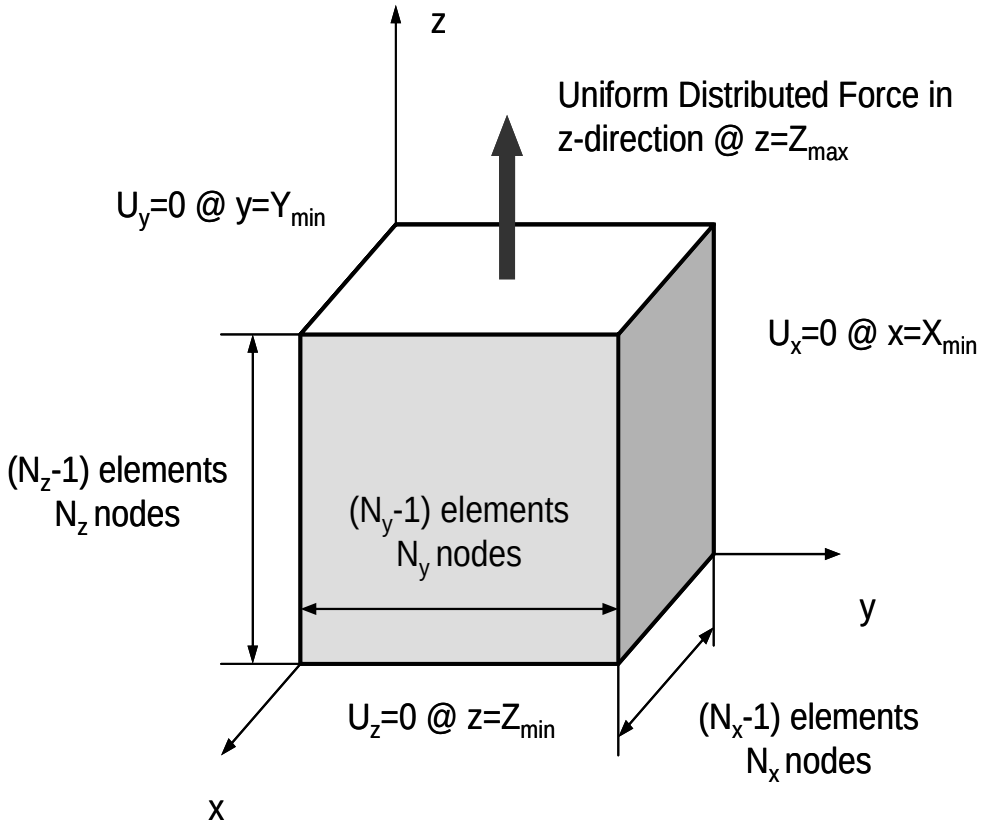
通信と計算のオーバーラップ
Communication-Computation Overlapping

中島研吾
東京大学情報基盤センター

対象とする問題



- 境界条件
  - 対称条件
    - U_X=0@X=0
    - U_Y=0@Y=0
    - U_Z=0@Z=0
  - 等分布荷重
    - F_Z=1@Z=Z_{max}

- 弾性体
  - ヤング率：E (=1.00)，ポアソン比：ν (=0.30)
- 直方体
  - 一辺長さ1の立方体（六面体）要素
  - 各方向にNX・NY・NZ個の節点

プログラムの概要

- 三次元弾性問題
  - 3×3ブロック処理
- 前処理無しCG法
- Hybrid

- 並列分散メッシュをプログラム内で自動生成
  - 予めメッシュ生成，領域分割等の必要ナシ
- CG法の行列ベクトル積部に「計算と通信のオーバーラップ」を導入
  - SEND_RECV：通信量少ない，実質的には呼び出し～立ち上がりのオーバーヘッド（latency）がほとんど

ファイルコピー ＋コンパイル

Fortranのみ

```
>$ cd ~/pFEM

>$ cp /home/ss/aics60/2014Summer/cc_overlap.tar .

>$ tar xvf cc_overlap.tar

>$ cd cc_overlap
>$ cd src0
>$ make

>$ cd ../src0m
>$ make

>$ cd ../run
>$ ls -l sol*
   sol0      計算-通信オーバーラップ無し
   sol0m     計算-通信オーバーラップ有り
```

実　行

Fortranのみ

```
>$ cd ~/pFEM/run

修正するファイル

mesh.inp     問題設定
go00.sh      sol0実行用スクリプト
go0m.sh      sol0m実行用スクリプト
```

“mesh.inp”の中身：Hybrid 16×1

```
(値)          （変数名）              （変数内容）

200 200 300 npx,npy,npz          p.2のNx，Ny，Nz
  2   2   3 ndx,ndy,ndz          X，Y，Z軸方向の分割数
 16   1     PEsmpTOT,(unused)    各MPIプロセスにおけるスレッド
                                 数(=1)，未使用（1を入れる）
200            ITERmax           CG法の反復回数
```

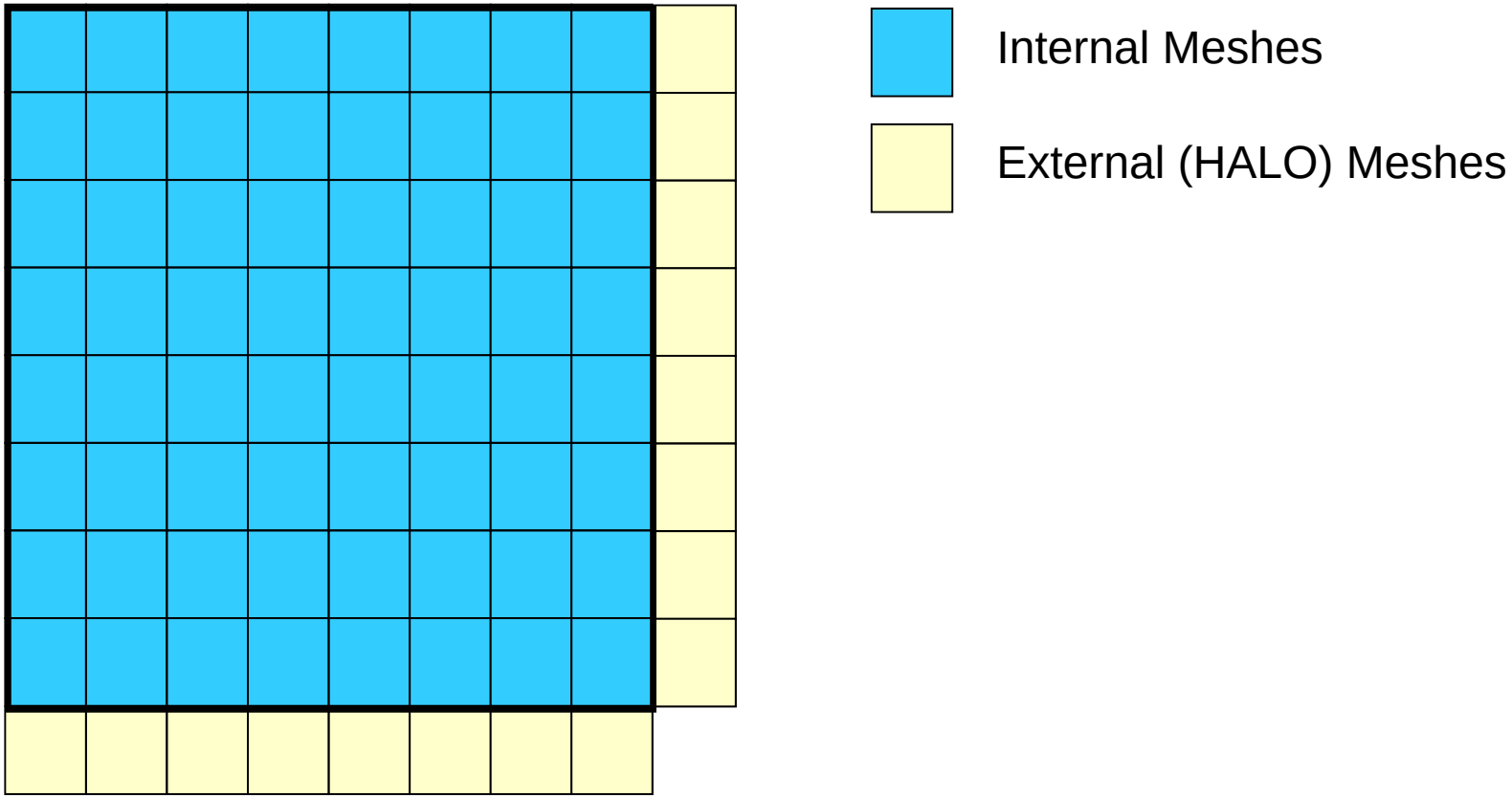
- npx,npy,npzはndx,ndy,ndzで割り切れる必要あり
- ndx×ndy×ndzが総MPIプロセス数
  - 上記の場合は12ノード，192コア，12プロセス，1プロセスあたりのスレッド数＝16（PEsmpTOT）
- とりあえず計算の性能だけを見るのであればITERmaxは上記のように少なめにする
  - 収束するまで計算すると時間がかかる

go00.shの例

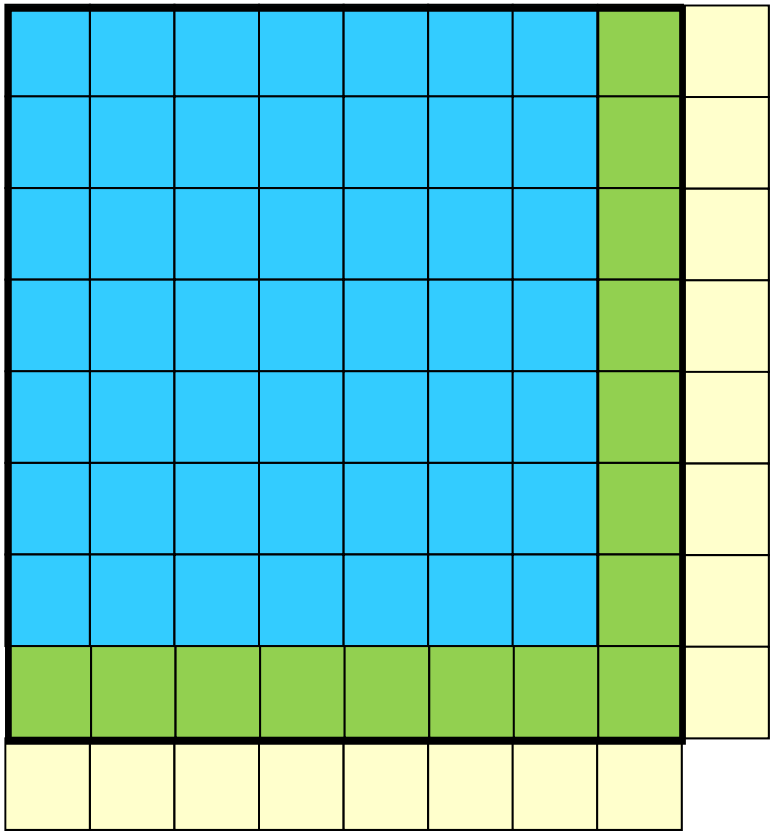
export OMP_NUM_THREADS=PEsmpTOT

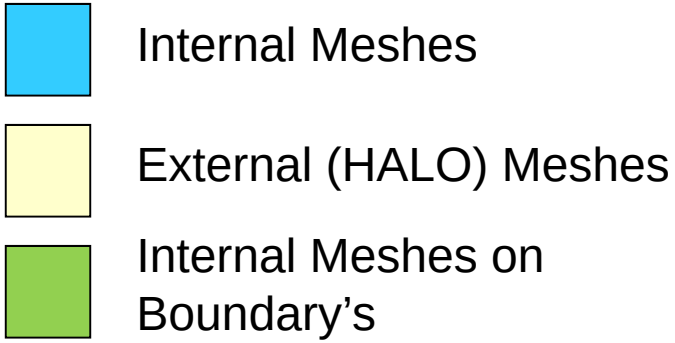
```
#!/bin/sh
#PJM -L "node=12"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=school"
#PJM -o "t00-012-128-128-128-03.lst"
#PJM --mpi "proc=12"

export OMP_NUM_THREADS=16
mpiexec ./sol0
rm wk.*
```

計算と通信のオーバーラップ



計算と通信のオーバーラップ





行列ベクトル積

- リナンバリング：■⇒■の順番に番号を振り直す
- 外点情報を通信する
- 同期を待たずに■の計算を実施し通信とオーバーラップ
- 通信の同期
- 続いて■の計算を実施する

q=Apの部分：src0

```
      call SOLVER_SEND_RECV_3                                          &
     &   ( N, NP, NEIBPETOT, NEIBPE, STACK_IMPORT, NOD_IMPORT,         &
     &     STACK_EXPORT, NOD_EXPORT, WS, WR, WW(1,P) , SOLVER_COMM,    &
     &     my_rank)

!$omp parallel do private (j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
      do j= 1, N
          X1= WW(3*j-2,P)
          X2= WW(3*j-1,P)
          X3= WW(3*j  ,P)
        WVAL1= D(9*j-8)*X1 + D(9*j-7)*X2 + D(9*j-6)*X3
        WVAL2= D(9*j-5)*X1 + D(9*j-4)*X2 + D(9*j-3)*X3
        WVAL3= D(9*j-2)*X1 + D(9*j-1)*X2 + D(9*j  )*X3
        do k= INL(j-1)+1, INL(j)
           i= IAL(k)
          X1= WW(3*i-2,P)
          X2= WW(3*i-1,P)
          X3= WW(3*i  ,P)
          WVAL1= WVAL1 + AL(9*k-8)*X1 + AL(9*k-7)*X2 + AL(9*k-6)*X3
          WVAL2= WVAL2 + AL(9*k-5)*X1 + AL(9*k-4)*X2 + AL(9*k-3)*X3
          WVAL3= WVAL3 + AL(9*k-2)*X1 + AL(9*k-1)*X2 + AL(9*k  )*X3
        enddo
        do k= INU(j-1)+1, INU(j)
           i= IAU(k)
          X1= WW(3*i-2,P)
          X2= WW(3*i-1,P)
          X3= WW(3*i  ,P)
          WVAL1= WVAL1 + AU(9*k-8)*X1 + AU(9*k-7)*X2 + AU(9*k-6)*X3
          WVAL2= WVAL2 + AU(9*k-5)*X1 + AU(9*k-4)*X2 + AU(9*k-3)*X3
          WVAL3= WVAL3 + AU(9*k-2)*X1 + AU(9*k-1)*X2 + AU(9*k  )*X3
        enddo

        WW(3*j-2,Q)= WVAL1
        WW(3*j-1,Q)= WVAL2
        WW(3*j  ,Q)= WVAL3
      enddo
```

q=Apの部分：src0m（1/3）

```
      do neib= 1, NEIBPETOT
        istart= STACK_EXPORT(neib-1)
        inum  = STACK_EXPORT(neib  ) - istart
!$omp parallel do private (ii)
        do k= istart+1, istart+inum
              ii   = 3*NOD_EXPORT(k)
           WS(3*k-2)= WW(ii-2,P)
           WS(3*k-1)= WW(ii-1,P)
           WS(3*k  )= WW(ii  ,P)
        enddo
        call MPI_ISEND (WS(3*istart+1), 3*inum,MPI_DOUBLE_PRECISION,    &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, req1(neib),    &
     &                  ierr)
      enddo

      do neib= 1, NEIBPETOT
        istart= STACK_IMPORT(neib-1)
        inum  = STACK_IMPORT(neib  ) - istart
        call MPI_IRECV (WW(3*(istart+N)+1,P), 3*inum,                   &
     &                  MPI_DOUBLE_PRECISION,                           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD,                &
     &                  req1(neib+NEIBPETOT), ierr)
      enddo

!C
!C-- Pure Inner Nodes

!$omp parallel do private (j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
      do j= 1, Ninn
           X1= WW(3*j-2,P)
           X2= WW(3*j-1,P)
           X3= WW(3*j  ,P)
        WVAL1= D(9*j-8)*X1 + D(9*j-7)*X2 + D(9*j-6)*X3
        WVAL2= D(9*j-5)*X1 + D(9*j-4)*X2 + D(9*j-3)*X3
        WVAL3= D(9*j-2)*X1 + D(9*j-1)*X2 + D(9*j  )*X3
```

q=Apの部分：src0m（2/3）

```
!C
!C-- Pure Inner Nodes

!$omp parallel do private (j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
      do j= 1, Ninn
           X1= WW(3*j-2,P)
           X2= WW(3*j-1,P)
           X3= WW(3*j  ,P)
        WVAL1= D(9*j-8)*X1 + D(9*j-7)*X2 + D(9*j-6)*X3
        WVAL2= D(9*j-5)*X1 + D(9*j-4)*X2 + D(9*j-3)*X3
        WVAL3= D(9*j-2)*X1 + D(9*j-1)*X2 + D(9*j  )*X3
        do k= INL(j-1)+1, INL(j)
           i= IAL(k)
          X1= WW(3*i-2,P)
          X2= WW(3*i-1,P)
          X3= WW(3*i  ,P)
          WVAL1= WVAL1 + AL(9*k-8)*X1 + AL(9*k-7)*X2 + AL(9*k-6)*X3
          WVAL2= WVAL2 + AL(9*k-5)*X1 + AL(9*k-4)*X2 + AL(9*k-3)*X3
          WVAL3= WVAL3 + AL(9*k-2)*X1 + AL(9*k-1)*X2 + AL(9*k  )*X3
        enddo
        do k= INU(j-1)+1, INU(j)
           i= IAU(k)
          X1= WW(3*i-2,P)
          X2= WW(3*i-1,P)
          X3= WW(3*i  ,P)
          WVAL1= WVAL1 + AU(9*k-8)*X1 + AU(9*k-7)*X2 + AU(9*k-6)*X3
          WVAL2= WVAL2 + AU(9*k-5)*X1 + AU(9*k-4)*X2 + AU(9*k-3)*X3
          WVAL3= WVAL3 + AU(9*k-2)*X1 + AU(9*k-1)*X2 + AU(9*k  )*X3
        enddo

        WW(3*j-2,Q)= WVAL1
        WW(3*j-1,Q)= WVAL2
        WW(3*j  ,Q)= WVAL3
      enddo

      call MPI_WAITALL (2*NEIBPETOT, req1, sta1, ierr)      ここで同期をとる
```

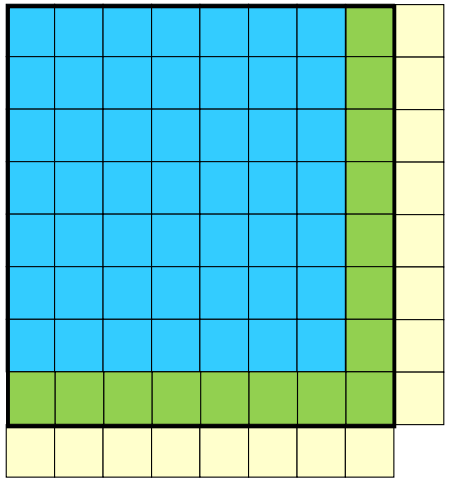
q=Apの部分：src0m（3/3）

```
!C
!C-- Boundary Nodes

!$omp parallel do private (j,k,i,X1,X2,X3,WVAL1,WVAL2,WVAL3)
      do j= Ninn+1, N
           X1= WW(3*j-2,P)
           X2= WW(3*j-1,P)
           X3= WW(3*j  ,P)
        WVAL1= D(9*j-8)*X1 + D(9*j-7)*X2 + D(9*j-6)*X3
        WVAL2= D(9*j-5)*X1 + D(9*j-4)*X2 + D(9*j-3)*X3
        WVAL3= D(9*j-2)*X1 + D(9*j-1)*X2 + D(9*j  )*X3
        do k= INL(j-1)+1, INL(j)
           i= IAL(k)
          X1= WW(3*i-2,P)
          X2= WW(3*i-1,P)
          X3= WW(3*i  ,P)
          WVAL1= WVAL1 + AL(9*k-8)*X1 + AL(9*k-7)*X2 + AL(9*k-6)*X3
          WVAL2= WVAL2 + AL(9*k-5)*X1 + AL(9*k-4)*X2 + AL(9*k-3)*X3
          WVAL3= WVAL3 + AL(9*k-2)*X1 + AL(9*k-1)*X2 + AL(9*k  )*X3
        enddo
        do k= INU(j-1)+1, INU(j)
           i= IAU(k)
          X1= WW(3*i-2,P)
          X2= WW(3*i-1,P)
          X3= WW(3*i  ,P)
          WVAL1= WVAL1 + AU(9*k-8)*X1 + AU(9*k-7)*X2 + AU(9*k-6)*X3
          WVAL2= WVAL2 + AU(9*k-5)*X1 + AU(9*k-4)*X2 + AU(9*k-3)*X3
          WVAL3= WVAL3 + AU(9*k-2)*X1 + AU(9*k-1)*X2 + AU(9*k  )*X3
        enddo

        WW(3*j-2,Q)= WVAL1
        WW(3*j-1,Q)= WVAL2
        WW(3*j  ,Q)= WVAL3
      enddo
```

計算と通信のオーバーラップ

With Reordering (current)

```
      call MPI_Isend
      call MPI_Irecv

      do i= 1, Ninn
        (calculations)
      enddo

      call MPI_Waitall

      do i= Ninn+1, Nall
        (calculationas)
      enddo
```



Without Reordering

```
      call MPI_Isend
      call MPI_Irecv

      do i= 1, Nall
        if (INNflag(i).eq.1) then
          (calculations)
        endif
      enddo

      call MPI_Waitall

      do i= 1, Nall
        if (INNflag(i).eq.0) then
          (calculations)
        endif
      enddo
```

計算結果：行列ベクトル積（q=Ap）一回あたりの計算時間（sec.）

12ノード（プロセス）使用時

オーバーラップさせる計算時間が充分ないと効果が出ない
⇒1プロセスあたりの計算量が大きい必要有り

	(npx,npy,npz)(ndx,ndy,ndz)	
	(256,256,384) (2,2,3)	(200,200,300) (2,2,3)
sol0	1.06×10^{-1}	3.75×10^{-2}
sol0m	7.92×10^{-2}	3.76×10^{-2}

まとめ

- 様々なケースで実行して見よ
  - 問題サイズ
  - プロセスあたりスレッド数
  - リオーダリング無しの場合
- 計算と通信のオーバーラップ
  - 行列ベクトル積程度ではあまり性能の向上は期待できない
  - 陽解法でループあたりの計算量が大きい場合は大幅な向上が期待できる場合がある
  - 複雑な前処理がはいった場合の対処法は研究の途上にある
    - 頭を使う