

MPIによる並列アプリケーション開発 「超」入門 (II)

東京大学情報基盤センター
中島 研吾

第5回お試しアカウント付き並列プログラミング講習会(試行)
2009年7月2日

スケジュール(予定)

- 1300~1415 背景, 一次元熱伝導解析プログラムの概要
- 1415~1545 並列データ構造の考え方
- 1600~1630 一次元熱伝導解析プログラムの並列化
- 1630~ 実習

Jul2009

第二部 概要

- 科学技術計算とMPI
- 1対1通信: 一般化された通信テーブル
 - 一次元問題
 - 二次元問題
- 1対1通信の実装例
- 差分法による一次元熱伝導方程式ソルバーの並列化

3

科学技術計算のためのMPI

- 文法
 - 「MPI-1」の基本的な機能(10程度)について習熟する
 - MPI-2では色々と便利な機能があるが...
 - あとは自分に必要な機能について調べる, あるいは知っている人, 知っていそうな人に尋ねる
- 実習の重要性
 - プログラミング
 - その前にまず実行してみる
- SPMDのオペレーションに慣れること...「つかむ」こと
 - Single Program/Instruction Multiple Data
 - 基本的に各プロセスは「同じことをやる」が「データが違う」
 - 大規模なデータを分割し, 各部分について各プロセス(プロセッサ)が計算する
 - 通信「以外」は単体CPUの場合とほぼ同じ
 - 全体データと局所データ, 全体番号と局所番号

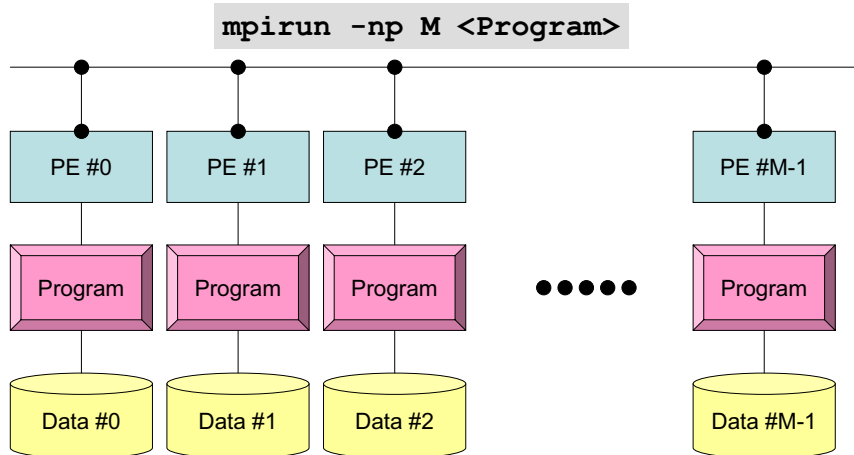
Jul2009

4

PE: Processing Element
プロセッサ, 領域, プロセス

SPMD

この絵が理解できればMPIは
9割方理解できたことになる。



各プロセスは「同じことをやる」が「データが違う」
大規模なデータを分割し、各部分について各プロセス(プロセッサ)が計算する
通信以外は、単体CPUのときと同じ、というのが理想

データ構造とアルゴリズム

- コンピュータ上で計算を行うプログラムはデータ構造とアルゴリズムから構成される。
- 両者は非常に密接な関係にあり、あるアルゴリズムを実現するためには、それに適したデータ構造が必要である。
 - 極論を言えば「データ構造＝アルゴリズム」と言っても良い。
 - もちろん「そうではない」と主張する人もいるが、科学技術計算に関する限り、中島の経験では「データ構造＝アルゴリズム」と言える。
- 並列計算を始めるにあたって、基本的なアルゴリズムに適したデータ構造を定める必要がある。

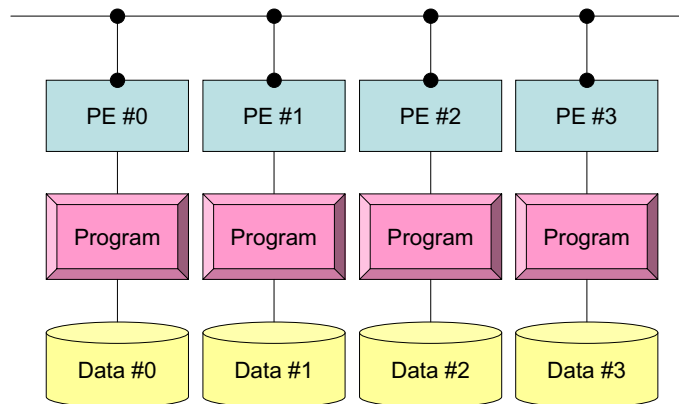
用語

- プロセッサ, コア
 - ハードウェアとしての各演算装置。シングルコアではプロセッサ＝コア
- プロセス
 - MPI計算のための実行単位、ハードウェア的な「コア」とほぼ同義。
 - しかし1つの「プロセッサ・コア」で複数の「プロセス」を起動する場合もある(効率的ではないが)。
- PE (Processing Element)
 - 本来、「プロセッサ」の意味なのであるが、本講義では「プロセス」の意味で使う場合も多い。次項の「領域」とほぼ同義でも使用。
 - マルチコアの場合は:「コア＝PE」という意味で使うことが多い。
- 領域
 - 「プロセス」とほぼ同じ意味であるが、SPMDの「MD」のそれぞれ一つ、「各データ」の意味合いが強い。しばしば「PE」と同義で使用。
- MPIのプロセス番号(PE番号, 領域番号)は0から開始
 - したがって8プロセス(PE, 領域)ある場合は番号は0～7

SPMD: Single Program Multiple Data

- 一言で「並列計算」と言っても色々なものがあり、基本的なアルゴリズムも様々。
- 共通して言えることは、SPMD (Single Program Multiple Data)
- なるべく単体CPUのときと同じようにできることが理想
 - 通信が必要な部分とそうでない部分を明確にする必要がある。

SPMDに適したデータ構造とは？



SPMDに適したデータ構造(1/2)

- 大規模なデータ領域を分割して、各プロセッサ、プロセスで計算するのがSPMDの基本的な考え方
- 例えば長さNG(=20)のベクトルVGに対して以下のような計算を考えてみよう:

```
integer, parameter :: NG= 20
real(kind=8), dimension(20) :: VG

do i= 1, NG
  VG(i)= 2.0 * VG(i)
enddo
```

- これを4つのプロセッサで分担して計算するとすれば、 $20/4=5$ ずつ記憶し、処理すればよい。

SPMDに適したデータ構造(2/2)

- すなわち、こんな感じ:

```
integer, parameter :: NL= 5
real(kind=8), dimension(5) :: VL

do i= 1, NL
  VL(i)= 2.0 * VL(i)
enddo
```

- このようにすれば「一種類の」プログラム (Single Program) で並列計算を実施できる。
 - 各プロセスにおいて、「VL」の中身が違う: Multiple Data
 - 可能な限り計算を「VL」のみで実施することが、並列性能の高い計算へつながる。
 - 単体CPUの場合ともほとんど変わらない。

全体データと局所データ

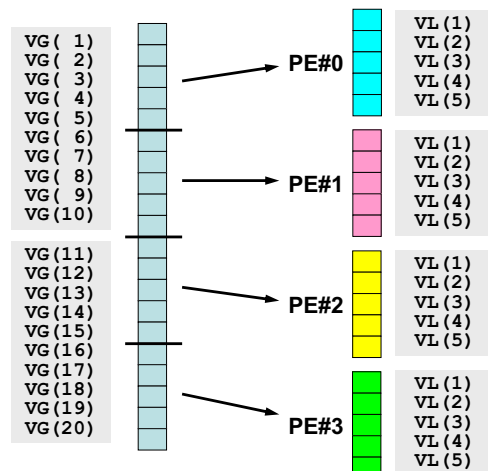
- VG
 - 領域全体
 - 1番から20番までの「全体番号」を持つ「全体データ (Global Data)」
- VL
 - 各プロセス (PE, プロセッサ, 領域)
 - 1番から5番までの「局所番号」を持つ「局所データ (Local Data)」
 - できるだけ局所データを有効に利用することで、高い並列性能が得られる。

局所データの考え方

「全体データ」VGの:

- 1～5番成分が0番PE
- 6～10番成分が1番PE
- 11～15番成分が2番PE
- 16～20番成分が3番PE

のそれぞれ, 「局所データ」VLの1番～5番成分となる(局所番号が1番～5番となる)。



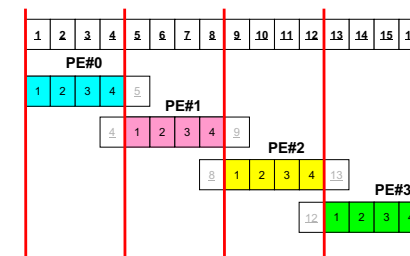
全体データと局所データ

- VG
 - 領域全体
 - 1番から20番までの「全体番号」を持つ「全体データ(Global Data)」
- VL
 - 各プロセッサ
 - 1番から5番までの「局所番号」を持つ「局所データ(Local Data)」
- この講義で常に注意してほしいこと
 - VG(全体データ)からVL(局所データ)をどのように生成するか。
 - VGからVL, VLからVGへデータの中身をどのようにマッピングするか。
 - VLがプロセスごとに独立して計算できない場合はどうするか。
 - できる限り「局所性」を高めた処理を実施する⇒高い並列性能
 - そのための「データ構造」「アルゴリズム」

- 科学技術計算とMPI
- 1対1通信: 一般化された通信テーブル
 - 一次元問題
 - 二次元問題
- 1対1通信の実装例
- 差分法による一次元熱伝導方程式ソルバーの並列化

1対1通信とは ?

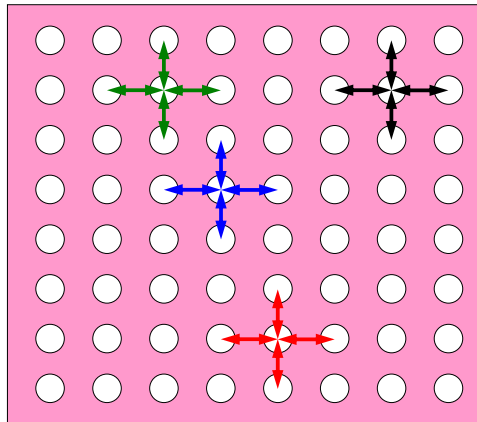
- グループ通信: Collective Communication
 - MPI_Reduce, MPI_Scatter/Gather など
 - 同じコミュニケータ内の全プロセスと通信する
 - 適用分野
 - 境界要素法, スペクトル法, 分子動力学等グローバルな相互作用のある手法
 - 内積, 最大値などのオペレーション
- 1対1通信: Point-to-Point
 - MPI_Send, MPI_Receive
 - 特定のプロセスとのみ通信がある
 - 隣接領域
 - 適用分野
 - 差分法, 有限要素法などローカルな情報を使う手法



グループ通信, 1対1通信

近接PE(領域)のみとの相互作用

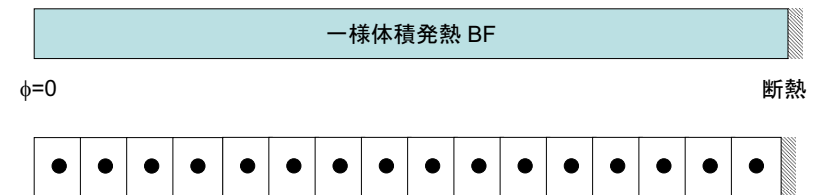
差分法, 有限要素法



一次元熱伝導方程式ソルバーの並列化

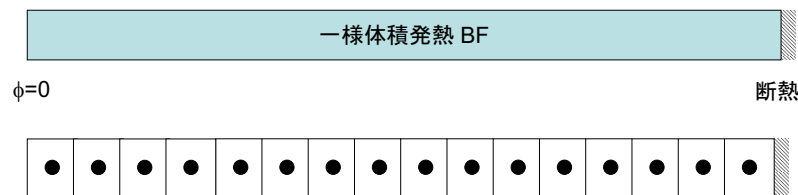
$$\frac{d^2\phi}{dx^2} + BF = 0, \quad \phi = 0 @ x = 0, \quad \frac{d\phi}{dx} = 0 @ x = x_{\max}$$

$$\phi = -\frac{1}{2} BF x^2 + BF x_{\max} x$$



一次元熱伝導方程式ソルバーの並列化

- 全体データと局所データ
- 例えば, NG=16の問題を4PEで実行する場合, 各PEにおいてはNL=16/4=4となる



全体データと局所データ

NG=16, PE#=4

全体番号

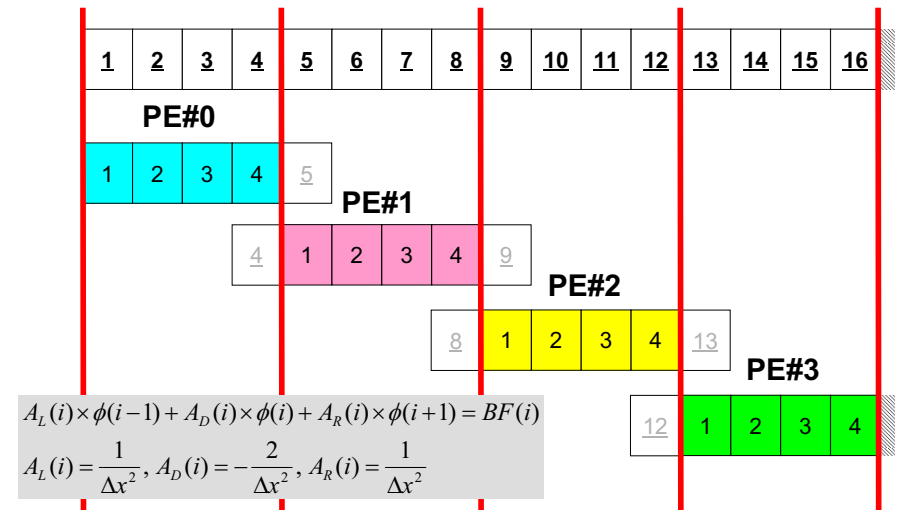
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

全体データと局所データ NG=16, PE#=4

全体番号

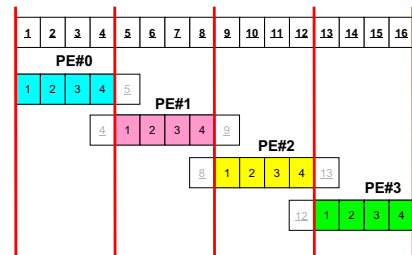


差分法のオペレーションを実施するため には隣接要素の情報が必要

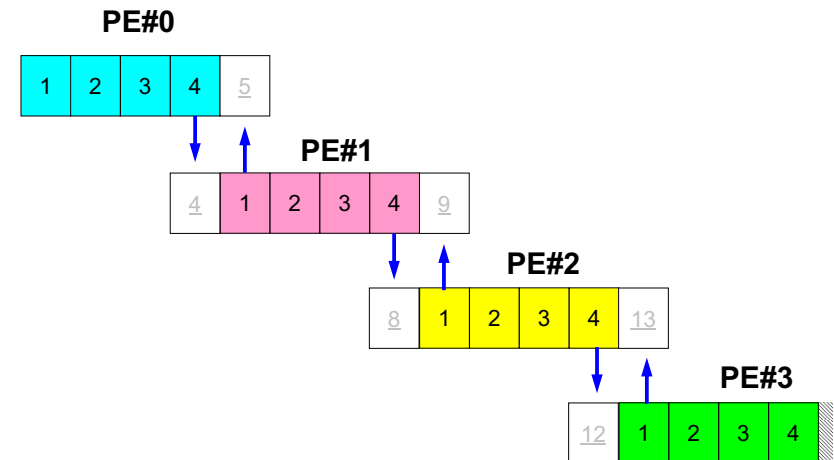


一次元熱伝導方程式ソルバーの並列化

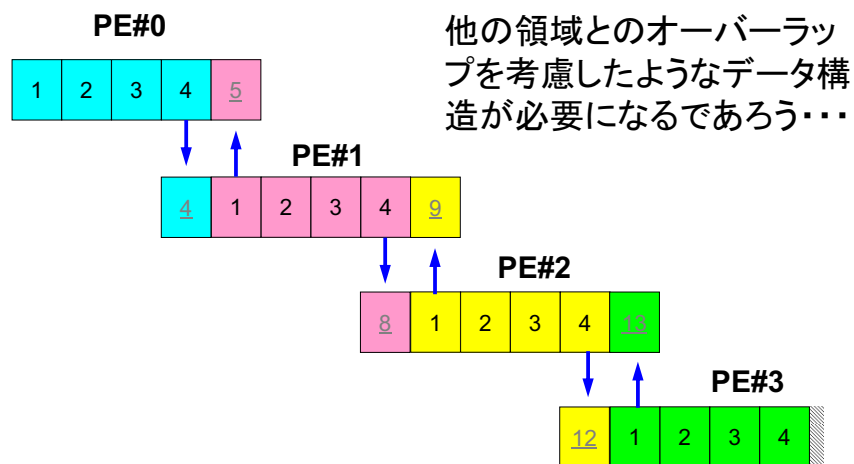
- 全体データと局所データ
- 差分法のオペレーションには隣接要素の値が必要
 - 領域(PE)の境界では隣接領域に属する要素からの情報が必要
 - 例えば, PE#1での計算を完結させるためには, PE#0の「4」番, PE#2の「1」番要素の情報が必要
- 隣接領域からの情報を考慮可能なデータ構造が必要
 - それほど単純では無い...



1対1通信が必要になる場面: 1D中央差分 差分法のオペレーションのためには隣接要素の情報が必要

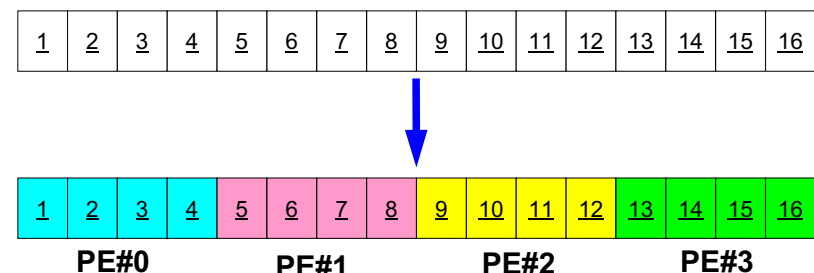


1対1通信が必要になる場面: 1D中央差分 差分法のオペレーションのためには隣接要素の情報が必要



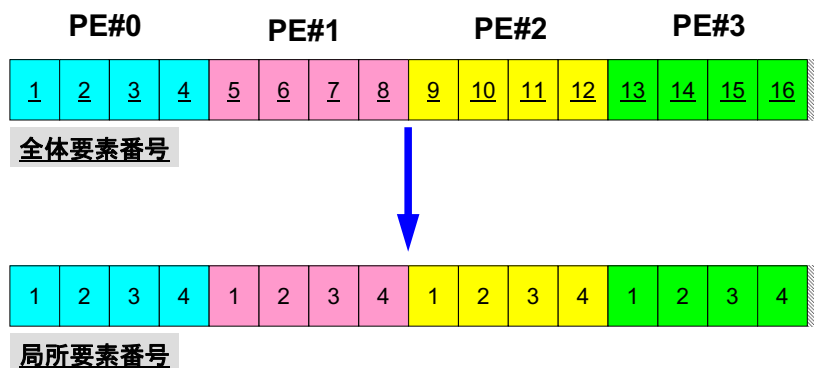
一次元差分法モデル局所データ構造 (1/5) NG=16の一次元領域

- NG=16の一次元領域を4領域(プロセッサ)に分割して並列に差分法(二次精度)の計算を実施するために必要なデータ構造を考える。



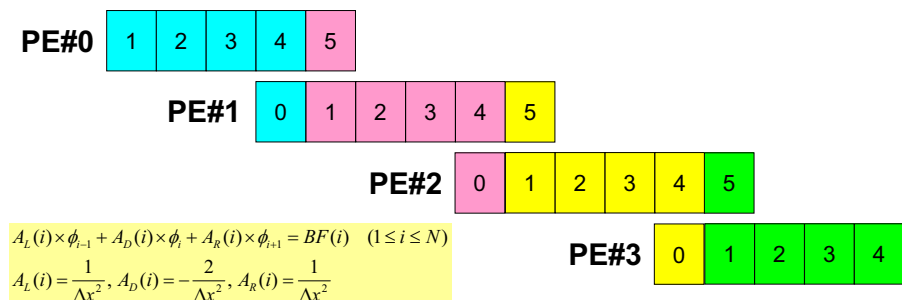
一次元差分法モデル局所データ構造 (2/5) 局所要素番号

- 各領域(プロセッサ)において, N=4の局所データを扱う



一次元差分法モデル局所データ構造 (3/5) 領域外の要素

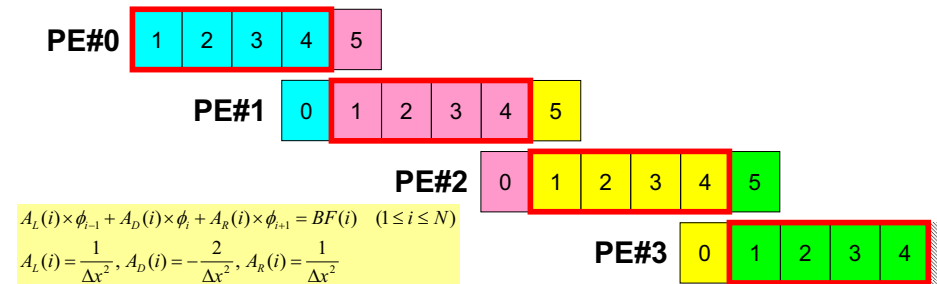
- 差分法(二次精度中央差分)の計算を並列に実施するためには, 本来領域外の要素の影響を考慮する必要がある。
- 領域間オーバーラップ
- 要素番号が0, N+1(=5)の要素を加えた, 局所データ構造



一次元差分法モデル局所データ構造(4/5)

必要な情報: 内点

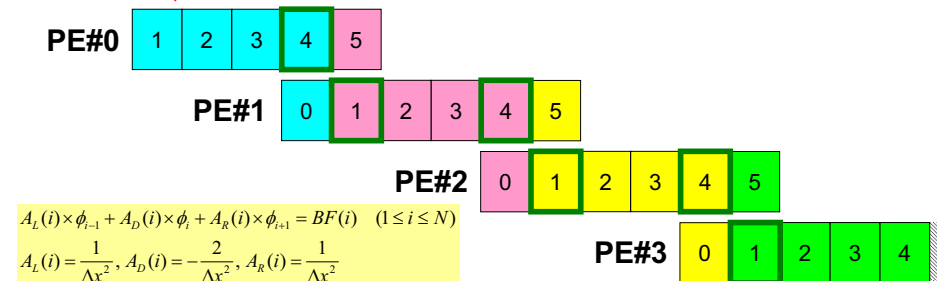
- 領域内の要素, オーバーラップ(本来領域外)要素の区別
 - 領域内要素 ($i=1\sim N$) : 内点 (Interior/Internal Points)



一次元差分法モデル局所データ構造(4/5)

必要な情報: 境界点

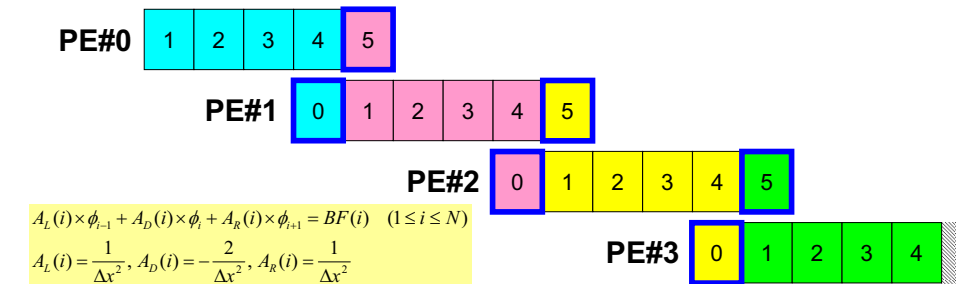
- 領域内要素, オーバーラップ(本来領域外)要素の区別
 - 領域内要素 ($i=1\sim N$) : 内点 (Interior/Internal Points)
 - オーバーラップ要素 ($i=0, N+1$) : 外点 (Exterior/External Points)
 - $i=0, i=N+1$ の要素が存在しない場合もある
 - 「内点」のうち他領域の「外点」となっている要素: 境界点 (Boundary Points)



一次元差分法モデル局所データ構造(4/5)

必要な情報: 外点

- 領域内要素, オーバーラップ(本来領域外)要素の区別
 - 領域内要素 ($i=1\sim N$) : 内点 (Interior/Internal Points)
 - オーバーラップ要素 ($i=0, N+1$) : 外点 (Exterior/External Points)
 - $i=0, i=N+1$ の要素が存在しない場合もある



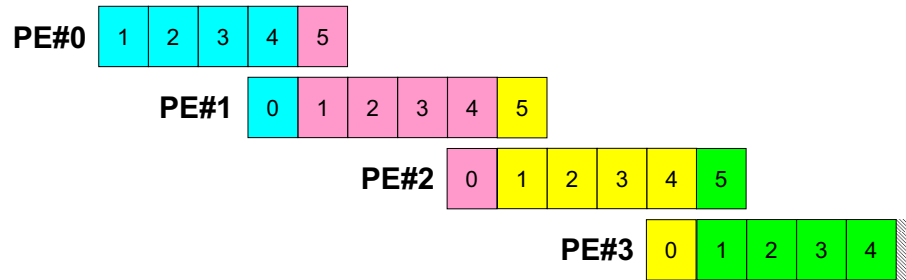
一次元差分法モデル局所データ構造(5/5)

必要な情報(続き)

- 「領域間オーバーラップ」を考慮した並列計算を実現するためには?
 - 各領域(プロセッサ)における「境界点」の情報を, 各隣接領域に「外点」の情報として配信する必要がある。
 - そのための局所データ構造が必要
- 隣接している領域数, 隣接している領域番号
 - 1対1通信の場合重要
 - オーバーラップ要素を共有している領域(プロセッサ)
- 隣接領域との「通信テーブル」
 - 境界点: 隣接領域への「送信」(send)
 - 外点: 隣接領域からの「受信」(recv)

これには問題がある

- C言語:「-1」から始まる配列は不可
- 二次元



例えばこのような二次元系の場合

33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

例えばこのような二次元系の場合

33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

「外点」の局所番号はどうする?

9	10	11	12	?
5	6	7	8	?
1	2	3	4	?

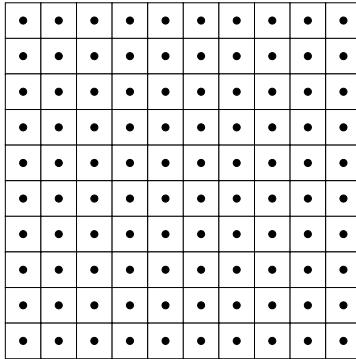
?	9	10	11	12	?
?	5	6	7	8	?
?	1	2	3	4	?

?	9	10	11	12	?
?	5	6	7	8	?
?	1	2	3	4	?

?	9	10	11	12
?	5	6	7	8
?	1	2	3	4

二次元差分法(1/5)

全体メッシュ

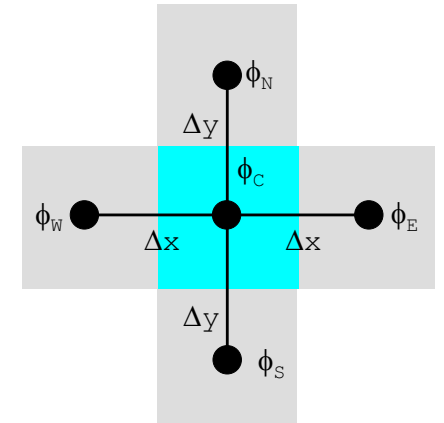


Jul2009

二次元中央差分法(5点差分法)の定式化

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = f$$

$$\left(\frac{\phi_E - 2\phi_C + \phi_W}{\Delta x^2} \right) + \left(\frac{\phi_N - 2\phi_C + \phi_S}{\Delta y^2} \right) = f_C$$

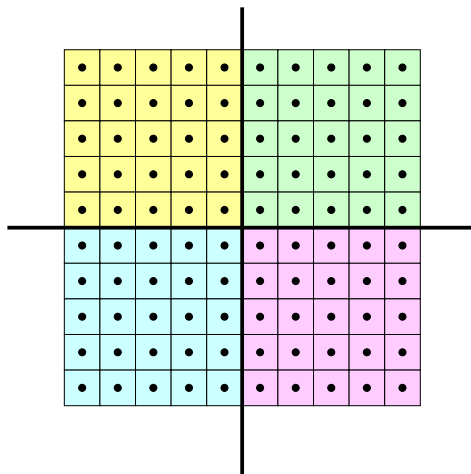


Jul2009

39

二次元差分法(2/5)

4領域に分割

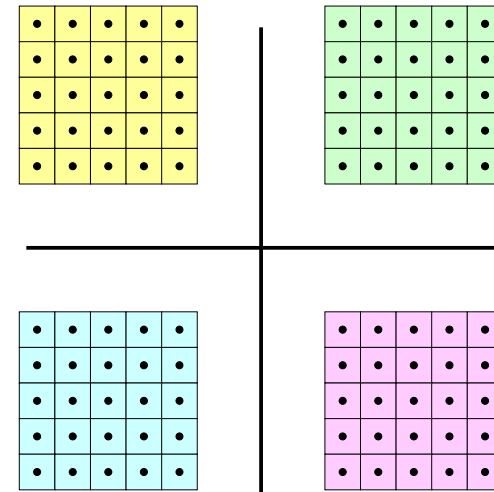


Jul2009

40

二次元差分法(3/5)

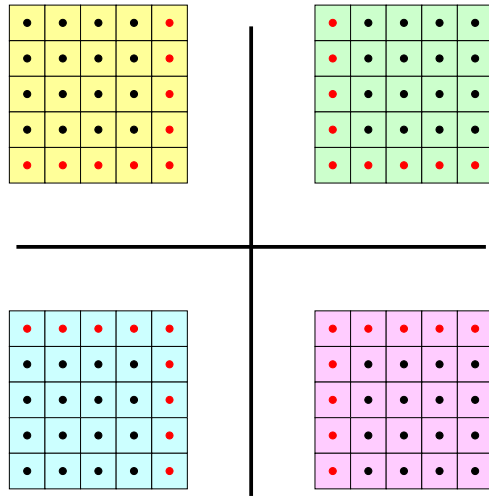
4領域に分割



Jul2009

二次元差分法(4/5)

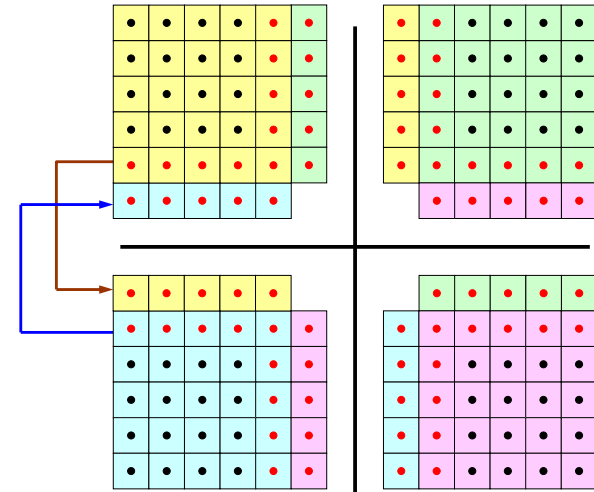
計算時には隣接領域の値が必要: オーバーラップ領域



Jul2009

二次元差分法(5/5)

計算時には隣接領域の値が必要: オーバーラップ領域

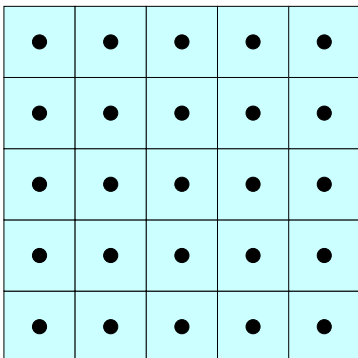


Jul2009

43

二次元差分法 各領域に必要な情報(1/4)

内点 (Internal Points)
その領域にアサインされた要素



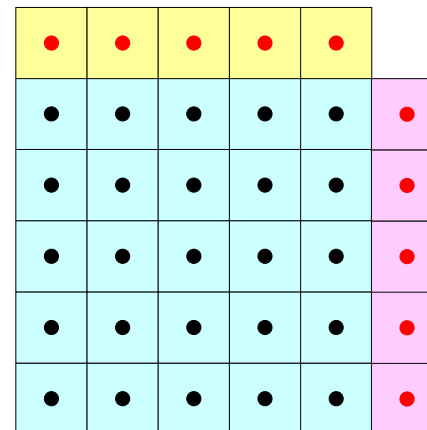
Jul2009

44

二次元差分法 各領域に必要な情報(2/4)

内点 (Internal Points)
その領域にアサインされた要素

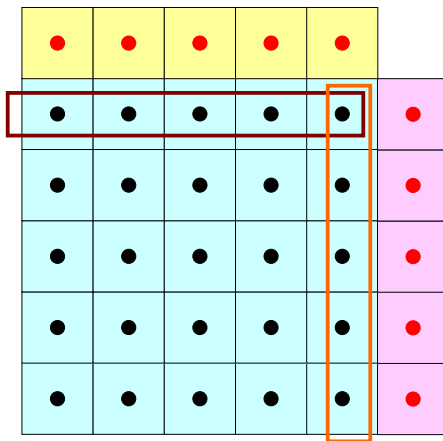
外点 (External Points)
他の領域にアサインされた要素であるがその領域の計算を実施するのに必要な要素
(オーバーラップ領域の要素)



Jul2009

二次元差分法

各領域に必要な情報(4/4)



内点 (Internal Points)
その領域にアサインされた要素

外点 (External Points)

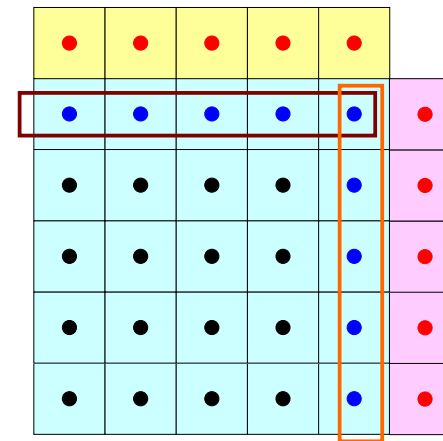
他の領域にアサインされた要素であるがその領域の計算を実施するのに必要な要素
(オーバーラップ領域の要素)

境界点(Boundary Points)

内点のうち、他の領域の外点となっている要素
他の領域の計算に使用される要素

二次元差分法

各領域に必要な情報(4/4)



内点 (Internal Points)
その領域にアサインされた要素

外点 (External Points)

他の領域にアサインされた要素であるがその領域の計算を実施するのに必要な要素
(オーバーラップ領域の要素)

境界点 (Boundary Points)

内点のうち、他の領域の外点となっている要素
他の領域の計算に使用される要素

領域間相互の関係

各領域データ(局所データ)仕様

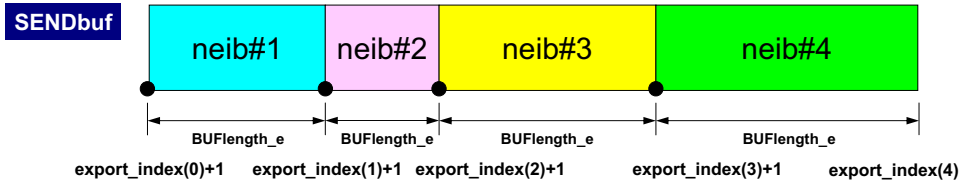
21	22	23	24	
13	14	15	16	20
9	10	11	12	19
5	6	7	8	18
1	2	3	4	17

- 内点, 外点
 - 内点～外点となるように局所番号をつける
- 隣接領域情報
 - オーバーラップ要素を共有する領域
 - 隣接領域数, 番号
- 外点情報
 - どの領域から, 何個の, どの外点の情報を「受信:import」するか
- 境界点情報
 - 何個の, どの境界点の情報を, どの領域に「送信:export」するか

一般化された通信テーブル:送信

- 送信相手
 - NEIBPETOT, NEIBPE(neib)
- それぞれの送信相手に送るメッセージサイズ
 - export_index(neib), neib= 1, NEIBPETOT
- 「境界点」番号
 - export_item(k), k= 1, export_index(NEIBPETOT)
- それぞれの送信相手に送るメッセージ
 - SENDbuf(k), k= 1, export_index(NEIBPETOT)

送信 (MPI_Isend/Irecv/Waitall)



```
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= VAL(kk)
  enddo
enddo

do neib= 1, NEIBPETOT
  iS_e= export_index(neib-1) + 1
  iE_e= export_index(neib )
  BUFLength_e= iE_e + 1 - iS_e

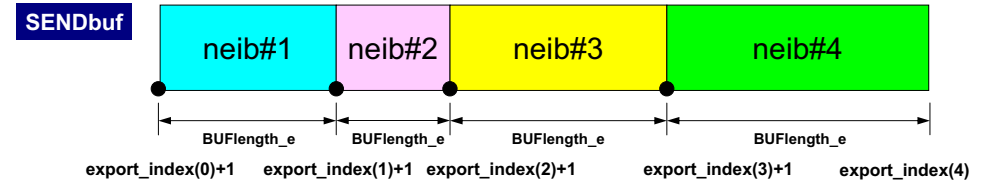
  call MPI_ISEND
  & (SENDbuf(iS_e), BUFLength_e, MPI_INTEGER, NEIBPE(neib), 0, &
  & MPI_COMM_WORLD, request_send(neib), ierr)
enddo

call MPI_WAITALL (NEIBPETOT, request_send, stat_rcv, ierr)
```

送信バッファへの代入
温度などの変数を直接送信、受信に使うのではなく、このようなバッファへ一回代入して計算することを勧める。

Jul2009

送信 (MPI_Sendrecv)



```
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= VAL(kk)
  enddo
enddo

do neib= 1, NEIBPETOT
  iS_e= export_index(neib-1) + 1
  iE_e= export_index(neib )
  BUFLength_e= iE_e + 1 - iS_e

  call MPI_SENDRECV
  & (SENDbuf(iS_e), BUFLength_e, MPI_INTEGER, NEIBPE(neib), 0, &
  & RECVbuf(iS_i), BUFLength_i, MPI_INTEGER, NEIBPE(neib), 0, &
  & MPI_COMM_WORLD, stat_sr, ierr)
enddo
```

送信バッファへの代入

Jul2009

一般化された通信テーブル: 受信

- 受信相手
 - NEIBPETOT, NEIBPE(neib)
- それぞれの受信相手から受け取るメッセージサイズ
 - import_index(neib)
- 「外点」番号
 - import_item(k), k= 1, import_index(NEIBPETOT)
- それぞれの受信相手から受け取るメッセージ
 - RECVbuf(k), k= 1, import_index(NEIBPETOT)

Jul2009

受信 (MPI_Isend/Irecv/Waitall)

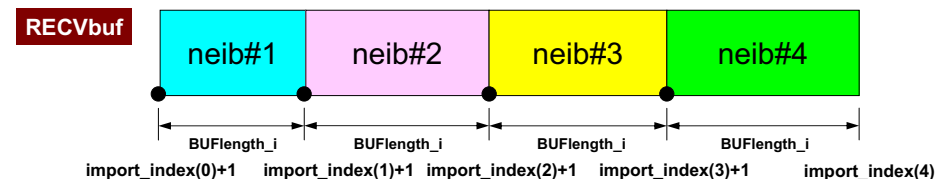
```
do neib= 1, NEIBPETOT
  iS_i= import_index(neib-1) + 1
  iE_i= import_index(neib )
  BUFLength_i= iE_i + 1 - iS_i

  call MPI_Irecv
  & (RECVbuf(iS_i), BUFLength_i, MPI_INTEGER, NEIBPE(neib), 0, &
  & MPI_COMM_WORLD, request_rcv(neib), ierr)
enddo

call MPI_WAITALL (NEIBPETOT, request_rcv, stat_rcv, ierr)

do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    VAL(kk)= RECVbuf(k)
  enddo
enddo
```

受信バッファから代入



Jul2009

受信 (MPI_Sendrecv)

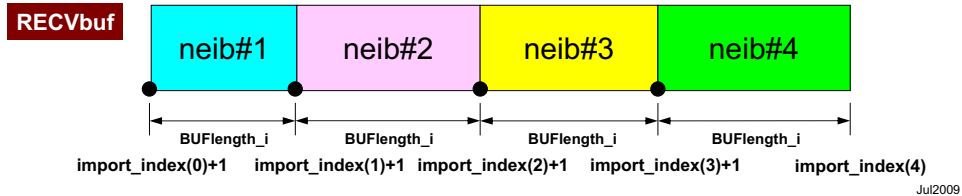
```

do neib= 1, NEIBPETOT
  is_i= import_index(neib-1) + 1
  ie_i= import_index(neib)
  BUFlength_i= ie_i + 1 - is_i

  call MPI_SENDRECV
&      (SENDbuf(is_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&      RECVbuf(is_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&      MPI_COMM_WORLD, stat_sr, ierr)
enddo

do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)  受信バッファからの代入
    kk= import_item(k)
    VAL(kk)= RECVbuf(k)
  enddo
enddo

```



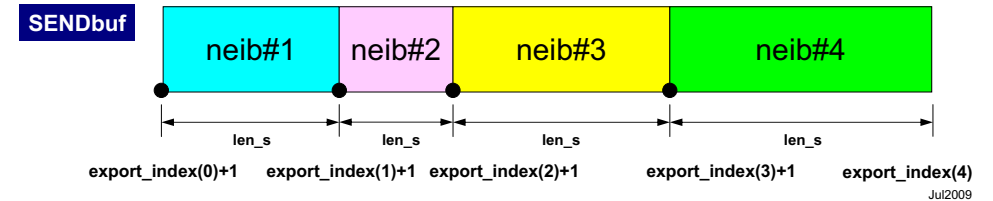
一般化された通信テーブル: 送信

```

!C
!C-- PREPARE send buffer
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= BUF(kk)
  enddo
enddo

!C
!C-- SEND
do neib= 1, NEIBPETOT
  is = export_index(neib-1) + 1
  len_s= export_index(neib) - export_index(neib-1)
  call MPI_ISEND (SENDbuf(is), len_s, MPI_INTEGER,
&      NEIBPE(neib), 0, MPI_COMM_WORLD,
&      request_send(neib), ierr)
enddo

```



一般化された通信テーブル: 受信

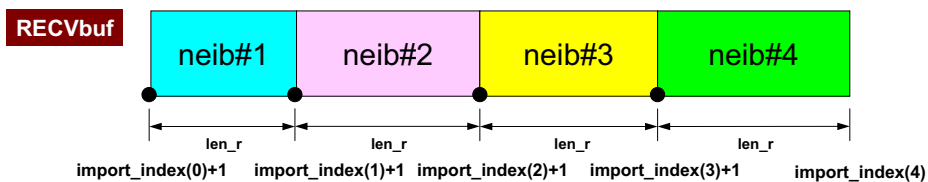
```

!C
!C-- RECV
do neib= 1, NEIBPETOT
  ir = import_index(neib-1) + 1
  len_r= import_index(neib) - import_index(neib-1)
  call MPI_IRECV (RECVbuf(ir), len_r, MPI_INTEGER,
&      NEIBPE(neib), 0, MPI_COMM_WORLD,
&      request_rcv(neib), ierr)
enddo

!C
!C-- WAITall for RECV
call MPI_WAITALL (NEIBPETOT, request_rcv, stat_rcv, ierr)

!C
!C-- update array
do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    BUF(kk)= RECVbuf(k)
  enddo
enddo

```



実際何をやっているのか

```

!C
!C +-----+
!C | SEND |
!C +-----+
!C==
do neib= 1, NEIBPETOT
  call MPI_ISEND (SENDbuf(neib), len_s, MPI_INTEGER,
&      NEIBPE(neib), 0, MPI_COMM_WORLD,
&      request_send(neib), ierr)
enddo
!C==

!C
!C +-----+
!C | RECV |
!C +-----+
!C==
do neib= 1, NEIBPETOT
  call MPI_IRECV (RECVbuf(neib), len_r, MPI_INTEGER,
&      NEIBPE(neib), 0, MPI_COMM_WORLD,
&      request_rcv(neib), ierr)
enddo
!C==

```

PE#0 PE#1 PE#2 PE#3

- 例えば、PE#1からはPE#0、PE#2に送信して、PE#0、PE#2から受信する。
- PE#0はPE#1から、PE#2はPE#1とPE#3から受信する。
- IRECVの部分では、相手のPE (NEIBPE(neib)) がISENDのループで自分に向けて送ったメッセージを受信する。
- 送信側のlen_s、受信側のlen_rは一致していなければならない。
 - NEIBPE(neib)がマッチしたときに通信が発生する。

送信と受信の関係

```
do neib= 1, NEIBPETOT
  iS_e= export_index(neib-1) + 1
  iE_e= export_index(neib )
  BUFlength_e= iE_e + 1 - iS_e

  call MPI_ISEND
&      (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0, &
&      MPI_COMM_WORLD, request_send(neib), ierr)
enddo
```

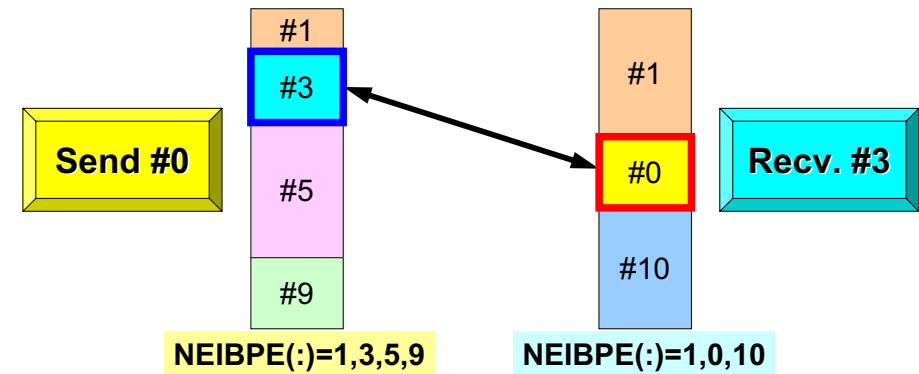
```
do neib= 1, NEIBPETOT
  iS_i= import_index(neib-1) + 1
  iE_i= import_index(neib )
  BUFlength_i= iE_i + 1 - iS_i

  call MPI_IRECV
&      (RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0, &
&      MPI_COMM_WORLD, request_recv(neib), ierr)
enddo
```

- 送信元・受信先プロセス番号, メッセージサイズ, 内容の整合性 !
- NEIBPE(neib) がマッチしたときに通信が起こる。

Jul2009

送信と受信の関係 (#0⇒#3)



- 送信元・受信先プロセス番号, メッセージサイズ, 内容の整合性 !
- NEIBPE(neib) がマッチしたときに通信が起こる。

Jul2009

なぜ SENDbuf, RECVbuf を用意するか?

- BUFを直接使っても良いのであるが...

```
!C
!C +-----+
!C | SEND |
!C +-----+
!C===
      do neib= 1, NEIBPETOT
        call MPI_ISEND (SENDbuf(neib), len_s, MPI_INTEGER,
&                      NEIBPE(neib), 0, MPI_COMM_WORLD,
&                      request_send(neib), ierr)
&
&      enddo
!C===

!C
!C +-----+
!C | RECV |
!C +-----+
!C===
      do neib= 1, NEIBPETOT
        call MPI_IRECV (RECVbuf(neib), len_r, MPI_INTEGER,
&                      NEIBPE(neib), 0, MPI_COMM_WORLD,
&                      request_recv(neib), ierr)
&
&      enddo
!C===
```

Jul2009

なぜ SENDbuf, RECVbuf を用意するか?

- BUFを直接使っても良いのであるが...
 - こんな感じになって使いにくい

```
!C
!C +-----+
!C | SEND |
!C +-----+
!C===
      do neib= 1, NEIBPETOT
        if (neib.eq.1) then
          ikk= 1
        else
          ikk= N
        endif

        if (my_rank.eq.0 .and. neib.eq.1) then
          ikk= N
        endif

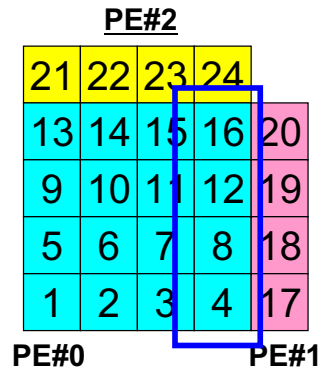
        call MPI_ISEND (BUF(ikk), len_s, MPI_INTEGER,
&                      NEIBPE(neib), 0, MPI_COMM_WORLD,
&                      request_send(neib), ierr)
&
&      enddo
!C===
```

Jul2009

送信バッファの効能

```
do neib= 1, NEIBPETOT
  is_e= export_index(neib-1) + 1
  ie_e= export_index(neib)
  BUlength_e= ie_e + 1 - is_e

  call MPI_ISEND
&      (VAL(...), BUlength_e, MPI_INTEGER, NEIBPE(neib), 0, &
&      MPI_COMM_WORLD, request_send(neib), ierr)
enddo
```



たとえば、この境界点は連続していないので、

- 送信バッファの先頭アドレス
- そこから数えて●●のサイズのメッセージ

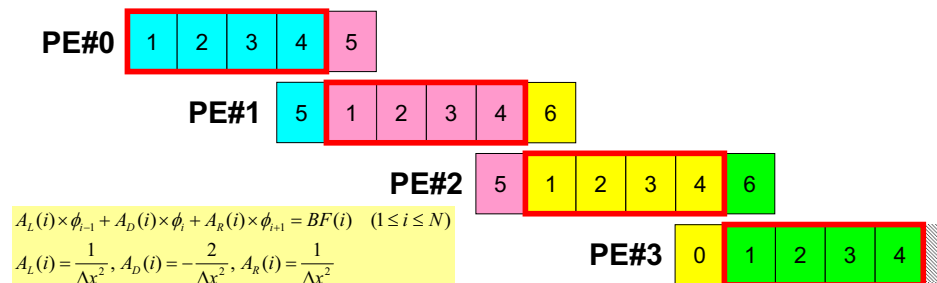
というような方法が困難

Jul2009

- 科学技術計算とMPI
- 1対1通信:一般化された通信テーブル
 - 一次元問題
 - 二次元問題
- 1対1通信の実装例
- 差分法による一次元熱伝導方程式ソルバーの並列化

一次元差分法モデル局所データ構造 新しい番号付け

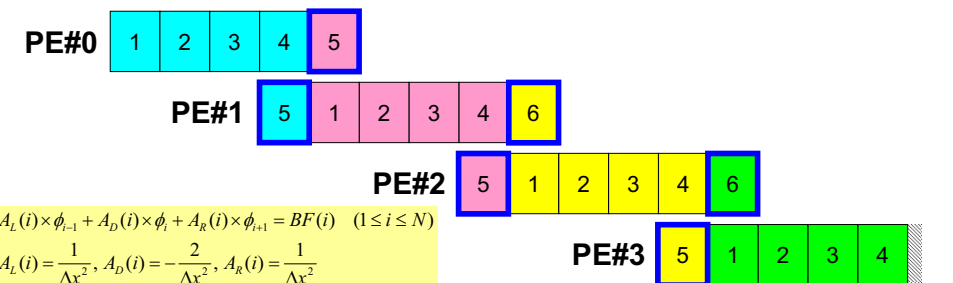
- 領域内の要素, オーバーラップ(本来領域外)要素の区別
 - 領域内要素 (i=1~N) : 内点 (Interior/Internal Points)



Jul2009

一次元差分法モデル局所データ構造 新しい番号付け

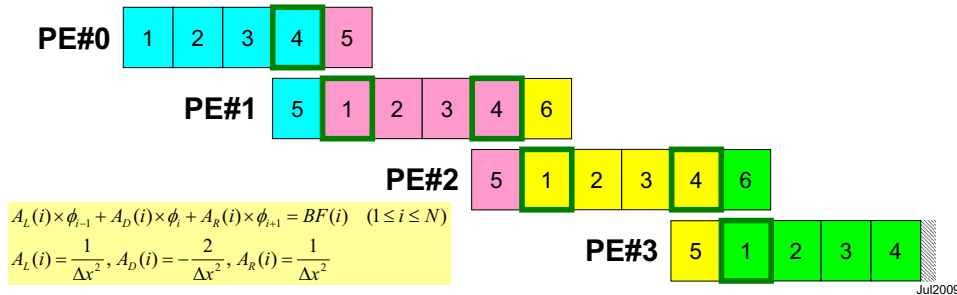
- 領域内要素, オーバーラップ(本来領域外)要素の区別
 - 領域内要素 (i=1~N) : 内点 (Interior/Internal Points)
 - オーバーラップ要素 (i=N+1, N+2) : 外点 (Exterior/External Points)
 - i=N+1, i=N+2の要素が存在しない場合もある



Jul2009

一次元差分法モデル局所データ構造 新しい番号付け

- 領域内要素, オーバーラップ(本来領域外)要素の区別
 - 領域内要素 ($i=1\sim N$) : 内点 (Interior/Internal Points)
 - オーバーラップ要素 ($i=N+1, N+2$) : 外点 (Exterior/External Points)
 - $i=N+1, i=N+2$ の要素が存在しない場合もある
 - 「内点」のうち他領域の「外点」となっている要素: 境界点 (Boundary Points)



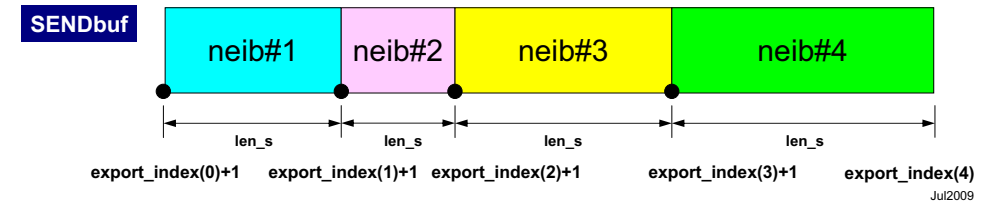
$$A_L(i) \times \phi_{i-1} + A_D(i) \times \phi_i + A_R(i) \times \phi_{i+1} = BF(i) \quad (1 \leq i \leq N)$$

$$A_L(i) = \frac{1}{\Delta x^2}, A_D(i) = -\frac{2}{\Delta x^2}, A_R(i) = \frac{1}{\Delta x^2}$$

一般化された通信テーブル: 送信

```
!C
!C-- PREPARE send buffer
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= BUF(kk)
  enddo
enddo

!C
!C-- SEND
do neib= 1, NEIBPETOT
  is = export_index(neib-1) + 1
  len_s= export_index(neib) - export_index(neib-1)
  call MPI_ISEND (SENDbuf(is), len_s, MPI_INTEGER,
    & NEIBPE(neib), 0, MPI_COMM_WORLD,
    & request_send(neib), ierr)
enddo
```

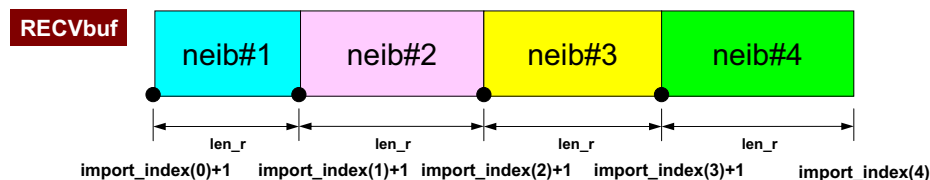


一般化された通信テーブル: 受信

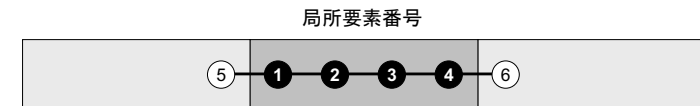
```
!C
!C-- RECV
do neib= 1, NEIBPETOT
  ir = import_index(neib-1) + 1
  len_r= import_index(neib) - import_index(neib-1)
  call MPI_Irecv (RECVbuf(ir), len_r, MPI_INTEGER,
    & NEIBPE(neib), 0, MPI_COMM_WORLD,
    & request_rcv(neib), ierr)
enddo

!C
!C-- WAITall for RECV
call MPI_WAITALL (NEIBPETOT, request_rcv, stat_rcv, ierr)

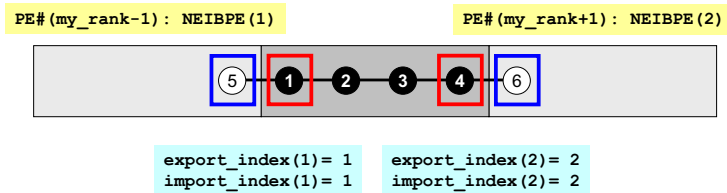
!C
!C-- update array
do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    BUF(kk)= RECVbuf(k)
  enddo
enddo
```



1D差分法の並列計算に必要な情報(1/4) 内点・外点・(境界点)



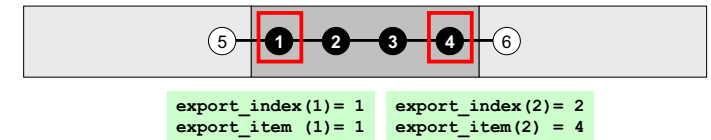
1D差分法の並列計算に必要な情報(2/4) 隣接プロセッサ情報



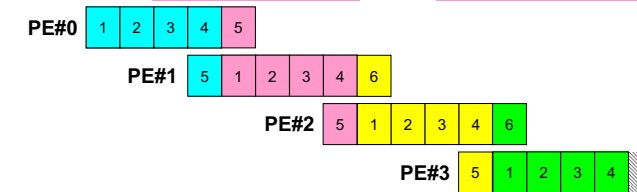
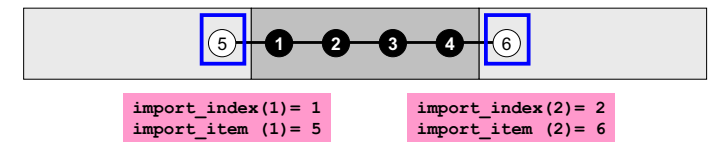
この領域から2つの隣接領域へ合計2つの要素(境界点□)を送信し、
2つの要素(外点□)を受信する

1D差分法の並列計算に必要な情報(3/4) 通信テーブル

隣接PEへの
送信(境界点)

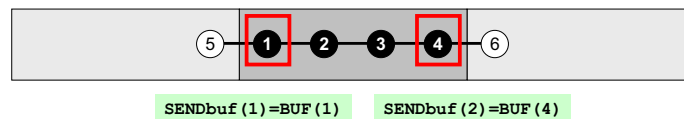


隣接PEからの
受信(外点)

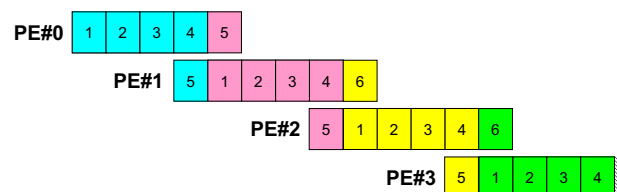
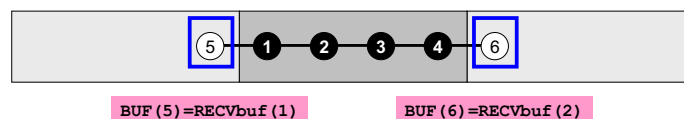


1D差分法の並列計算に必要な情報(4/4) 通信テーブル

隣接PEへの
送信(境界点)



隣接PEからの
受信(外点)



送信: 一次元問題

- 送信相手
 - NEIBPETOT, NEIBPE(neib)
 - NEIBPETOT=2, NEIB(1)= my_rank-1, NEIB(2)= my_rank+1
- それぞれの送信相手に送るメッセージサイズ
 - export_index(neib), neib= 1, NEIBPETOT
 - export_index(0)=0, export_index(1)= 1, export_index(2)= 2
- 「境界点」番号
 - export_item(k), k= 1, export_index(NEIBPETOT)
 - export_item(1)= 1, export_item(2)= N
- それぞれの送信相手に送るメッセージ
 - SENDbuf(k), k= 1, export_index(NEIBPETOT)
 - SENDbuf(1)= BUF(1), SENDbuf(2)= BUF(N)

受信:一次元問題

- 受信相手

- NEIBPETOT, NEIBPE(neib)

- NEIBPETOT=2, NEIB(1)= my_rank-1, NEIB(2)= my_rank+1

- それぞれの受信相手から受け取るメッセージサイズ

- import_index(neib), neib= 1, NEIBPETOT

- import_index(0)=0, import_index(1)= 1, import_index(2)= 2

- 「外点」番号

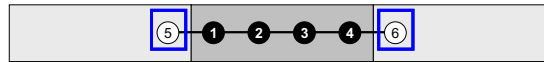
- import_item(k), k= 1, import_index(NEIBPETOT)

- import_item(1)= N+1, import_item(2)= N+2

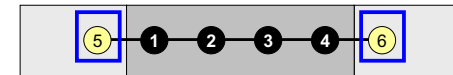
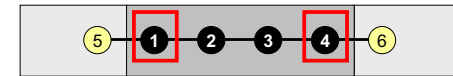
- それぞれの受信相手から受け取るメッセージ

- RECVbuf(k), k= 1, import_index(NEIBPETOT)

- BUF(N+1)=RECVbuf(1), BUF(N+2)=RECVbuf(2)



一般化された通信テーブル



```
NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item (1)= N+1
import_item (2)= N+2

export_index(1)= 1
export_index(2)= 2
export_item (1)= 1
export_item (2)= N

if (my_rank.eq.0) then
  import_item (1)= N+1
  export_item (1)= N
  NEIBPE(1)= my_rank+1
endif
```

Jul2009

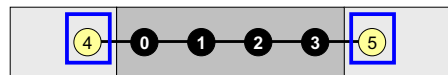
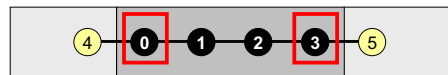
Jul2009

75

Jul2009

76

一般化された通信テーブル:C言語



BUF(0)=RECVbuf(1) BUF(5)=RECVbuf(2)

```
NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item (1)= N
import_item (2)= N+1

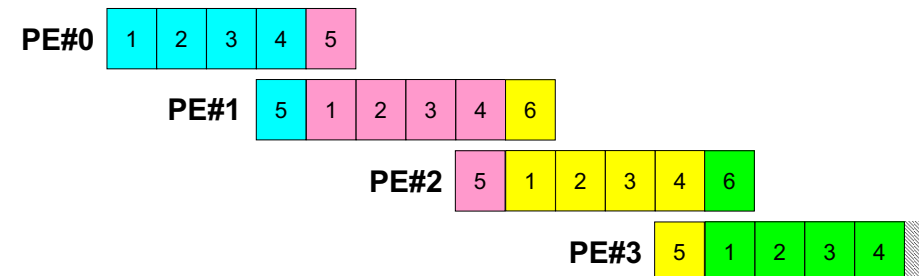
export_index(1)= 1
export_index(2)= 2
export_item (1)= 0
export_item (2)= N-1

if (my_rank.eq.0) then
  import_item (1)= N
  export_item (1)= N-1
  NEIBPE(1)= my_rank+1
endif
```

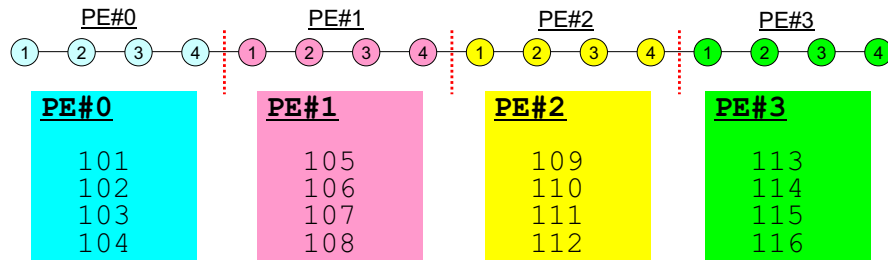
Jul2009

一次元モデル例題(1/10)

- オーバーラップのある一次元分散データ(NG=16, 4領域)について, 各プロセッサで「内点」に関する情報を入力し, 「境界点」の情報を, 各隣接領域に「外点」の情報として配信する。

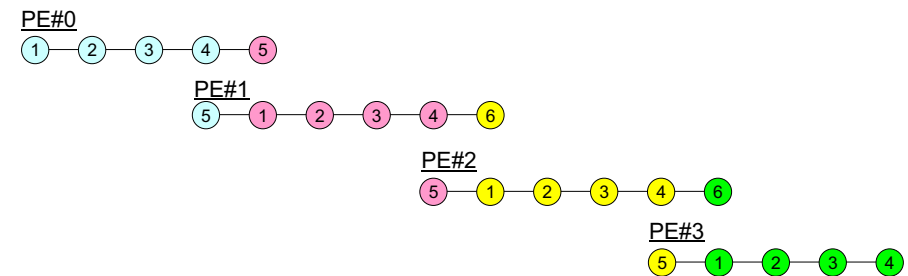


演算内容(1/3)



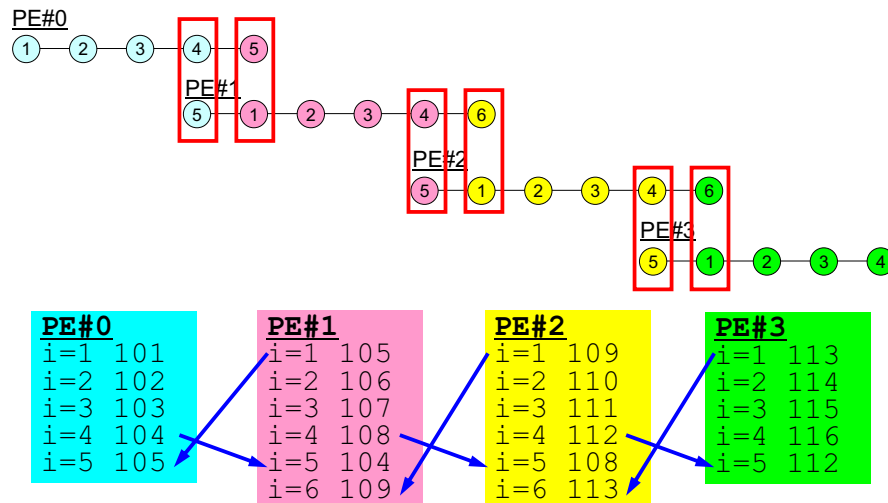
- 各PEの内点($i=1 \sim N(=4)$)において局所データを読み込み、「境界点」のデータを各隣接領域における「外点」として配信する。

演算内容(2/3):送信, 受信前



PE#0	PE#1	PE#2	PE#3
i=1 101	i=1 105	i=1 109	i=1 113
i=2 102	i=2 106	i=2 110	i=2 114
i=3 103	i=3 107	i=3 111	i=3 115
i=4 104	i=4 108	i=4 112	i=4 116
i=5 ?	i=5 ?	i=5 ?	i=5 ?
	i=6 ?	i=6 ?	

演算内容(3/3):送信, 受信後



サンプルプログラム

FORTRAN

```
$ cd <$1DH>/data
$ mpif90 -Oss -noprofile 1d-srb1.f
$ qsub go.sh
```

ISEND/IRECV

```
$ mpif90 -Oss -noprofile 1d-srb2.f
$ qsub go.sh
```

SENDRECV

C

```
$ cd <$1DH>/data
$ mpicc -Oss -noprofile 1d-srb1.c
$ qsub go.sh
```

ISEND/IRECV

```
$ mpicc -Oss -noprofile 1d-srb2.c
$ qsub go.sh
```

SENDRECV

一次元モデル例題: 1d-srb1.f (2/10)

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'

integer(kind=4) :: my_rank, PETOT
integer(kind=4) :: N, NEIBPETOT, BUFLength
integer(kind=4), dimension(2) :: NEIBPE

integer(kind=4), dimension(0:2) :: import_index, export_index
integer(kind=4), dimension( 2) :: import_item , export_item

integer(kind=4), dimension(2) :: SENDbuf, RECVbuf

integer(kind=4), dimension(:), allocatable :: BUF

integer(kind=4), dimension(:,:), allocatable :: stat_send
integer(kind=4), dimension(:,:), allocatable :: stat_rcv
integer(kind=4), dimension(: ), allocatable :: request_send
integer(kind=4), dimension(: ), allocatable :: request_rcv

!C
!C +-----+
!C | INIT. MPI |
!C +-----+
!C===
call MPI_INIT (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )
!C===
```

一次元モデル例題: 1d-srb1.f (3/10)

各PEにおいて局所データ(内点の値)読み込み

```
!C
!C +-----+
!C | INIT |
!C +-----+
!C===
N= 4
allocate (BUF(N+2))
BUF= 0

if (my_rank.eq.0) open (11, file='1d.0', status='unknown')
if (my_rank.eq.1) open (11, file='1d.1', status='unknown')
if (my_rank.eq.2) open (11, file='1d.2', status='unknown')
if (my_rank.eq.3) open (11, file='1d.3', status='unknown')

do i= 1, N
  read (11,*) BUF(i)
enddo

close (11)
```

PE#0

101
102
103
104

PE#1

105
106
107
108

PE#2

109
110
111
112

PE#3

113
114
115
116

一次元モデル例題: 1d-srb1.f (4/10)

各領域における外点

```
iSTART= 1
iEND = N+2
if (my_rank.eq.0) iEND = N+1
if (my_rank.eq.PETOT-1) iEND = N+1
do i= iSTART, iEND
  write (*,'(a, 3i8)') '%% before', my_rank, i, BUF(i)
enddo
```

PE#0

iS= 1
iE= N+1
外点数: 1

PE#1

iS= 1
iE= N+2
外点数: 2

PE#2

iS= 1
iE= N+2
外点数: 2

PE#3

iS= 1
iE= N+1
外点数: 1

```
%% before 0 1 101
%% before 0 2 102
%% before 0 3 103
%% before 0 4 104
%% before 0 5 0
```

```
%% before 1 1 105
%% before 1 2 106
%% before 1 3 107
%% before 1 4 108
%% before 1 5 0
%% before 1 6 0
```

```
%% before 2 1 109
%% before 2 2 110
%% before 2 3 111
%% before 2 4 112
%% before 2 5 0
%% before 2 6 0
```

```
%% before 3 1 113
%% before 3 2 114
%% before 3 3 115
%% before 3 4 116
%% before 3 5 0
```



一次元モデル例題: 1d-srb1.f (5/10)

隣接領域, 領域数

```
NEIBPETOT= 2
if (my_rank.eq.0) NEIBPETOT= 1
if (my_rank.eq.PETOT-1) NEIBPETOT= 1
if (PETOT.eq.1) NEIBPETOT= 0

NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

if (my_rank.eq.0) NEIBPE(1)= my_rank + 1
if (my_rank.eq.PETOT-1) NEIBPE(1)= my_rank - 1

import_index= 0
export_index= 0
import_item = 0
export_item = 0

import_index(1)= 1
import_index(2)= 2
import_item (1)= N+1
import_item (2)= N+2

export_index(1)= 1
export_index(2)= 2
export_item (1)= 1
export_item (2)= N

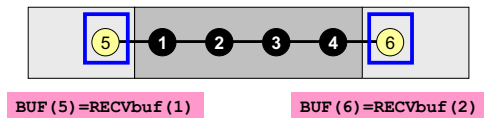
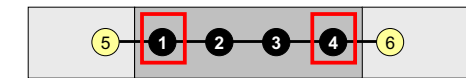
if (my_rank.eq.0) then
  import_item (1)= N+1
  export_item (1)= N
endif

BUFLength= 1
```

隣接PE数(NEIBPETOT),
隣接PE(NEIBPE)を決定

PE#0 **PE#1** **PE#2** **PE#3**

一般化された通信テーブル



```

NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item (1)= N+1
import_item (2)= N+2

export_index(1)= 1
export_index(2)= 2
export_item (1)= 1
export_item (2)= N

if (my_rank.eq.0) then
  import_item (1)= N+1
  export_item (1)= N
  NEIBPE(1)= my_rank+1
endif
  
```

Jul2009

Jul2009

一次元モデル例題: 1d-srb1.f (6/10)

通信識別子等の宣言, 記憶領域確保

```

!C
!C +-----+
!C | INIT. arrays for MPI_WAITALL |
!C +-----+
!C==
allocate (stat_send(MPI_STATUS_SIZE,NEIBPETOT))
allocate (stat_rcv(MPI_STATUS_SIZE,NEIBPETOT))
allocate (request_send(NEIBPETOT))
allocate (request_rcv(NEIBPETOT))
!C==
  
```

- 記憶領域を確保するだけで良い!!
- 通信識別子: request handle (通信リクエスト, とも言う)
 - 送信用: request_send, 受信用: request_rcv
- 状況オブジェクト配列: status object
 - 送信用: stat_send, 受信用: stat_rcv
- MPI_STATUS_SIZE
 - “mpif.h”, “mpi.h” で定義されている。
 - 勝手にこの数字を変更してはいけない。

Jul2009

一次元モデル例題: 1d-srb1.f (7/10)

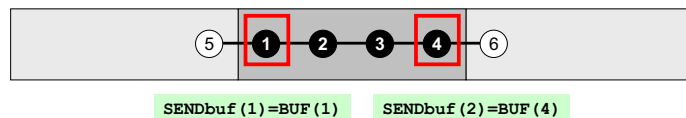
境界点情報送信準備

```

!C
!C +-----+
!C | PREPARE send buffer |
!C +-----+
!C==
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= BUF(kk)
  enddo
enddo
!C==
  
```

隣接プロセッサに BUF(1), BUF(N) を送る準備をする
境界点情報を SENDbufに代入

PE#(my_rank-1): NEIBPE(1) 局所要素番号 PE#(my_rank+1): NEIBPE(2)



87

Jul2009

一次元モデル例題: 1d-srb1.f (8/10)

境界点情報の送信

```

!C
!C +-----+
!C | SEND |
!C +-----+
!C==
do neib= 1, NEIBPETOT
  is = export_index(neib-1) + 1
  len_s = export_index(neib) - export_index(neib-1)
  call MPI_ISEND (SENDbuf(is), len_s, MPI_INTEGER,
    & NEIBPE(neib), 0, MPI_COMM_WORLD,
    & request_send(neib), ierr)
enddo
!C==

!C
!C +-----+
!C | RECV |
!C +-----+
!C==
do neib= 1, NEIBPETOT
  ir = import_index(neib-1) + 1
  len_r = import_index(neib) - import_index(neib-1)
  call MPI_Irecv (RECVbuf(ir), len_r, MPI_INTEGER,
    & NEIBPE(neib), 0, MPI_COMM_WORLD,
    & request_rcv(neib), ierr)
enddo
!C==

!C
!C +-----+
!C | WAITAll for RECV |
!C +-----+
!C==
call MPI_WAITALL (NEIBPETOT, request_rcv, stat_rcv, ierr)
!C==
  
```

SENDbuf(neib) から始まる, 長さlen_s(=1)の
メッセージを隣接PE(NEIBPE(neib))に送信する。

88

一次元モデル例題: 1d-srb1.f (8/10)

外点情報の送信

```
!C
!C +-----+
!C | SEND |
!C +-----+
!C===
do neib= 1, NEIBPETOT
  is = export_index(neib-1) + 1
  len_s = export_index(neib) - export_index(neib-1)
  call MPI_ISEND (SENDbuf(is), len_s, MPI_INTEGER,
    & NEIBPE(neib), 0, MPI_COMM_WORLD,
    & request_send(neib), ierr)
enddo
!C===

!C +-----+
!C | RECV |
!C +-----+
!C===
do neib= 1, NEIBPETOT
  ir = import_index(neib-1) + 1
  len_r = import_index(neib) - import_index(neib-1)
  call MPI_Irecv (RECVbuf(ir), len_r, MPI_INTEGER,
    & NEIBPE(neib), 0, MPI_COMM_WORLD,
    & request_rcv(neib), ierr)
enddo
!C===

!C +-----+
!C | WAITall for RECV |
!C +-----+
!C===
call MPI_WAITALL (NEIBPETOT, request_rcv, stat_rcv, ierr)
!C===
```

RECVbuf(neib) から始まる, 長さlen_r(=1)のメッセージを隣接PE(NEIBPE(neib))から受信する。

一次元モデル例題: 1d-srb1.f (8/10)

```
!C
!C +-----+
!C | SEND |
!C +-----+
!C===
do neib= 1, NEIBPETOT
  call MPI_ISEND (SENDbuf(neib), len_s, MPI_INTEGER,
    & NEIBPE(neib), 0, MPI_COMM_WORLD,
    & request_send(neib), ierr)
enddo
!C===

!C +-----+
!C | RECV |
!C +-----+
!C===
do neib= 1, NEIBPETOT
  call MPI_Irecv (RECVbuf(neib), len_r, MPI_INTEGER,
    & NEIBPE(neib), 0, MPI_COMM_WORLD,
    & request_rcv(neib), ierr)
enddo
!C===

!C +-----+
!C | WAITall for RECV |
!C +-----+
!C===
call MPI_WAITALL (NEIBPETOT, request_rcv, stat_rcv, ierr)
!C===
```

このメッセージのあと, RECVbufの中身を利用することが可能となる。

一次元モデル例題: 1d-srb1.f (8/10)

```
!C
!C +-----+
!C | SEND |
!C +-----+
!C===
do neib= 1, NEIBPETOT
  call MPI_ISEND (SENDbuf(neib), len_s, MPI_INTEGER,
    & NEIBPE(neib), 0, MPI_COMM_WORLD,
    & request_send(neib), ierr)
enddo
!C===

!C +-----+
!C | RECV |
!C +-----+
!C===
do neib= 1, NEIBPETOT
  call MPI_Irecv (RECVbuf(neib), len_r, MPI_INTEGER,
    & NEIBPE(neib), 0, MPI_COMM_WORLD,
    & request_rcv(neib), ierr)
enddo
!C===

!C +-----+
!C | WAITall for RECV |
!C +-----+
!C===
call MPI_WAITALL (NEIBPETOT, request_rcv, stat_rcv, ierr)
!C===
```

通信識別子(受信)

このメッセージのあと, RECVbufの中身を利用することが可能となる。

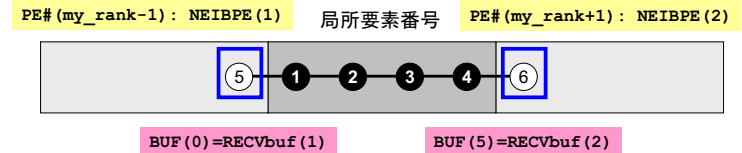
通信識別子 状況オブジェクト配列

一次元モデル例題: 1d-srb1.f (9/10)

外点情報代入

```
!C
!C +-----+
!C | update array |
!C +-----+
!C===
do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    BUF(kk)= RECVbuf(k)
  enddo
enddo
do i= iSTART, iEND
  write (*, '(a, 3i8)') '### after', my_rank, i, BUF(i)
enddo
!C===
```

隣接プロセッサから受け取った値をBUF(N+1), BUF(N+2)に代入する。



一次元モデル例題: 1d-srb1.f (10/10)

```
!C
!C +-----+
!C | update array |
!C +-----+
!C==
do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    BUF(kk)= RECVbuf(k)
  enddo
enddo

do i= iSTART, iEND
  write (*,'(a, 3i8)') '### after', my_rank, i, BUF(i)
enddo
!C==

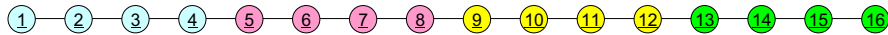
!C
!C +-----+
!C | WAITall for SEND |
!C +-----+
!C==
call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
!C==
```

%% before	0	1	101
%% before	0	2	102
%% before	0	3	103
%% before	0	4	104
%% before	0	5	0

%% before	1	1	105
%% before	1	2	106
%% before	1	3	107
%% before	1	4	108
%% before	1	5	104
%% before	1	6	109

%% before	2	1	109
%% before	2	2	110
%% before	2	3	111
%% before	2	4	112
%% before	2	5	108
%% before	2	6	113

%% before	3	1	113
%% before	3	2	114
%% before	3	3	115
%% before	3	4	116
%% before	3	5	112



一次元モデル例題: 1d-sr1.f (9/9)

```
!C
!C +-----+
!C | update array |
!C +-----+
!C==
do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    BUF(kk)= RECVbuf(k)
  enddo
enddo

do i= iSTART, iEND
  write (*,'(a, 3i8)') '### after', my_rank, i, BUF(i)
enddo
!C==

!C
!C +-----+
!C | WAITall for SEND |
!C +-----+
!C==
call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
!C==
```

このメッセージのあと、SENDbufの中身を変更することが可能となる。

一次元モデル例題: 1d-srb1.f (9/9)

```
!C
!C +-----+
!C | update array |
!C +-----+
!C==
do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    BUF(kk)= RECVbuf(k)
  enddo
enddo

do i= iSTART, iEND
  write (*,'(a, 3i8)') '### after', my_rank, i, BUF(i)
enddo!C==

!C
!C +-----+
!C | WAITall for SEND |
!C +-----+
!C==
call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
!C==
```

通信識別子 (送信用) 状況オブジェクト配列 (送信用)

MPI_ISEND/Irecv/WAITALLとMPI_SENDRECV

1d-srb1.f: Isend/Irecv/Waitall

```
allocate (stat_send(MPI_STATUS_SIZE,NEIBPETOT))
allocate (stat_rcv(MPI_STATUS_SIZE,NEIBPETOT))
allocate (request_send(NEIBPETOT))
allocate (request_rcv(NEIBPETOT))

...
do neib= 1, NEIBPETOT
  call MPI_ISEND (SENDbuf(neib), len_s, MPI_INTEGER,
    & NEIBPE(neib), 0, MPI_COMM_WORLD,
    & request_send(neib), ierr)
enddo

...
do neib= 1, NEIBPETOT
  call MPI_Irecv (RECVbuf(neib), len_r, MPI_INTEGER,
    & NEIBPE(neib), 0, MPI_COMM_WORLD,
    & request_rcv(neib), ierr)
enddo

...
call MPI_WAITALL (NEIBPETOT, request_rcv, stat_rcv, ierr)
call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)
```

1d-srb2.f: Sendrecv

```
allocate (stat_sr(MPI_STATUS_SIZE))

...
do neib= 1, NEIBPETOT
  call MPI_SENDRECV
    & (SENDbuf(neib), len_s, MPI_INTEGER, NEIBPE(neib), 0,
    & RECVbuf(neib), len_r, MPI_INTEGER, NEIBPE(neib), 0,
    & MPI_COMM_WORLD, stat_sr, ierr)
enddo
```

MPI_ISEND/Irecv/WAITALLの使い方

1d-srb1.f: Isend/Irecv/Waitall

```

allocate (stat_send(MPI_STATUS_SIZE,NEIBPETOT))
allocate (stat_rcv(MPI_STATUS_SIZE,NEIBPETOT))
allocate (request_send(NEIBPETOT))
allocate (request_rcv(NEIBPETOT))

...
do neib= 1, NEIBPETOT
  call MPI_ISEND (SENDbuf(neib), len_s, MPI_INTEGER,      &
&               NEIBPE(neib), 0, MPI_COMM_WORLD,          &
&               request_send(neib), ierr)
&
& enddo

...
do neib= 1, NEIBPETOT
  call MPI_Irecv (RECVbuf(neib), len_r, MPI_INTEGER,      &
&               NEIBPE(neib), 0, MPI_COMM_WORLD,          &
&               request_rcv(neib), ierr)
&
& enddo

...
call MPI_WAITALL (NEIBPETOT, request_rcv, stat_rcv, ierr)
call MPI_WAITALL (NEIBPETOT, request_send, stat_send, ierr)

```

- 通信識別子 request を定義: 送信用, 受信用
- 状況オブジェクト配列 status を定義: 送信用, 受信用
- MPI_Isend, MPI_Irecvにそれぞれ対応したMPI_Waitallを呼ぶ
 - request, statusで区別

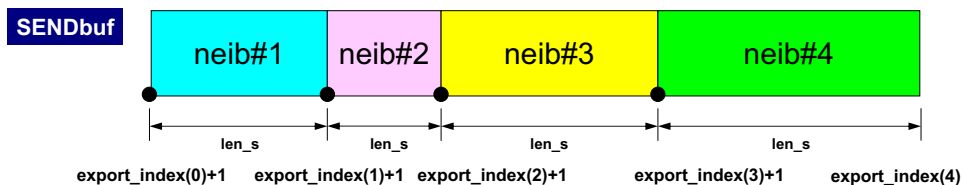
一般化された通信テーブル: 送信

```

!C
!C-- PREPARE send buffer
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= BUF(kk)
  enddo
enddo

!C
!C-- SEND
do neib= 1, NEIBPETOT
  is = export_index(neib-1) + 1
  len_s= export_index(neib) - export_index(neib-1)
  call MPI_ISEND (SENDbuf(is), len_s, MPI_INTEGER,      &
&               NEIBPE(neib), 0, MPI_COMM_WORLD,          &
&               request_send(neib), ierr)
&
& enddo

```



MPI_SENDRECVの使い方

1d-srb2.f: Sendrecv

```

allocate (stat_sr(MPI_STATUS_SIZE))

...
do neib= 1, NEIBPETOT
  call MPI_SENDRECV
& (SENDbuf(neib), BUFlength, len_s, NEIBPE(neib), 0,      &
&  RECVbuf(neib), BUFlength, len_r, NEIBPE(neib), 0,      &
&  MPI_COMM_WORLD, stat_sr, ierr)
&
& enddo

```

- 状況オブジェクト配列 status
 - status(MPI_STATUS_SIZE) を定義するだけで良い
 - Isend/Irecv/Waitallのときとは statusのサイズが異なるので注意すること。

一般化された通信テーブル: 受信

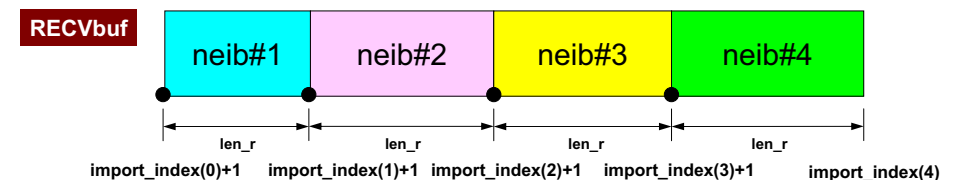
```

!C
!C-- RECV
do neib= 1, NEIBPETOT
  ir = import_index(neib-1) + 1
  len_r= import_index(neib) - import_index(neib-1)
  call MPI_Irecv (RECVbuf(ir), len_r, MPI_INTEGER,      &
&               NEIBPE(neib), 0, MPI_COMM_WORLD,          &
&               request_rcv(neib), ierr)
&
& enddo

!C
!C-- WAITall for RECV
call MPI_WAITALL (NEIBPETOT, request_rcv, stat_rcv, ierr)

!C
!C-- update array
do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    BUF(kk)= RECVbuf(k)
  enddo
enddo

```



- 科学技術計算とMPI
- 1対1通信:一般化された通信テーブル
 - 一次元問題
 - 二次元問題
- 1対1通信の実装例
- 差分法による一次元熱伝導方程式ソルバーの並列化

前処理付き共役勾配法

Preconditioned Conjugate Gradient Method (CG)

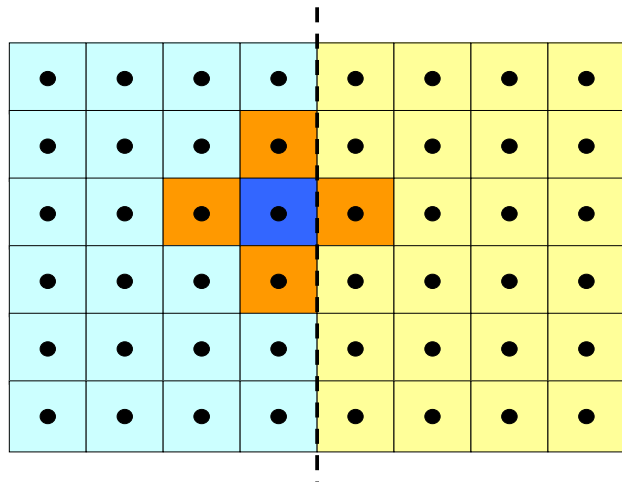
```

Compute  $\mathbf{r}^{(0)} = \mathbf{b} - [\mathbf{A}]\mathbf{x}^{(0)}$ 
for i = 1, 2, ...
  solve  $[\mathbf{M}]\mathbf{z}^{(i-1)} = \mathbf{r}^{(i-1)}$ 
   $\mathbf{p}_{i-1} = \mathbf{r}^{(i-1)} - \mathbf{z}^{(i-1)}$ 
  if i=1
     $\mathbf{p}^{(1)} = \mathbf{z}^{(0)}$ 
  else
     $\beta_{i-1} = \mathbf{p}_{i-1}^T \mathbf{p}_{i-2} / \mathbf{p}_{i-2}^T \mathbf{p}_{i-2}$ 
     $\mathbf{p}^{(i)} = \mathbf{z}^{(i-1)} + \beta_{i-1} \mathbf{p}^{(i-1)}$ 
  endif
   $\mathbf{q}^{(i)} = [\mathbf{A}]\mathbf{p}^{(i)}$ 
   $\alpha_i = \mathbf{p}^{(i)T} \mathbf{q}^{(i)} / \mathbf{q}^{(i)T} \mathbf{q}^{(i)}$ 
   $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha_i \mathbf{p}^{(i)}$ 
   $\mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \alpha_i \mathbf{q}^{(i)}$ 
  check convergence  $\|\mathbf{r}\|$ 
end
  
```

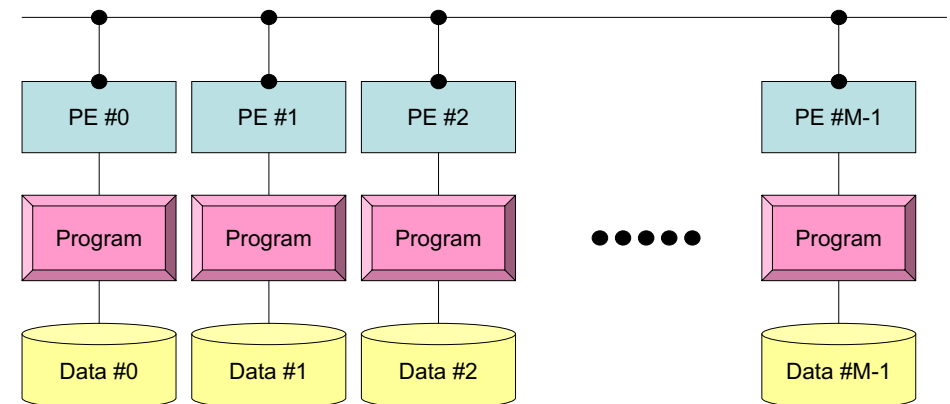
並列計算, 領域間通信が必要な部分

- 行列ベクトル積
- 内積

行列ベクトル積:他領域の値が必要
オーバーラップ+通信テーブル

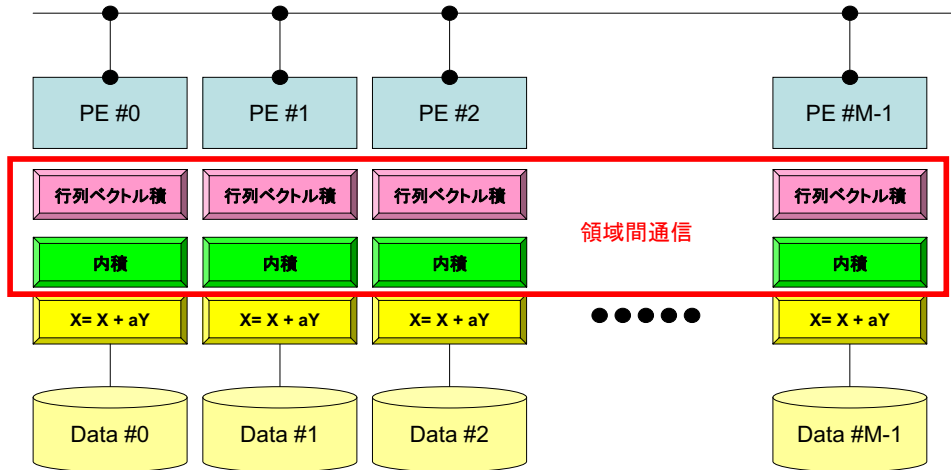


再びSPMD



CG法ではではこうなる

通信をやっている所だけ単体CPUと異なる



Jul2009

共役勾配法の「並列化」(1/2)

• 行列ベクトル積

- 領域境界データの交換を実施して、外点のベクトル値の最新値を得ておく。

```
!C
!C-- {q} = [A]{p}

exchange W(i,P)

do i= 1, N
  W(i,Q) = DIAG(i)*W(i,P)
  do j= INDEX(i-1)+1, INDEX(i)
    W(i,Q) = W(i,Q) + AMAT(j)*W(ITEM(j),P)
  enddo
enddo
```

Jul2009

共役勾配法の「並列化」(2/2)

• 内積

- MPI_ALLREDUCE

```
!C
!C +-----+
!C | RHO= {r}{z} |
!C +-----+
!C===
      RHO= 0.d0

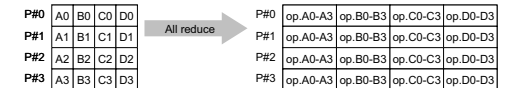
      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo

      allreduce RHO

!C===
```

Jul2009

MPI_ALLREDUCE



- MPI_REDUCE + MPI_BCAST
- 総和, 最大値を計算したら, 各プロセスで利用したい場合が多い

• call MPI_ALLREDUCE

(sendbuf, recvbuf, count, datatype, op, comm, ierr)

- sendbuf 任意 I 送信バッファの先頭アドレス,
- recvbuf 任意 O 受信バッファの先頭アドレス,
タイプは「datatype」により決定
- count 整数 I メッセージのサイズ
- datatype 整数 I メッセージのデータタイプ
- op 整数 I 計算の種類
- comm 整数 I コミュニケータを指定する
- ierr 整数 O 完了コード

Jul2009

MPI_Reduce/Allreduceの“op”

```
call MPI_REDUCE
(sendbuf, recvbuf, count, datatype, op, root, comm, ierr)
```

- MPI_MAX, MPI_MIN 最大値, 最小値
- MPI_SUM, MPI_PROD 総和, 積
- MPI_LAND 論理AND

```
real(kind=8):: X0, X1

call MPI_REDUCE
(X0, X1, 1, MPI_DOUBLE_PRECISION, MPI_MAX, 0, <comm>, ierr)
```

```
real(kind=8):: X0(4), XMAX(4)

call MPI_REDUCE
(X0, XMAX, 4, MPI_DOUBLE_PRECISION, MPI_MAX, 0, <comm>, ierr)
```

Jul2009

ファイル

```
$ cd <$1DH>/parallel
```

```
$ mpif90 -Oss -noprogram heat_cg_p.f
```

```
$ mpicc -Os -noprogram heat_cg_p.c
```

```
$ qsub go.sh
```

Jul2009

一次元熱伝導方程式: 並列版(1/7)

111

```
program CG_poi
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'

integer :: PETOT, my_rank, ierr
integer :: N, ITERmax
integer :: R, Z, P, Q, DD

real(kind=8) :: dx, RESID, dPHI, dPHImax, BF, EPS
real(kind=8), dimension(:), allocatable :: PHI, RHS
real(kind=8), dimension(:), allocatable :: DIAG, AMAT
real(kind=8), dimension(:, :), allocatable :: W

integer, dimension(:), allocatable :: INDEX, ITEM

integer(kind=4) :: NEIBPETOT, BUFLength
integer(kind=4), dimension(2) :: NEIBPE

integer(kind=4), dimension(0:2) :: import_index, export_index
integer(kind=4), dimension(2) :: import_item, export_item

real(kind=8), dimension(2) :: SENDbuf, RECVbuf

integer(kind=4), dimension(:), allocatable :: stat1

!C
!C +-----+
!C | INIT. |
!C +-----+
!C==

!C
!C-- MPI init.
call MPI_INIT (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr)
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr)
```

Jul2009

112

一次元熱伝導方程式: 並列版(2/7)

```
!C
!C-- CTRL data
if (my_rank.eq.0) then
  open (11, file='input.dat', status='unknown')
  read (11,*) Ng
  read (11,*) dx, BF
  read (11,*) ITERmax
  read (11,*) EPS
  close (11)
endif

call MPI_BCAST (Ng, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
call MPI_BCAST (ITERmax, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
call MPI_BCAST (dx, 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr) &
& call MPI_BCAST (BF, 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr) &
& call MPI_BCAST (EPS, 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr) &

!C
!C-- LOCAL MESH size
N= Ng / PETOT

if (N.le.1) goto 900

nr = Ng - nnn*PETOT
if (my_rank+1.le.nr) N= N + 1
```

Jul2009

一次元熱伝導方程式: 並列版(2/7)

```
!C
!C-- CTRL data
if (my_rank.eq.0) then
  open (11, file='input.dat', status='unknown')
  read (11,*) Ng
  read (11,*) dx, BF
  read (11,*) ITERmax
  read (11,*) EPS
  close (11)
endif

call MPI_BCAST (Ng, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
call MPI_BCAST (ITERmax, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
call MPI_BCAST (dx, 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr) &
call MPI_BCAST (BF, 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr) &
call MPI_BCAST (EPS, 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr) &

!C
!C-- LOCAL MESH size
N= Ng / PETOT

if (N.le.1) goto 900

nr = Ng - nnn*PETOT
if (my_rank+1.le.nr) N= N + 1
```

Jul2009

一次元熱伝導方程式: 並列版(3/7)

```
!C
!C-- MATRIX
allocate (PHI(N+2), DIAG(N), AMAT(2*N), RHS(N))
allocate (INDEX(0:N), ITEM(2*N), W(N+2,4))
PHI= 0.d0
AMAT= 1.d0/dx
DIAG= -2.d0/dx
RHS= -BF * dx

INDEX= 2

!C
!C-- CONNECTIVITY
INDEX(0)= 0

if (my_rank.eq.0) INDEX(1)= 1
if (my_rank.eq.PETOT-1) INDEX(nnn)= 1

do i= 1, nnn
  INDEX(i)= INDEX(i) + INDEX(i-1)
enddo
```

	1	2	3	4	5	6	7	8
1	$A_0(1)$	$A_R(1)$						
2	$A_L(2)$	$A_0(2)$	$A_R(2)$					
3		$A_L(3)$	$A_0(3)$	$A_R(3)$				
4			$A_L(4)$	$A_0(4)$	$A_R(4)$			
5				$A_L(5)$	$A_0(5)$	$A_R(5)$		
6					$A_L(6)$	$A_0(6)$	$A_R(6)$	
7						$A_L(7)$	$A_0(7)$	$A_R(7)$
8							$A_L(8)$	$A_0(8)$

$$\left(\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} \right) \times \Delta x + BF \times \Delta x = 0$$

$$\frac{\phi_{i+1} - \phi_i}{\Delta x} = \left(\frac{1}{\Delta x} \right) \phi_{i-1} - \left(\frac{2}{\Delta x} \right) \phi_i + \left(\frac{1}{\Delta x} \right) \phi_{i+1} = -BF \times \Delta x$$

A_L A_D A_R RHS

Jul2009

115

一次元熱伝導方程式: 並列版(3/7)

```
!C
!C-- MATRIX
allocate (PHI(0:N+1), DIAG(N), AMAT(2*N), RHS(N))
allocate (INDEX(0:N), ITEM(2*N), W(0:N+1,4))
PHI= 0.d0
AMAT= 1.d0/dx
DIAG= -2.d0/dx
RHS= -BF * dx

INDEX= 2

!C
!C-- CONNECTIVITY
INDEX(0)= 0

if (my_rank.eq.0) INDEX(1)= 1
if (my_rank.eq.PETOT-1) INDEX(N)= 1

do i= 1, N
  INDEX(i)= INDEX(i) + INDEX(i-1)
enddo
```

	1	2	3	4	5	6	7	8
1	AD1	AR1						
2	AL2	AD2	AR2					
3		AL3	AD3	AR3				
4			AL4	AD4	AR4			
5				AL5	AD5	AR5		
6					AL6	AD6	AR6	
7						AL7	AD7	AR7
8							AL8	AD8

非対角成分の数は「2」
ただし、両端では「1」

PE#0	PE#1	PE#2	PE#3
1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4

Jul2009

116

一次元熱伝導方程式: 並列版(4/7)

```
do i= 1, N
  js= INDEX(i-1)
  if (my_rank.eq.0 .and. i.eq.1) then
    ITEM(js+1)= i+1
    AMAT(js+1)= 0.d0
    DIAG(i)= 1.d0
    RHS(i)= 0.d0
  else if
    (my_rank.eq.PETOT-1 .and. i.eq.N) then
    ITEM(js+1)= i-1
    DIAG(i)= -1.d0/dx
  else
    ITEM(js+1)= i-1
    ITEM(js+2)= i+1
    if (i.eq.1) ITEM(js+1)= N+1
    if (i.eq.N) ITEM(js+2)= N+1
    if (my_rank.eq.0 .and. i.eq.N) ITEM(js+2)= N+1
    if (my_rank.eq.0 .and. i-1.eq.1) then
      AMAT(js+1)= 0.d0
    endif
  endif
enddo
```

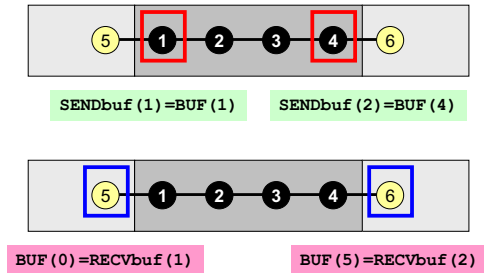
境界条件(Xmin)

境界条件(Xmax)

それ以外

Jul2009

一般化された通信テーブル



```

NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item(1)= N+1
import_item(2)= N+2

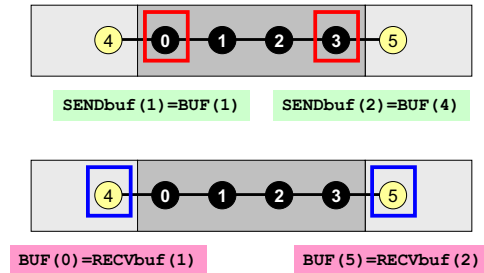
export_index(1)= 1
export_index(2)= 2
export_item(1)= 1
export_item(2)= N

if (my_rank.eq.0) then
  import_item(1)= N+1
  export_item(1)= N
  NEIBPE(1)= my_rank+1
endif
  
```

Jul2009

119

一般化された通信テーブル:C言語



```

NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item(1)= N
import_item(2)= N+1

export_index(1)= 1
export_index(2)= 2
export_item(1)= 0
export_item(2)= N-1

if (my_rank.eq.0) then
  import_item(1)= N
  export_item(1)= N-1
  NEIBPE(1)= my_rank+1
endif
  
```

Jul2009

120

一次元熱伝導方程式: 並列版(4/7)

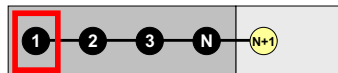
```

do i= 1, N
  js= INDEX(i-1)
  if (my_rank.eq.0 .and. i.eq.1) then
    ITEM(js+1)= i+1
    AMAT(js+1)= 0.d0
    DIAG(i) = 1.d0
    RHS(i) = 0.d0
  else if
    (my_rank.eq.PETOT-1 .and. i.eq.N) then
    ITEM(js+1)= i-1
    AMAT(js+1)= -1.d0/dx
    DIAG(i) = -1.d0/dx
  else
    ITEM(js+1)= i-1
    ITEM(js+2)= i+1

    if (i.eq.1) ITEM(js+1)= N+1
    if (i.eq.N) ITEM(js+2)= N+2
    if (my_rank.eq.0.and.i.eq.N) ITEM(js+2)= N+1

    if (my_rank.eq.0.and.i-1.eq.1) then
      AMAT(js+1)= 0.d0
    endif
  endif
endif
enddo
  
```

境界条件(Xmin)



Jul2009

一次元熱伝導方程式: 並列版(4/7)

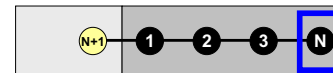
```

do i= 1, N
  js= INDEX(i-1)
  if (my_rank.eq.0 .and. i.eq.1) then
    ITEM(js+1)= i+1
    AMAT(js+1)= 0.d0
    DIAG(i) = 1.d0
    RHS(i) = 0.d0
  else if
    (my_rank.eq.PETOT-1 .and. i.eq.N) then
    ITEM(js+1)= i-1
    AMAT(js+1)= -1.d0/dx
    DIAG(i) = -1.d0/dx
  else
    ITEM(js+1)= i-1
    ITEM(js+2)= i+1

    if (i.eq.1) ITEM(js+1)= N+1
    if (i.eq.N) ITEM(js+2)= N+2
    if (my_rank.eq.0.and.i.eq.N) ITEM(js+2)= N+1

    if (my_rank.eq.0.and.i-1.eq.1) then
      AMAT(js+1)= 0.d0
    endif
  endif
endif
enddo
  
```

境界条件(Xmax)



Jul2009

一次元熱伝導方程式: 並列版(4/7)

```

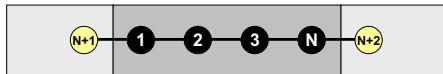
do i= 1, N
  js= INDEX(i-1)
  if (my_rank.eq.0 .and. i.eq.1) then
    ITEM(js+1)= i+1
    AMAT(js+1)= 0.d0
    DIAG(i )= 1.d0
    RHS(i )= 0.d0
  else if
& (my_rank.eq.PETOT-1 .and. i.eq.N) then
    ITEM(js+1)= i-1
    DIAG(i )= -1.d0/dX
  else
    ITEM(js+1)= i-1
    ITEM(js+2)= i+1

    if (i.eq.1) ITEM(js+1)= N+1
    if (i.eq.N) ITEM(js+2)= N+2
    if (my_rank.eq.0.and.i.eq.N) ITEM(js+2)= N+1

    if (my_rank.eq.0.and.i-1.eq.1) then
      AMAT(js+1)= 0.d0
    endif
  endif
endif
enddo

```

それ以外



Jul2009

123

一次元熱伝導方程式: 並列版(4/7)

```

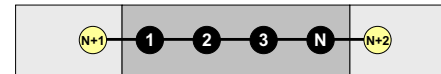
do i= 1, N
  js= INDEX(i-1)
  if (my_rank.eq.0 .and. i.eq.1) then
    ITEM(js+1)= i+1
    AMAT(js+1)= 0.d0
    DIAG(i )= 1.d0
    RHS(i )= 0.d0
  else if
& (my_rank.eq.PETOT-1 .and. i.eq.N) then
    ITEM(js+1)= i-1
    DIAG(i )= -1.d0/dX
  else
    ITEM(js+1)= i-1
    ITEM(js+2)= i+1

    if (i.eq.1) ITEM(js+1)= N+1
    if (i.eq.N) ITEM(js+2)= N+2
    if (my_rank.eq.0.and.i.eq.N) ITEM(js+2)= N+1

    if (my_rank.eq.0.and.i-1.eq.1) then
      AMAT(js+1)= 0.d0
    endif
  endif
endif
enddo

```

それ以外



Jul2009

124

一次元熱伝導方程式: 並列版(4/7)

```

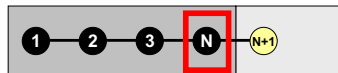
do i= 1, N
  js= INDEX(i-1)
  if (my_rank.eq.0 .and. i.eq.1) then
    ITEM(js+1)= i+1
    AMAT(js+1)= 0.d0
    DIAG(i )= 1.d0
    RHS(i )= 0.d0
  else if
& (my_rank.eq.PETOT-1 .and. i.eq.N) then
    ITEM(js+1)= i-1
    DIAG(i )= -1.d0/dX
  else
    ITEM(js+1)= i-1
    ITEM(js+2)= i+1

    if (i.eq.1) ITEM(js+1)= N+1
    if (i.eq.N) ITEM(js+2)= N+2
    if (my_rank.eq.0.and.i.eq.N) ITEM(js+2)= N+1

    if (my_rank.eq.0.and.i-1.eq.1) then
      AMAT(js+1)= 0.d0
    endif
  endif
endif
enddo

```

それ以外



Jul2009

一次元熱伝導方程式: 並列版(5/7)

```

!C
!C-- COMMUNICATION
NEIBPETOT= 2
if (my_rank.eq.0 ) NEIBPETOT= 1
if (my_rank.eq.PETOT-1) NEIBPETOT= 1
if (PETOT.eq.1) NEIBPETOT= 0

NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

if (my_rank.eq.0 ) NEIBPE(1)= my_rank + 1
if (my_rank.eq.PETOT-1) NEIBPE(1)= my_rank - 1

BUFlength= 1

import_index= 0
export_index= 0
import_item = 0
export_item = 0

import_index(1)= 1
import_index(2)= 2
import_item (1)= N+1
import_item (2)= N+2

export_index(1)= 1
export_index(2)= 2
export_item (1)= 1
export_item (2)= N

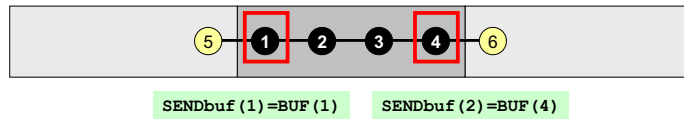
if (my_rank.eq.0) then
  import_item (1)= N+1
  export_item (1)= N
endif

```

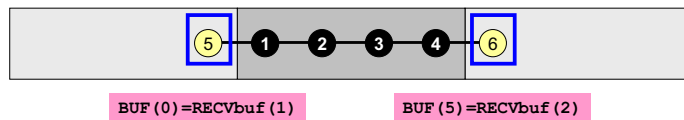
Jul2009

1D差分法の並列計算に必要な情報 新しい局所番号付け+通信テーブル

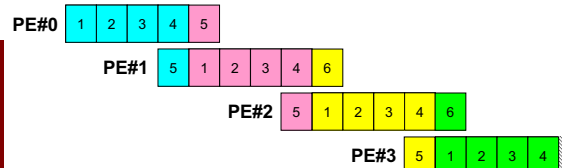
隣接PEへの
送信(境界点)



隣接PEからの
受信(外点)

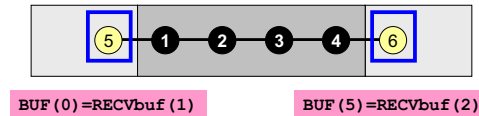
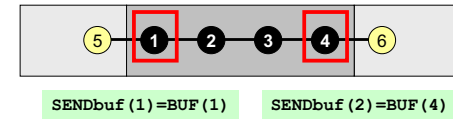


多次元への拡張を
考えるとこのような
局所番号付けが自然
(0~N+1)⇒(1~N+2)



Jul2009

一般化された通信テーブル



```
NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item(1)= N+1
import_item(2)= N+2

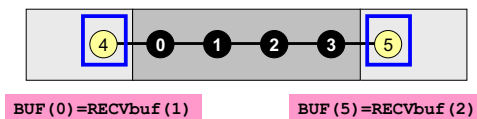
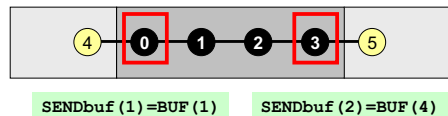
export_index(1)= 1
export_index(2)= 2
export_item(1)= 1
export_item(2)= N

if (my_rank.eq.0) then
  import_item(1)= N+1
  export_item(1)= N
  NEIBPE(1)= my_rank+1
endif
```

Jul2009

127

一般化された通信テーブル:C言語



```
NEIBPETOT= 2
NEIBPE(1)= my_rank - 1
NEIBPE(2)= my_rank + 1

import_index(1)= 1
import_index(2)= 2
import_item(1)= N
import_item(2)= N+1

export_index(1)= 1
export_index(2)= 2
export_item(1)= 0
export_item(2)= N-1

if (my_rank.eq.0) then
  import_item(1)= N
  export_item(1)= N-1
  NEIBPE(1)= my_rank+1
endif
```

Jul2009

128

一次元熱伝導方程式:並列版(6/7) 行列ベクトル積

```
!C
!C-- {q}= [A]{p}
!C-- init
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= W(kk,P)
  enddo
enddo

!C
!C-- SEND & RECV.
do neib= 1, NEIBPETOT
  is= export_index(neib-1) + 1
  ir= import_index(neib-1) + 1
  len_s= export_index(neib) - export_index(neib-1)
  len_r= import_index(neib) - import_index(neib-1)
  call MPI_SENDRECV
    (SENDbuf(is), len_s, MPI_DOUBLE_PRECISION, NEIBPE(neib), 0, &
    RECVbuf(ir), len_r, MPI_DOUBLE_PRECISION, NEIBPE(neib), 0, &
    MPI_COMM_WORLD, stat1, ierr)
enddo
update
do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    W(kk,P)= RECVbuf(k)
  enddo
enddo
do i= 1, N
  W(i,Q)= DIAG(i)*W(i,P)
  do j= INDEX(i-1)+1, INDEX(i)
    W(i,Q)= W(i,Q) + AMAT(j)*W(ITEM(j),P)
  enddo
enddo
```

送信バッファに
W(i,p) の値を入れる

Jul2009

送信: 一次元問題

送信相手

– NEIBPETOT, NEIBPE(neib)

• NEIBPETOT=2, NEIB(1)= my_rank-1, NEIB(2)= my_rank+1

それぞれの送信相手に送るメッセージサイズ

– export_index(neib), neib= 1, NEIBPETOT

• export_index(0)=0, export_index(1)= 1, export_index(2)= 2

「境界点」番号

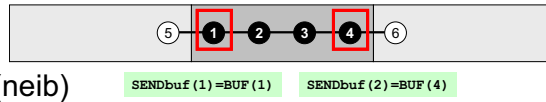
– export_item(k), k= 1, export_index(NEIBPETOT)

• export_item(1)= 1, export_item(2)= N

それぞれの送信相手に送るメッセージ

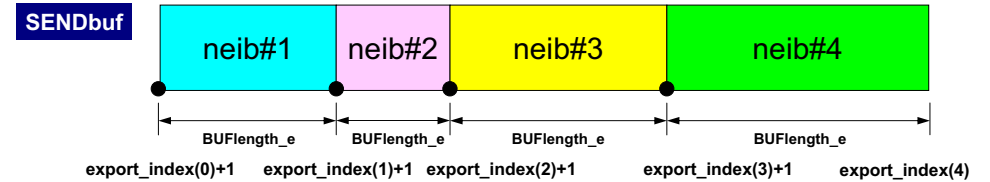
– SENDbuf(k), k= 1, export_index(NEIBPETOT)

• SENDbuf(1)= BUF(1), SENDbuf(2)= BUF(N)



Jul2009

送信 (MPI_Isend/Irecv/Waitall)



```
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= VAL(kk)
  enddo
enddo

do neib= 1, NEIBPETOT
  iS_e= export_index(neib-1) + 1
  iE_e= export_index(neib)
  BUFlength_e= iE_e + 1 - iS_e

  call MPI_ISEND
  & (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0, &
  & MPI_COMM_WORLD, request_send(neib), ierr)
enddo

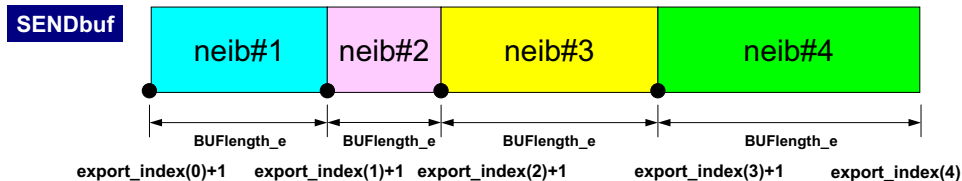
call MPI_WAITALL (NEIBPETOT, request_send, stat_rcv, ierr)
```

送信バッファへの代入
温度などの変数を直接送信, 受信に使うのではなく, このようなバッファへ一回代入して計算することを勧める。

Jul2009

131

送信 (MPI_Sendrecv)



```
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= VAL(kk)
  enddo
enddo

do neib= 1, NEIBPETOT
  iS_e= export_index(neib-1) + 1
  iE_e= export_index(neib)
  BUFlength_e= iE_e + 1 - iS_e

  call MPI_SENDRECV
  & (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0, &
  & RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0, &
  & MPI_COMM_WORLD, stat_sr, ierr)
enddo
```

送信バッファへの代入

Jul2009

132

受信: 一次元問題

受信相手

– NEIBPETOT, NEIBPE(neib)

• NEIBPETOT=2, NEIB(1)= my_rank-1, NEIB(2)= my_rank+1

それぞれの受信相手から受け取るメッセージサイズ

– import_index(neib), neib= 1, NEIBPETOT

• import_index(0)=0, import_index(1)= 1, import_index(2)= 2

「外点」番号

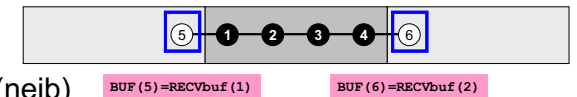
– import_item(k), k= 1, import_index(NEIBPETOT)

• import_item(1)= N+1, import_item(2)= N+2

それぞれの受信相手から受け取るメッセージ

– RECVbuf(k), k= 1, import_index(NEIBPETOT)

• BUF(N+1)=RECVbuf(1), BUF(N+2)=RECVbuf(2)



Jul2009

受信 (MPI_Isend/Irecv/Waitall)

```

do neib= 1, NEIBPETOT
  is_i= import_index(neib-1) + 1
  ie_i= import_index(neib)
  BUFlength_i= ie_i + 1 - is_i

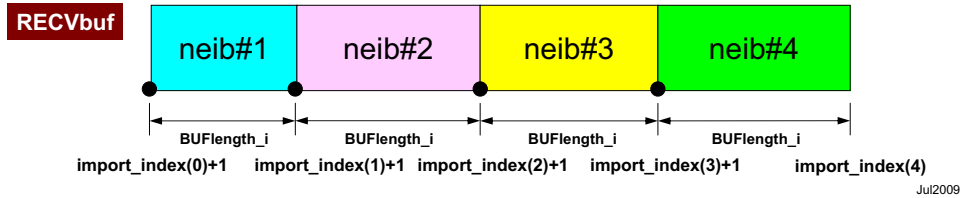
  call MPI_Irecv
  & (RECVbuf(is_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0, &
  & MPI_COMM_WORLD, request_recv(neib), ierr)
enddo

call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)

do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    VAL(kk)= RECVbuf(k)
  enddo
enddo

```

受信バッファから代入



一次元熱伝導方程式: 並列版 (6/7)

行列ベクトル積

```

!C
!C-- {q}= [A]{p}
!C- init
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= W(kk,P)
  enddo
enddo

!C
!C-- SEND & RECV.
do neib= 1, NEIBPETOT
  is= export_index(neib-1) + 1
  ie= import_index(neib-1) + 1
  len_s= export_index(neib) - export_index(neib-1)
  len_r= import_index(neib) - import_index(neib-1)
  call MPI_SENDRECV
  & (SENDbuf(is), len_s, MPI_DOUBLE_PRECISION, NEIBPE(neib), 0, &
  & RECVbuf(ie), len_r, MPI_DOUBLE_PRECISION, NEIBPE(neib), 0, &
  & MPI_COMM_WORLD, stat1, ierr)
enddo

!C- update
do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    W(kk,P)= RECVbuf(k)
  enddo
enddo

do i= 1, N
  W(i,Q) = DIAG(i)*W(i,P)
  do j= INDEX(i-1)+1, INDEX(i)
    W(i,Q) = W(i,Q) + AMAT(j)*W(ITEM(j),P)
  enddo
enddo

```

送受信

受信 (MPI_Sendrecv)

```

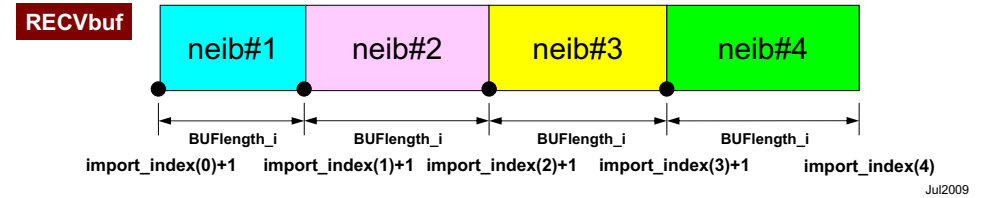
do neib= 1, NEIBPETOT
  is_i= import_index(neib-1) + 1
  ie_i= import_index(neib)
  BUFlength_i= ie_i + 1 - is_i

  call MPI_SENDRECV
  & (SENDbuf(is_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0, &
  & RECVbuf(is_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0, &
  & MPI_COMM_WORLD, stat_sr, ierr)
enddo

do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    VAL(kk)= RECVbuf(k)
  enddo
enddo

```

受信バッファからの代入



一次元熱伝導方程式: 並列版 (6/7)

行列ベクトル積

```

!C
!C-- {q}= [A]{p}
!C- init
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= W(kk,P)
  enddo
enddo

!C
!C-- SEND & RECV.
do neib= 1, NEIBPETOT
  is= export_index(neib-1) + 1
  ie= import_index(neib-1) + 1
  len_s= export_index(neib) - export_index(neib-1)
  len_r= import_index(neib) - import_index(neib-1)
  call MPI_SENDRECV
  & (SENDbuf(is), len_s, MPI_DOUBLE_PRECISION, NEIBPE(neib), 0, &
  & RECVbuf(ie), len_r, MPI_DOUBLE_PRECISION, NEIBPE(neib), 0, &
  & MPI_COMM_WORLD, stat1, ierr)
enddo

!C- update
do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    W(kk,P)= RECVbuf(k)
  enddo
enddo

do i= 1, N
  W(i,Q) = DIAG(i)*W(i,P)
  do j= INDEX(i-1)+1, INDEX(i)
    W(i,Q) = W(i,Q) + AMAT(j)*W(ITEM(j),P)
  enddo
enddo

```

隣接PEから受信した受信バッファの値を $W(i,p)$ に入れる

行列ベクトル積
 $\{q\} = [A] \{p\}$

一次元熱伝導方程式: 並列版(7/7)

内積

```
!C
!C-- ALPHA= RHO / {p}{q}
C10= 0.d0
do i= 1, N
  C10= C10 + W(i,P)*W(i,Q)
enddo
call MPI_allREDUCE (C10, C1, 1, MPI_DOUBLE_PRECISION,
& MPI_SUM, MPI_COMM_WORLD, ierr)
& ALPHA= RHO / C1

!C
!C-- {x}= {x} + ALPHA*{p}
!C {x}= {x} - ALPHA*{q}
do i= 1, N
  PHI(i) = PHI(i) + ALPHA * W(i,P)
  W (i,R)= W(i,R) - ALPHA * W(i,Q)
enddo

DNRM20 = 0.0
do i= 1, N
  DNRM20= DNRM20 + W(i,R)**2
enddo
call MPI_allREDUCE (DNRM20, DNRM2, 1, MPI_DOUBLE_PRECISION,
& MPI_SUM, MPI_COMM_WORLD, ierr)
& RESID= dsqrt(DNRM2/DNRM2)

if (my_rank.eq.0.and.mod(iter,1000).eq.0) then
  write (*, '(i5,1p16.6)') iter, RESID
endif

if ( RESID.le.EPS) goto 900
RHO1 = RHO
enddo
```

137

Jul2009

139

まとめ

- 並列化: 元のプログラムの特性を良く理解しておくことが重要
- アプリケーションに適した, アルゴリズム=データ構造
 - 本講習の例: 元のプログラムとの違いはわずか

Jul2009

go.sh, input.dat を変えて計算して見よ

PE数, Ng

```
#!/bin/r paraCG
#!/bin/q lecture5
#!/bin-N 1
#!/bin-J T16
#!/bin-e err
#!/bin-o test-016.lst
#!/bin-lm 28GB
#!/bin-lt 00:15:00
#!/bin$

cd $PBS_O_WORKDIR
mpirun nl.sh ./a.out

exit
```

```
80000
1.d0 1.d0
80000
1.d-7
```

138

Jul2009