ バッファ領域がとれる場合は、メッセージがバッファリングされ、対応する受信が起動する前に、送信処理が完了できる
 - ▶ バッファ領域がとれない場合は、対応する受信が発行され、メッセージが受信側に完全にコピーされるまで、送信処理が完了できない
- ▶ MPI_Wait関数が呼ばれた場合の振舞いと理解すべき。

注意点

- ▶ 以下のように解釈してください:
 - ▶ **MPI_Send**関数
 - ▶ 関数中に**MPI_Wait**関数が入っている;
 - ▶ **MPI_Isend**関数
 - ▶ 関数中に**MPI_Wait**関数が入っていない;
 - ▶ かつ、すぐにユーザプログラム戻る;

並列化の注意 (MPI_Send, MPI_Recv)

- ▶ 全員がMPI_Sendを先に発行すると、その場所で処理が止まる。(cf. 標準通信モードを考慮)
(正確には、動いたり、動かなかったり、する)
 - ▶ MPI_Sendの処理中で、場合により、バッファ領域がなくなる。
 - ▶ バッファ領域が空くまで待つ(スピンウェイトする)。
 - ▶ しかし、送信側バッファ領域不足から、永遠に空かない。
 - ▶ これを回避するためには、例えば以下の実装を行う。
 - ▶ ランク番号が2で割り切れるプロセス:
 - ▶ MPI_Send();
 - ▶ MPI_Recv();
 - ▶ それ以外:
 - ▶ MPI_Recv();
 - ▶ MPI_Send();
- それぞれに対応
-

非同期通信 TIPS

- ▶ メッセージを完全に受け取ることなく、受信したメッセージの種類を確認したい
 - ▶ 送信メッセージの種類により、受信方式を変えたい場合
 - ▶ MPI_Probe 関数（ブロッキング）
 - ▶ MPI_Iprobe 関数（ノンブロッキング）
 - ▶ MPI_Cancel 関数（ノンブロッキング、ローカル）

MPI_Probe 関数

```
▶ ierr = MPI_Probe(isource, itag, ictomm,  
                   istatus) ;
```

- ▶ **isource**: 整数型。送信元のランク。
 - ▶ MPI_ANY_SOURCE (整数型)も指定可能
- ▶ **itag**: 整数型。タグ値。
 - ▶ MPI_ANY_TAG (整数型) も指定可能
- ▶ **ictomm**: 整数型。コミュニケーター。
- ▶ **istatus**: ステータスオブジェクト。
- ▶ isource, itagに指定されたものがある場合のみ戻る

MPI_Iprobe関数

```
▶ ierr = MPI_Iprobe(isource, itag, ictomm,  
                    iflag, istatus) ;
```

- ▶ **isource**: 整数型。送信元のランク。
 - ▶ MPI_ANY_SOURCE (整数型) も指定可能。
- ▶ **itag**: 整数型。タグ値。
 - ▶ MPI_ANY_TAG (整数型) も指定可能。
- ▶ **ictomm**: 整数型。コミュニケータ。
- ▶ **iflag**: 論理型。isource, itagに指定されたものがあつた場合はtrueを返す。
- ▶ **istatus**: ステータスオブジェクト。

MPI_Cancel 関数

```
▶ ierr = MPI_Cancel(irequest);
```

- ▶ **irequest**: 整数型。通信要求(ハンドル)
- ▶ 目的とする通信が実際に取り消される以前に、可能な限りすばやく戻る。
- ▶ 取消しを選択するため、**MPI_Request_free**関数、**MPI_Wait**関数、又は **MPI_Test**関数 (または任意の対応する操作)の呼出しを利用して完了されている必要がある。

ノン・ブロッキング通信例 (C言語)

```
if (myid == 0) {  
    ...  
    for (i=1; i<numprocs; i++) {  
        ierr = MPI_Isend( &a[0], N, MPI_DOUBLE, i,  
                          i_loop, MPI_COMM_WORLD, &irequest[i] );  
    }  
} else {  
    ierr = MPI_Recv( &a[0], N, MPI_DOUBLE, 0, i_loop,  
                   MPI_COMM_WORLD, &istatus );  
}  
  
a[ ]を使った計算処理;  
if (myid == 0) {  
    for (i=1; i<numprocs; i++) {  
        ierr = MPI_Wait(&irequest[i], &istatus);  
    }  
}
```

ランク0のプロセスは、
ランク1~numprocs-1までのプロセス
に対して、ノンブロッキング通信を
用いて、長さNのDouble型配列
データを送信

ランク1~numprocs-1までの
プロセスは、ランク0からの
受信待ち。

ランク0のPEは、
ランク1~numprocs-1までのプロセス
に対するそれぞれの送信に対し、
それぞれが受信完了するまで
ビジーウェイト(スピンウェイト)
する。

プロセス0は、recvを
待たず計算を開始

ノン・ブロッキング通信の例 (Fortran言語)

```
if (myid .eq. 0) then
  ...
  do i=1, numprocs - 1
    call MPI_ISEND( a, N, MPI_DOUBLE_PRECISION,
                   i, i_loop, MPI_COMM_WORLD, irequest, ierr )
  enddo
else
  call MPI_RECV( a, N, MPI_DOUBLE_PRECISION,
                0, i_loop, MPI_COMM_WORLD, istatus, ierr )
endif
a( )を使った計算処理
if (myid .eq. 0) then
  do i=1, numprocs - 1
    call MPI_WAIT(irequest(i), istatus, ierr )
  enddo
endif
```

ランク0のプロセスは、
ランク1~numprocs-1までの
プロセスに対して、ノンブロッキング
通信を用いて、長さNの
DOUBLE PRECISION型配列
データを送信

ランク1~numprocs-1までの
プロセスは、
ランク0からの受信待ち。

プロセス0は、recvを
待たず計算を開始

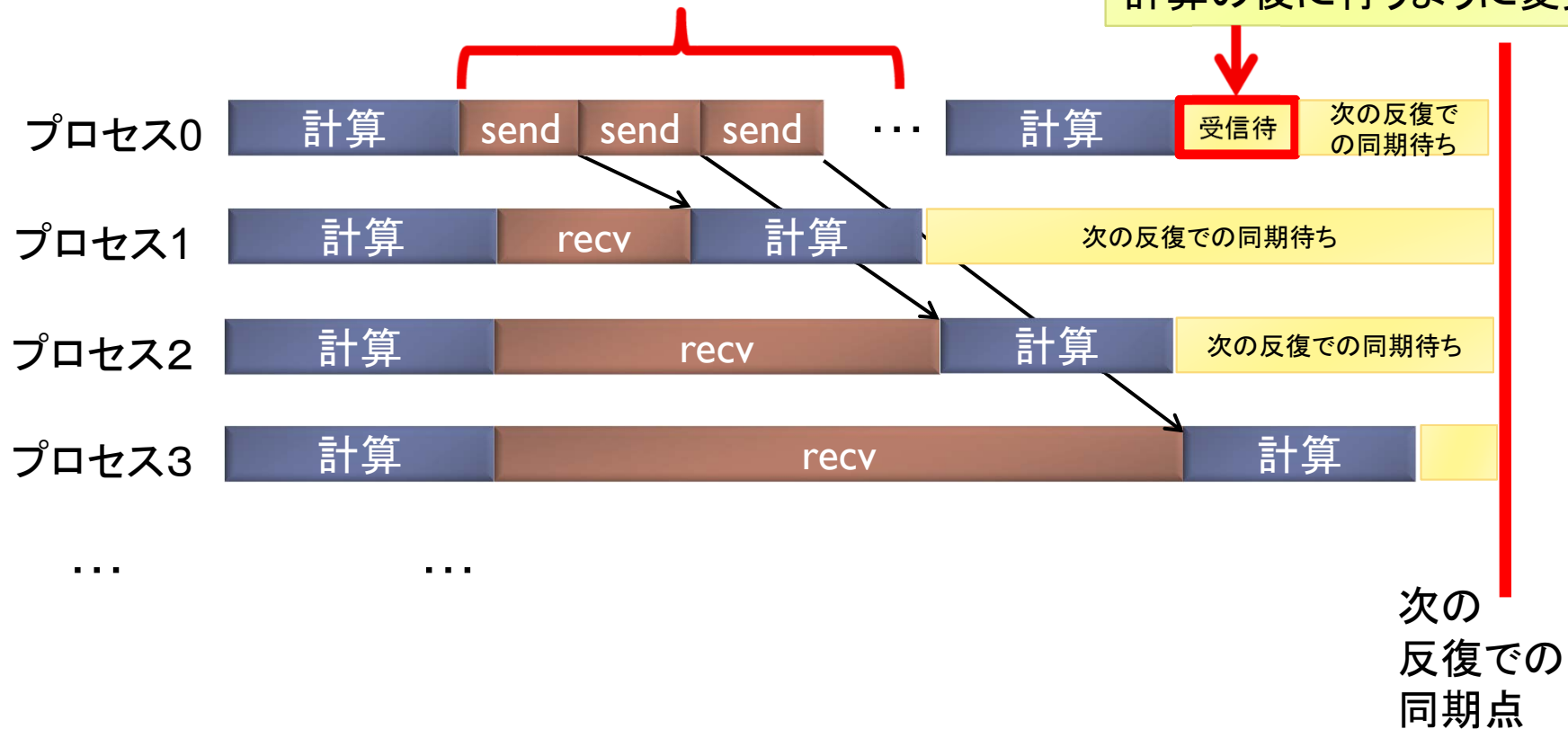
ランク0のプロセスは、
ランク1~numprocs-1までの
プロセスに対するそれぞれの送信
に対し、それぞれが受信完了
するまでビジーウェイト
(スピンウェイト)する。

ノン・ブロッキング通信による改善

▶ プロセス0が必要なデータを持っている場合

連続するsendにおける受信待ち時間を
ノン・ブロッキング通信で削減

受信待ちを、MPI_Waitで
計算の後に行うように変更



永続的通信（その1）

- ▶ ノン・ブロッキング通信は、MPI_ISENDの実装が、MPI_ISENDと呼ばれた時点で本当に通信を開始する実装になっていないと意味がない。
- ▶ ところが、MPIの実装によっては、MPI_WAITが呼ばれるまで、MPI_ISENDの通信を開始しない実装がされていることがある。
 - ▶ この場合には、ノン・ブロッキング通信の効果が全くない。
- ▶ 永続的通信(Persistent Communication)を利用すると、MPIライブラリの実装に依存し、ノン・ブロッキング通信の効果が期待できる場合がある。
 - ▶ 永続的通信は、MPI-1からの仕様(たいていのMPIで使える)
 - ▶ しかし、通信と演算がオーバーラップできる実装になっているかは別問題

永続的通信（その2）

▶ 永続的通信の利用法

1. 通信を利用するループ等に入る前に1度、通信相手先を設定する初期化関数を呼ぶ
2. その後、SENDをする箇所に**MPI_START関数**を書く
3. 真の同期ポイントに使う関数(MPI_WAIT等)は、ISENDと同じものを使う

▶ **MPI_SEND_INIT関数**で通信情報を設定しておく、MPI_START時に通信情報の設定が行われない

- ▶ 同じ通信相手に何度でもデータを送る場合、通常のノン・ブロッキング通信に対し、同等以上の性能が出ると期待

▶ 適用例

- ▶ 領域分割に基づく陽解法
- ▶ 陰解法のうち反復解法を使っている数値解法

永続的通信の実装例 (C言語)

```
MPI_Status istatus;  
MPI_Request irequest;
```

```
...
```

```
if (myid == 0) {
```

```
    for (i=1; i<numprocs; i++) {
```

```
        ierr = MPI_Send_init(a, N, MPI_DOUBLE_PRECISION, i,  
                             0, MPI_COMM_WORLD, irequest);
```

```
    }
```

```
}
```

```
...
```

```
if (myid == 0) {
```

```
    for (i=1; i<numprocs; i++) {
```

```
        ierr = MPI_Start(irequest);
```

```
    }
```

```
}
```

```
/* 以降は、Isendの例と同じ */
```

メインループに入る前に、
送信データの相手先情報を
初期化する

ここで、データを送る

永続的通信の実装例 (Fortran言語)

```
integer istatus(MPI_STATUS_SIZE)
integer irequest(0:MAX_RANK_SIZE)
...
if (myid .eq. 0) then
  do i=1, numprocs-1
    call MPI_SEND_INIT (a, N, MPI_DOUBLE_PRECISION, i,
      0, MPI_COMM_WORLD, irequest(i), ierr)
  enddo
endif
...
if (myid .eq. 0) then
  do i=1, numprocs-1
    call MPI_START (irequest, ierr)
  enddo
endif

/* 以降は、ISENDの例と同じ */
```

メインループに入る前に、
送信データの相手先情報を
初期化する

ここで、データを送る

サンプルプログラムの実行 (非同期通信)

はじめてのMPI_Isend

LU分解のサンプルプログラムの注意点

- ▶ C言語版／Fortran言語版のファイル名

`lsend-fx.tar`

- ▶ ジョブスクリプトファイル`lsend.bash` 中のキュー名を

`lecture` から `lecture4` に変更してから
`pjsub` してください。

- ▶ `lecture` : 実習時間外のキュー
- ▶ `lecture4`: 実習時間内のキュー

MPI_Isendのサンプルプログラムの実行 (C言語版/ Fortran版共通)

- ▶ 以下のコマンドを実行する

```
$ cp /home/z30082/ISend-fx.tar ./
$ tar xvf ISend-fx.tar
$ cd Isend
```
- ▶ 以下のどちらかを実行

```
$ cd C :C言語を使う人
$ cd F :Fortran言語を使う人
```
- ▶ 以下共通

```
$ make
$ pjsub isend.bash
```
- ▶ 実行が終了したら、以下を実行する

```
$ cat isend.bash.oXXXXXXXXX
```

出力結果

- ▶ 以下のような結果が出力される(C言語)

Execution time using MPI_Isend : 30.3248 [sec.]

サンプルプログラムの説明 (C言語版)

```
if (myid == 0) {  
    ...  
    for (i=1; i<numprocs; i++) {  
        ierr = MPI_Isend( &a[0], N, MPI_DOUBLE, i,  
                          i_loop, MPI_COMM_WORLD, &irequest[i] );  
    }  
} else {  
    ierr = MPI_Recv( &a[0], N, MPI_DOUBLE, 0, i_loop,  
                    MPI_COMM_WORLD, &istatus );  
}  
...  
if (myid == 0) {  
    for (i=1; i<numprocs; i++) {  
        ierr = MPI_Wait(&irequest[i], &istatus);  
    }  
}
```

ランク0のPEは、
ランク1~191までのPEに対して、
ノンブロッキング通信を用いて、
長さNのDouble型配列データを
送信

ランク1~191までのPEは、
ランク0からの受信待ち。

ランク0のPEは、
ランク1~191までのPEに対する
それぞれの送信に対し、
それぞれが受信完了するまで
ビジーウェイト(スピンウェイト)
する。

サンプルプログラムの説明 (Fortran言語版)

```
if (myid .eq. 0) then
  ...
  do i=1, numprocs - 1
    call MPI_ISEND( a, N, MPI_DOUBLE_PRECISION,
                   i, i_loop, MPI_COMM_WORLD, irequest, ierr )
  enddo
else
  call MPI_RECV( a, N, MPI_DOUBLE_PRECISION,
                0, i_loop, MPI_COMM_WORLD, istatus, ierr )
endif
...
if (myid .eq. 0) then
  do i=1, numprocs - 1
    call MPI_WAIT(irequest(i), istatus, ierr )
  enddo
endif
```

ランク0のPEは、
ランク1~191までのPEに対して、
ノンブロッキング通信を用いて、
長さNのDOUBLE PRECISION
型配列データを送信

ランク1~191までのPEは、
ランク0からの受信待ち。

ランク0のPEは、
ランク1~191までのPEに対する
それぞれの送信に対し、
それぞれが受信完了するまで
ビジーウェイト(スピンウェイト)
する。

レポート課題（その1）

1. [L5] ブロッキングは同期でないことを説明せよ。
2. [L10] MPIにおけるブロッキング、ノンブロッキング、および通信モードによる分類に対応する関数を調べ、一覧表にまとめよ。
3. [L15] 利用できる並列計算機環境で、ノンブロッキング送信（MPI_Isend関数）がブロッキング送信（MPI_Send関数）に対して有効となるメッセージの範囲（N=0～**適当な上限**）について調べ、結果を考察せよ。
4. [L20] MPI_Allreduce関数 の＜限定機能＞版を、ブロッキング送信、およびノンブロッキング送信を用いて実装せよ。さらに、その性能を比べてみよ。なお、＜限定機能＞は独自に設定してよい。

レポート課題（その2）

5. [L15] `MPI_Reduce`関数を実現するRecursive Halvingアルゴリズムについて、その性能を調査せよ。この際、従来手法も調べて、その手法との比較も行うこと。
6. [L35] Recursive Halvingアルゴリズムを、ブロッキング送信／受信、および、ノンブロッキング送信／受信を用いて実装せよ。また、それらの性能を評価せよ。
7. [L15] 身近の並列計算機環境で、永続的通信関数の性能を調べよ。
8. [L10～] 自分が持っているMPIプログラムに対し、ノンブロッキング通信(`MPI_Isend`, `MPI_Irecv`)を実装し、性能を評価せよ。また永続的通信が使えるプログラムの場合は実装して評価せよ。

発展話題（ソフトウェア自動チューニング）