3 次元流体コードの並列・ベクトルAMR化

報告書

平成２０年８月４日

Advanced Algorithm & Systems

１．概要

国内研究所で開発中の３次元流体解析コード（ナビエ・ストークス方程式）に対して、ベクトル化と MPI ライブラリ等による並列化、計算領域分割法として AMR（Adaptive Mesh Refinement；適合細分化格子法）の実装等のプログラム整備を実施することを主な目的とする。

上記の目的のうち、本報告書は、２分割法および３分割法による AMR の実装に関するものである。

検証は、分割領域を外部で指定することにより分割を行い、分割された速度ベクトルを可視化することによって、分割が正しく行われていることを確認した。

プログラムについての開発用メモを含めて報告する。

２．検証方法

AMR 化対象プログラムの速度ベクトルの初期条件は、以下の図のようになっている。
境界条件は、ｘ方向 EW、ｙ方向ＮＳ、ｚ方向ＴＢとすれば、

Ｅ、Ｗ、Ｎ、Ｓ平面で u=v=w＝０（ディレクレ境界）

Ｔ、Ｂ平面で周期境界

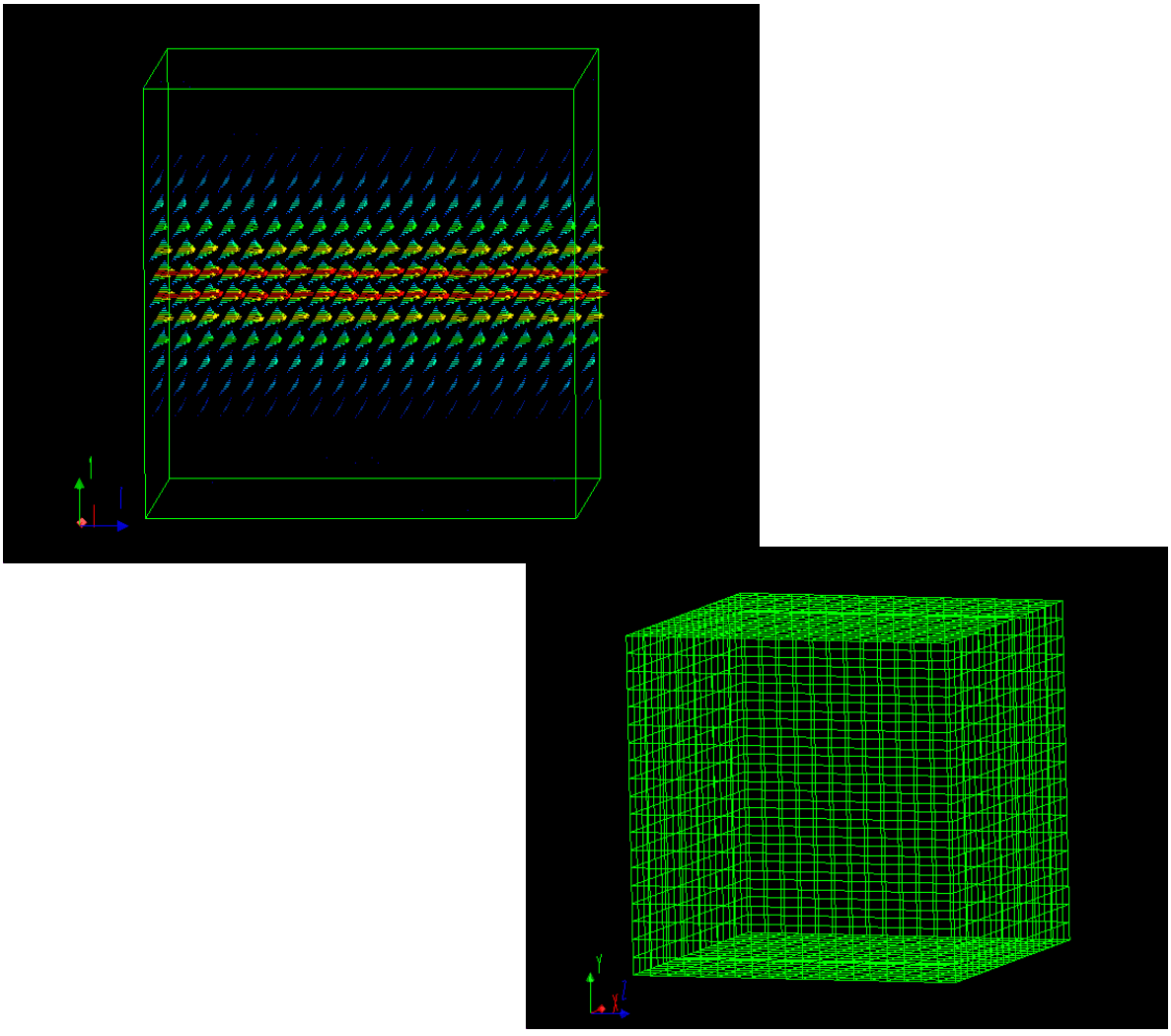
となっている。現在の条件で時間積分をすると解は初期条件と変わらないので各変数の誤差は変化しない。現在、誤差を「密度」で評価しているので、当然誤差がほとんどゼロとなり、分割する必要がない。そこで、AMR の検証方法として、「分割する領域および再分割領域」を誤差とは無関係に設定して計算し、その結果を以下の二種類で図化する。

（１） 下記のような流速ベクトル図
（２） 分割セル図

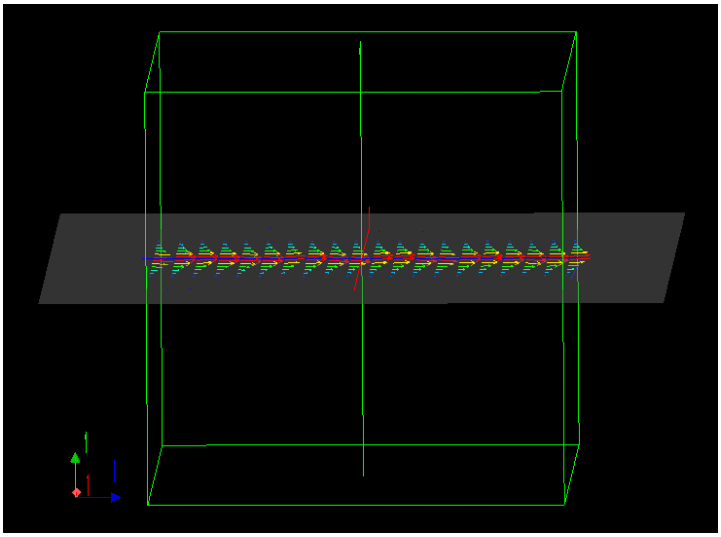
以上の手順を２分割モデルと３分割モデルについて行う。

以下は、１３ｘ１３ｘ１３のセルに３セルの糊代をつけたモデルの流速ベクトルで、格子は表示用に等間隔にしている。

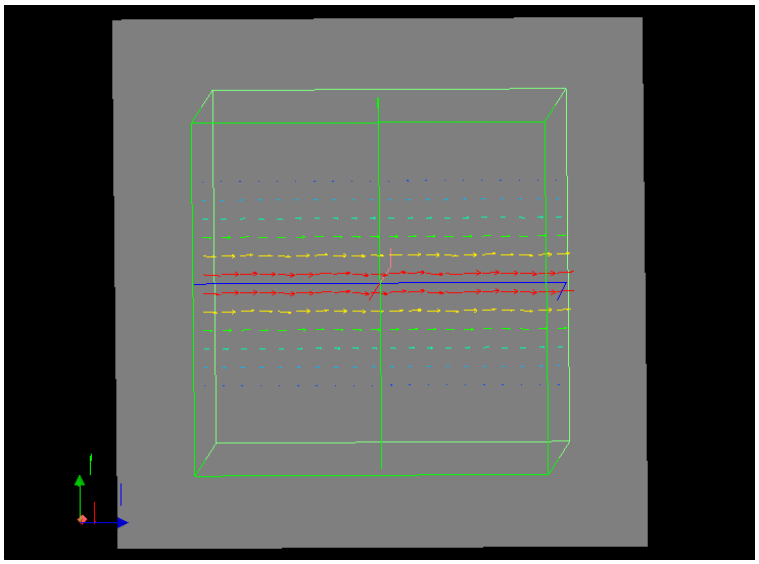
全体



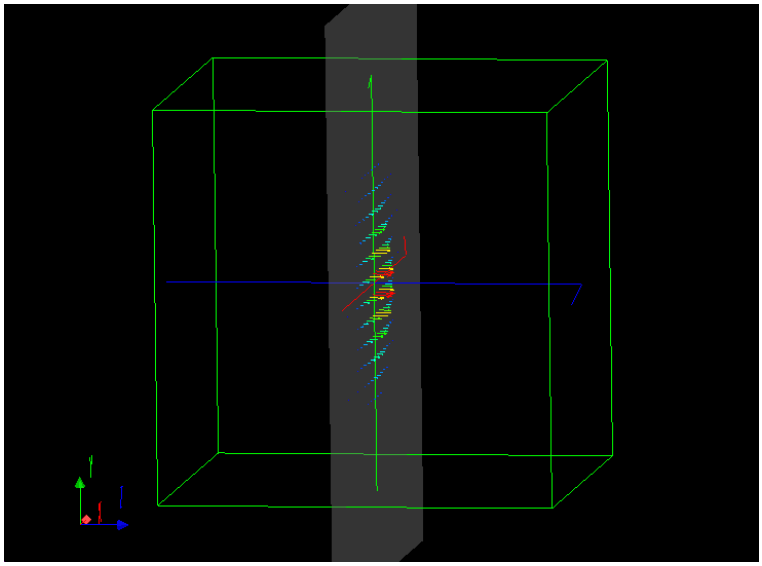
X-Z 平面



Y-Z 平面

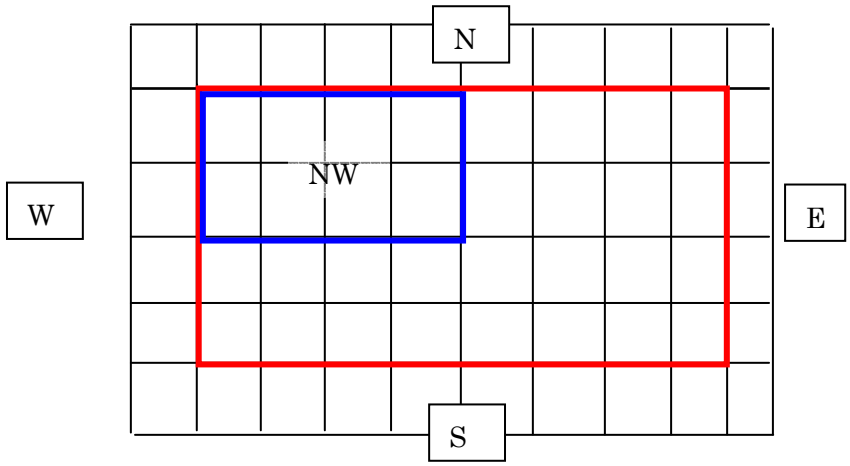


X-Y 平面

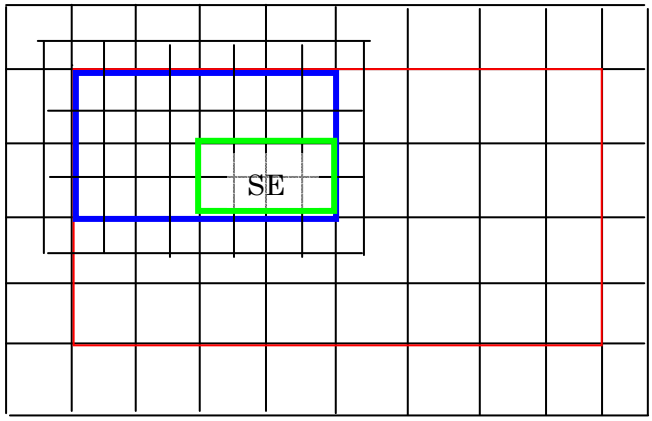


２次元例（説明用で糊代は１セル分）：　以下のように分割する設定をして検証する

lebe_0

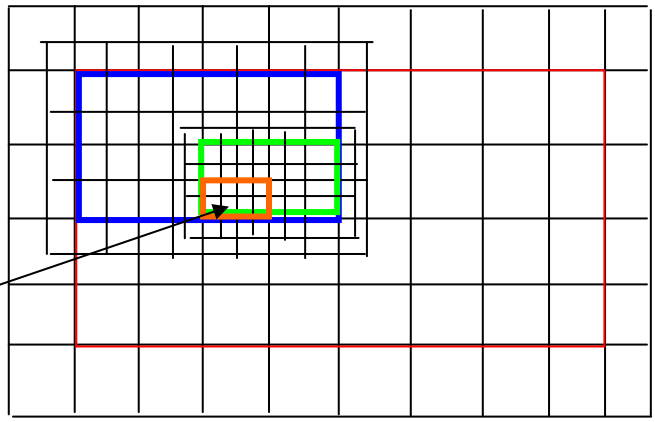


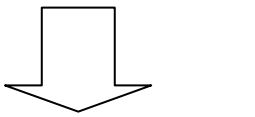
lebe_1　NW



lebe_2　SE

SW





lebe_3　SW

３．２分割モデルの検証

（１）格子サイズ

NGx1 = 3, NX = 4 , NGx2=3
NGy1 = 3, NY = 4 , NGy2=3
NGz1 = 3, NZ = 4 , NGz2=3

分割幅は、X と Y 方向に１、Z 方向にπ

（２）分割指定

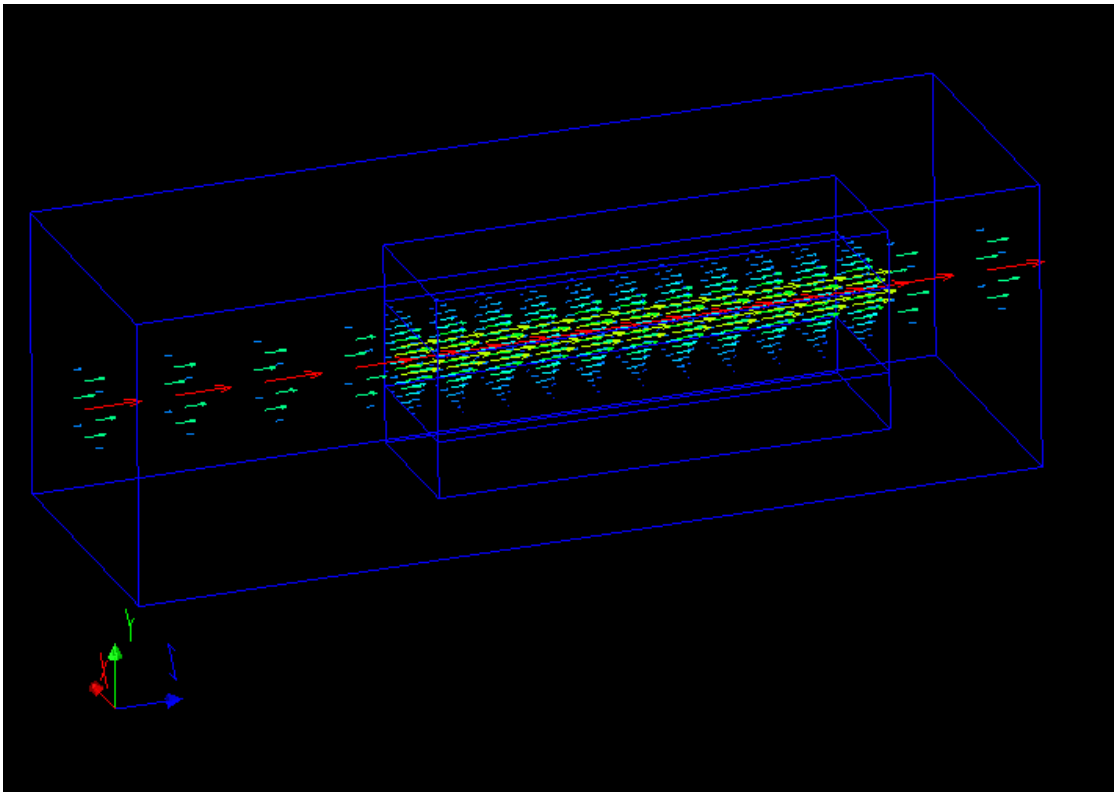
level_1: 親（level_0）の NW_T と SW_T を２分割

level_2: level_1:NS_T の NW_T と SW_T を２分割

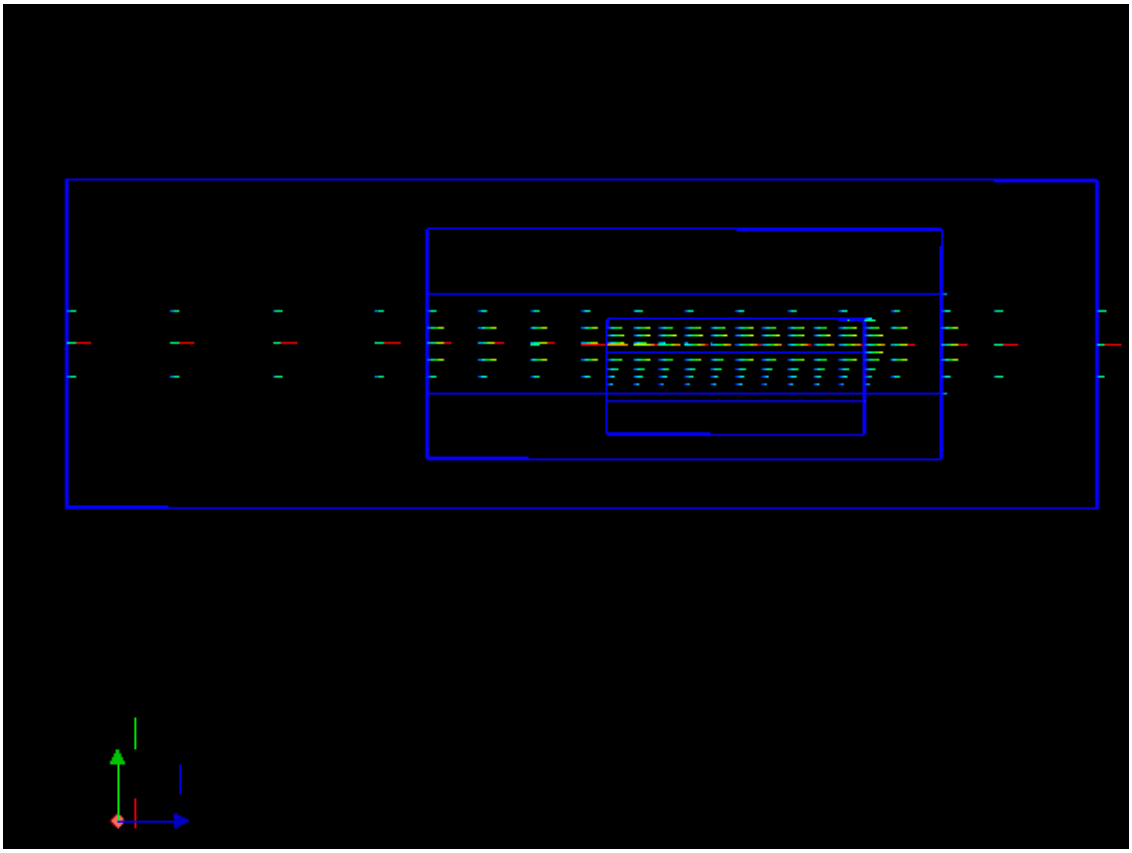
（３）初期条件

初期条件は、頂いた設定法。

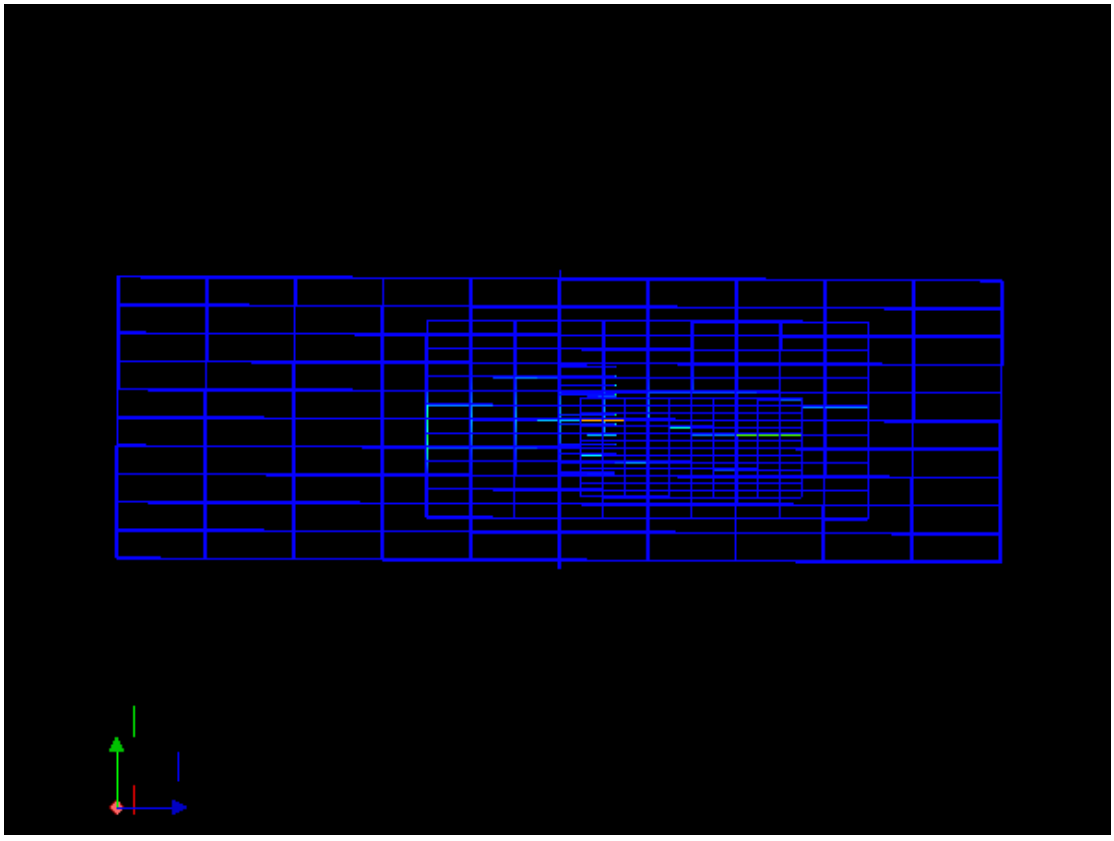
レベル０、１、２の重ね合わせ



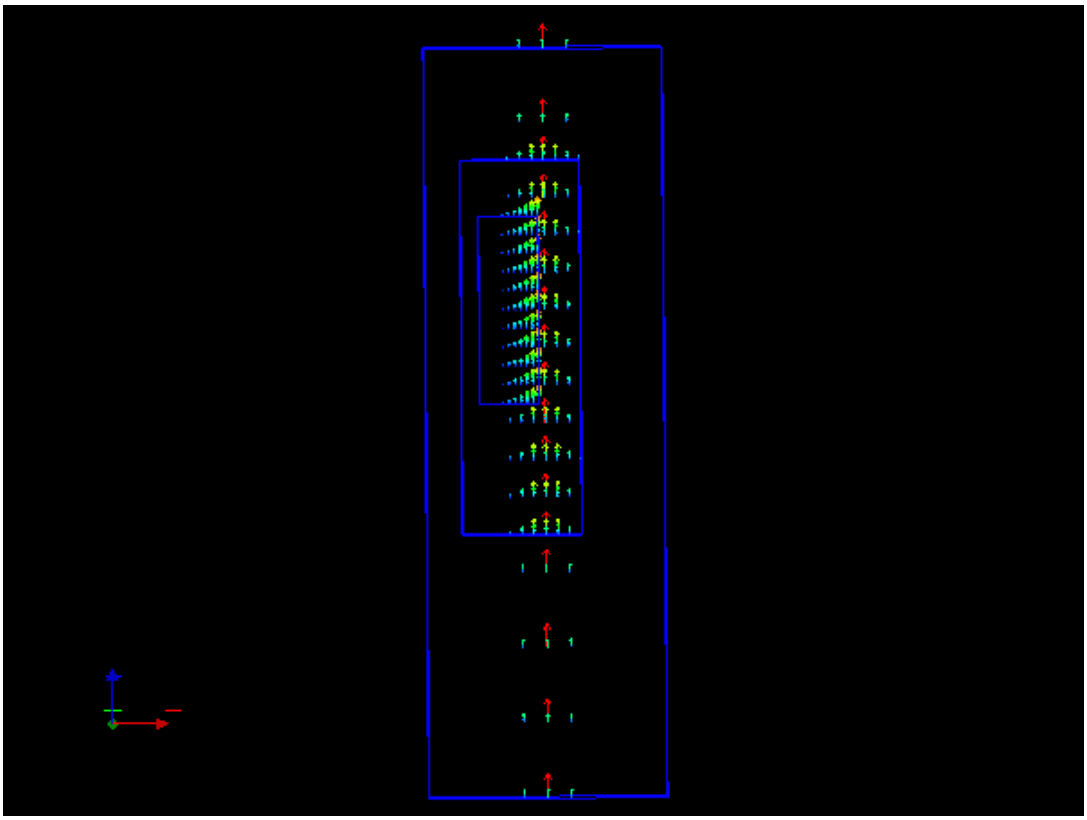
レベル０、１、２の重ね合わせ（Z-Y 平面）



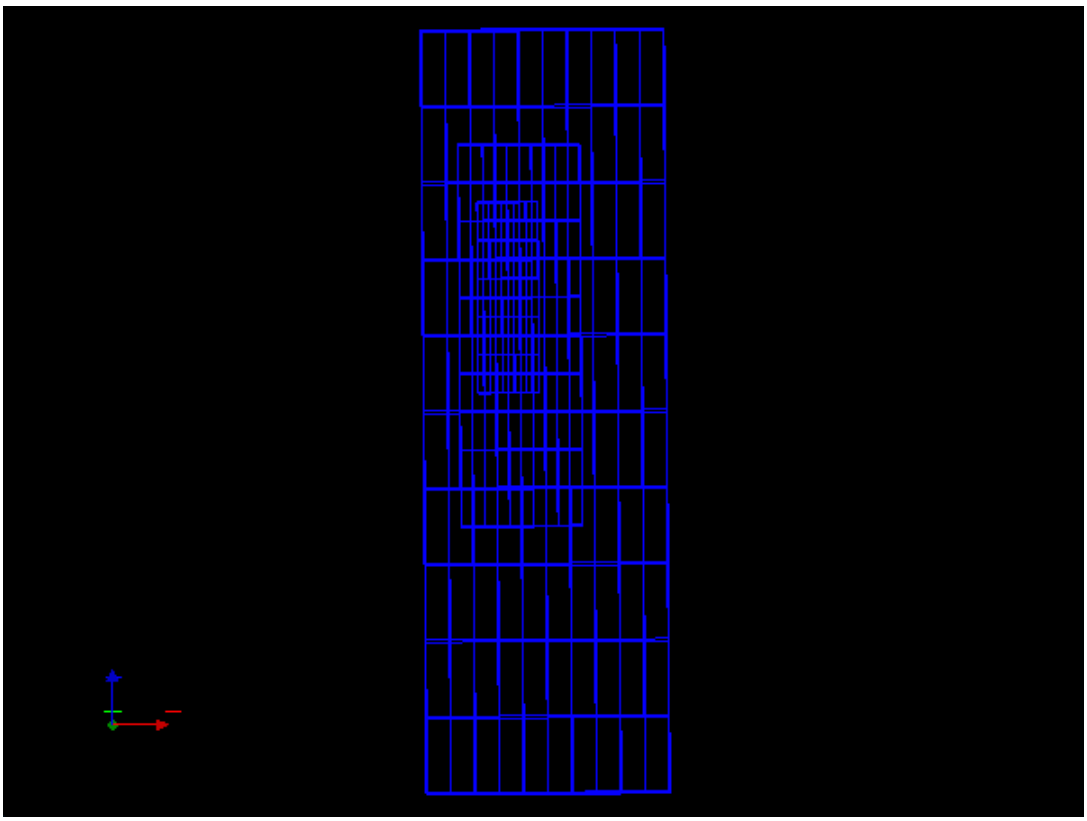
レベル０、１、２の重ね合わせ（Z-Y 平面）：メッシュ分割図



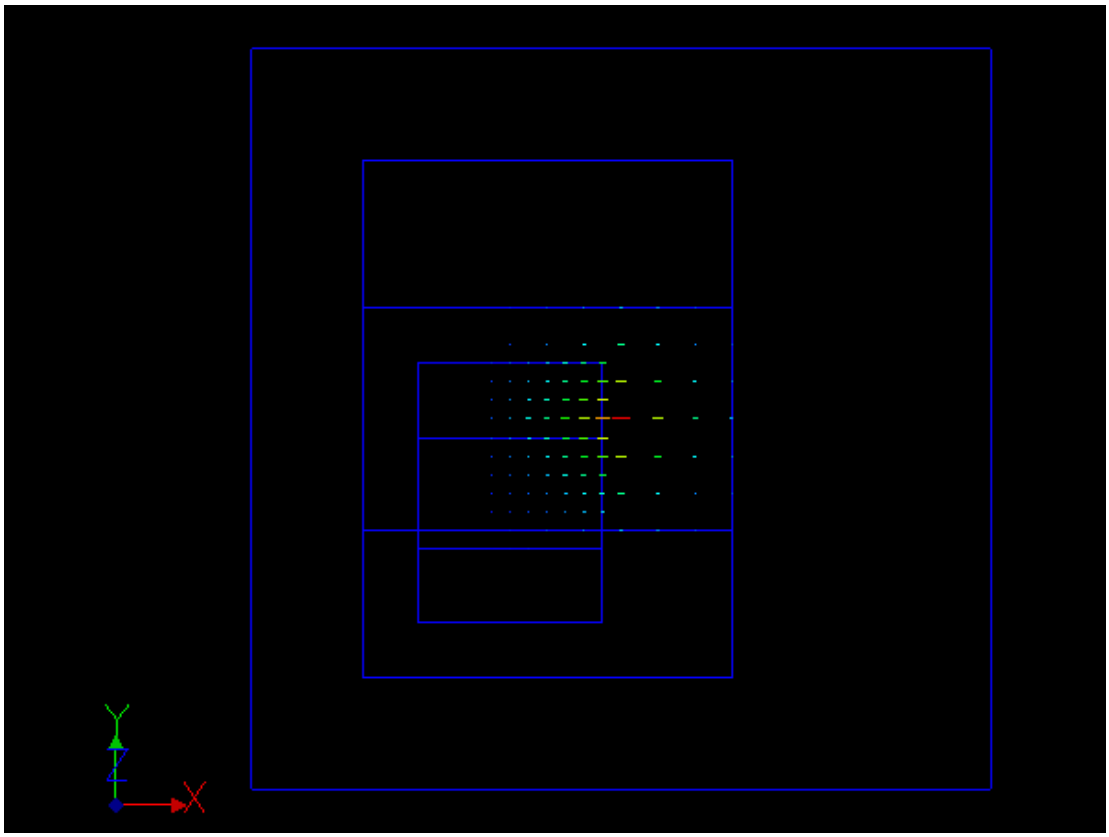
レベル０、１、２の重ね合わせ（Y-Z 平面）



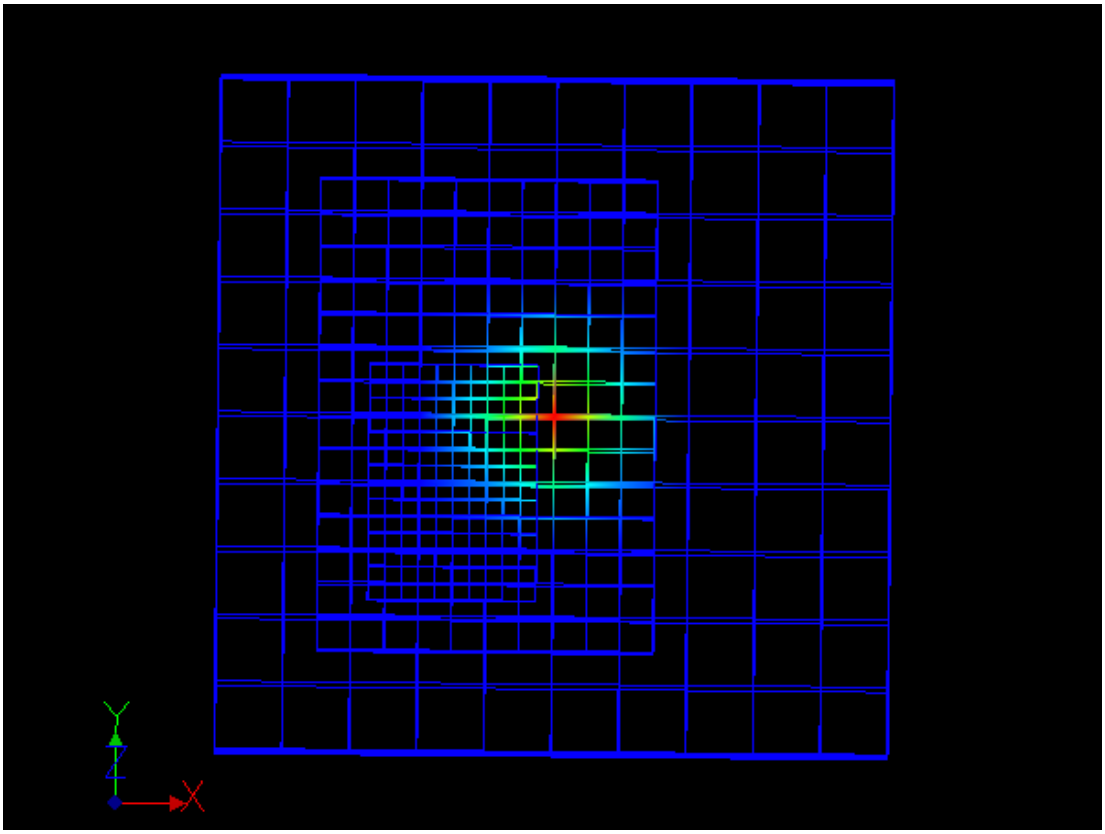
レベル０、１、２の重ね合わせ（Y-Z 平面）：メッシュ分割図



レベル０、１、２の重ね合わせ（X-Y 平面）



レベル０、１、２の重ね合わせ（X-Y 平面）：メッシュ分割図



４．３分割モデルの検証

（１）格子サイズ

NGx1 = 3, NX = 3, NGx2=3
NGy1 = 3, NY = 3, NGy2=3
NGz1 = 3, NZ = 3, NGz2=3

分割幅は、X と Y 方向に１、Z 方向にπ

（２）分割指定

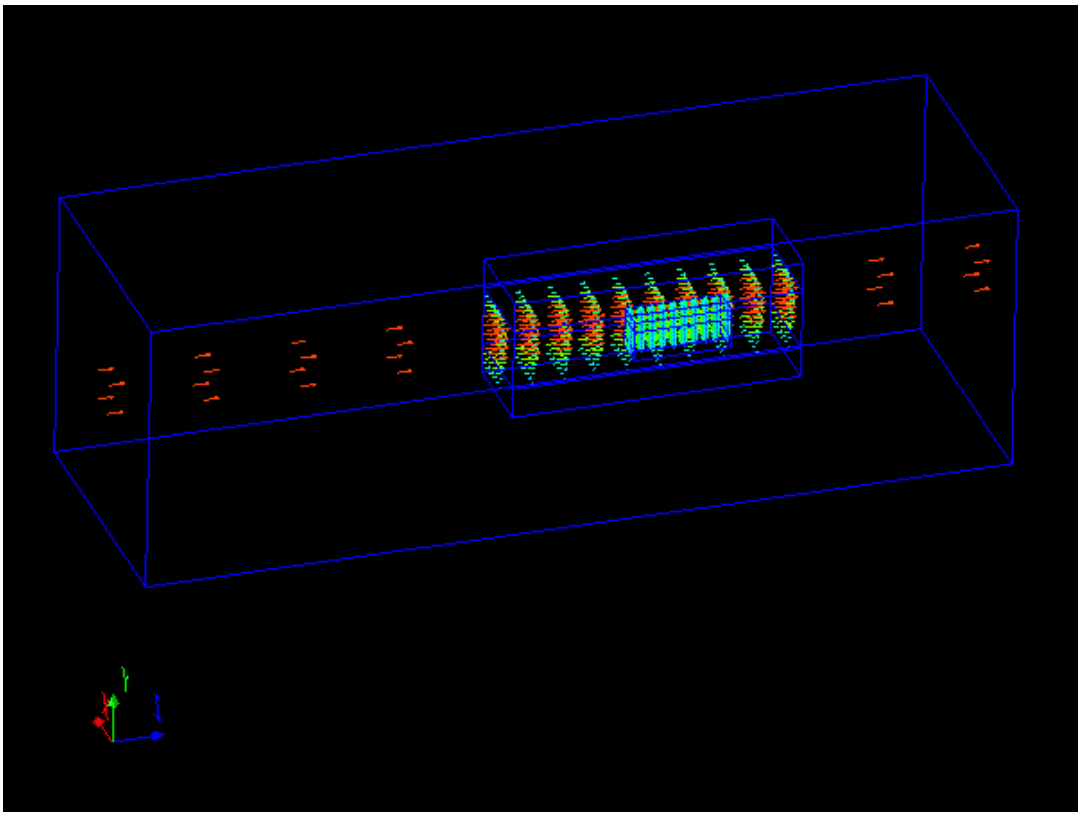
level_1:　親（level_0）の NW_T と_W_T を３分割

level_2:　 level_1:NS_T の NW_T と_W_T を３分割

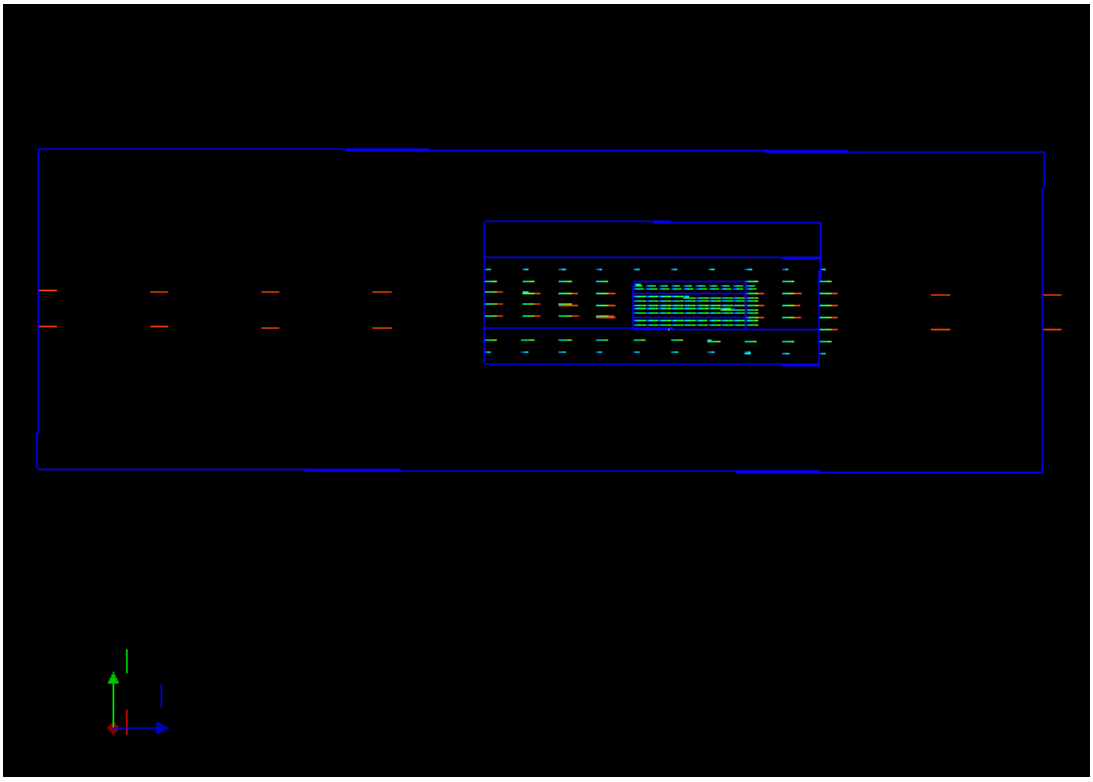
（３）初期条件

初期条件は、頂いた設定法だが、分割が荒いので、放物線になっていない。最初からもっと細かくすれば、きれいな放物線分布となる。

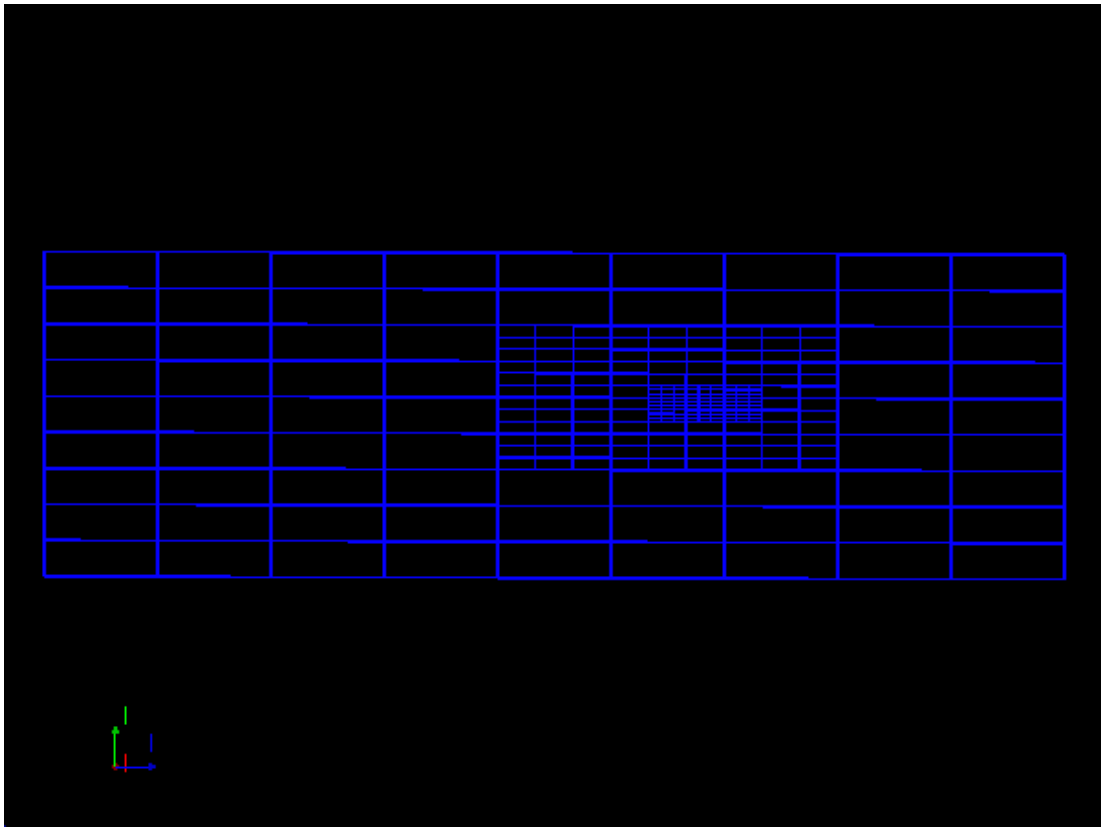
レベル０、１、２の重ね合わせ



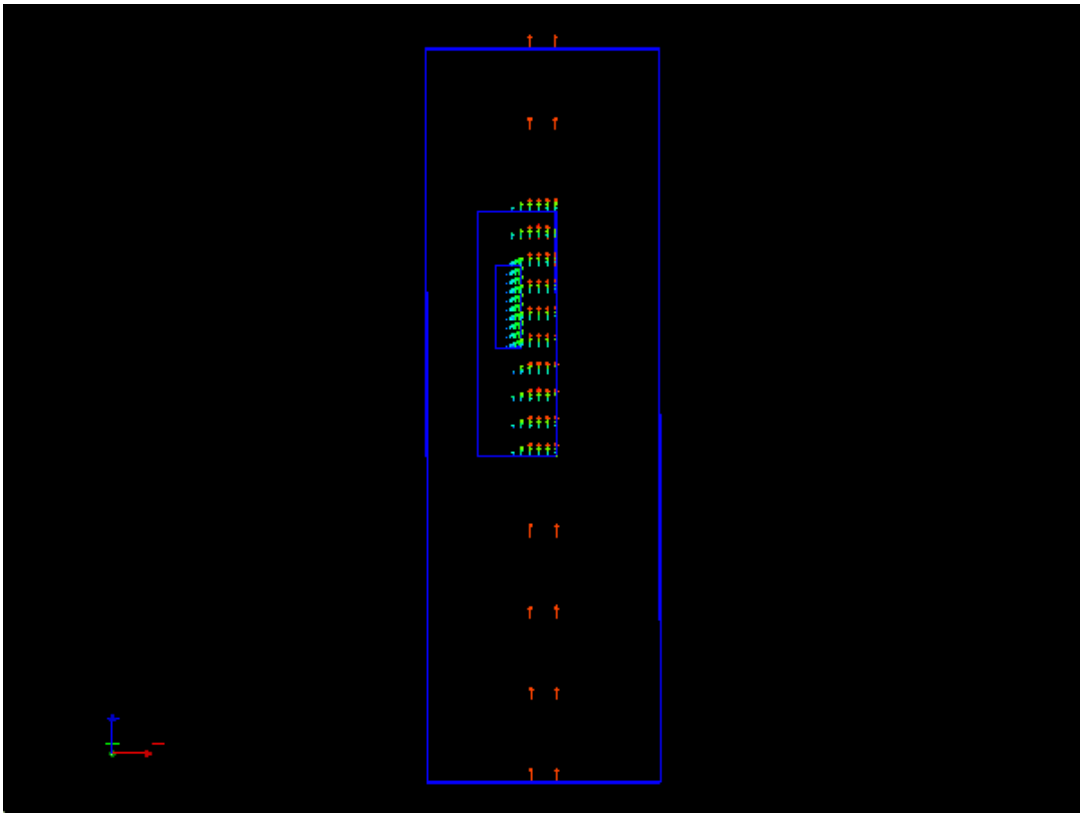
レベル０、１、２の重ね合わせ（Z-Y 平面）



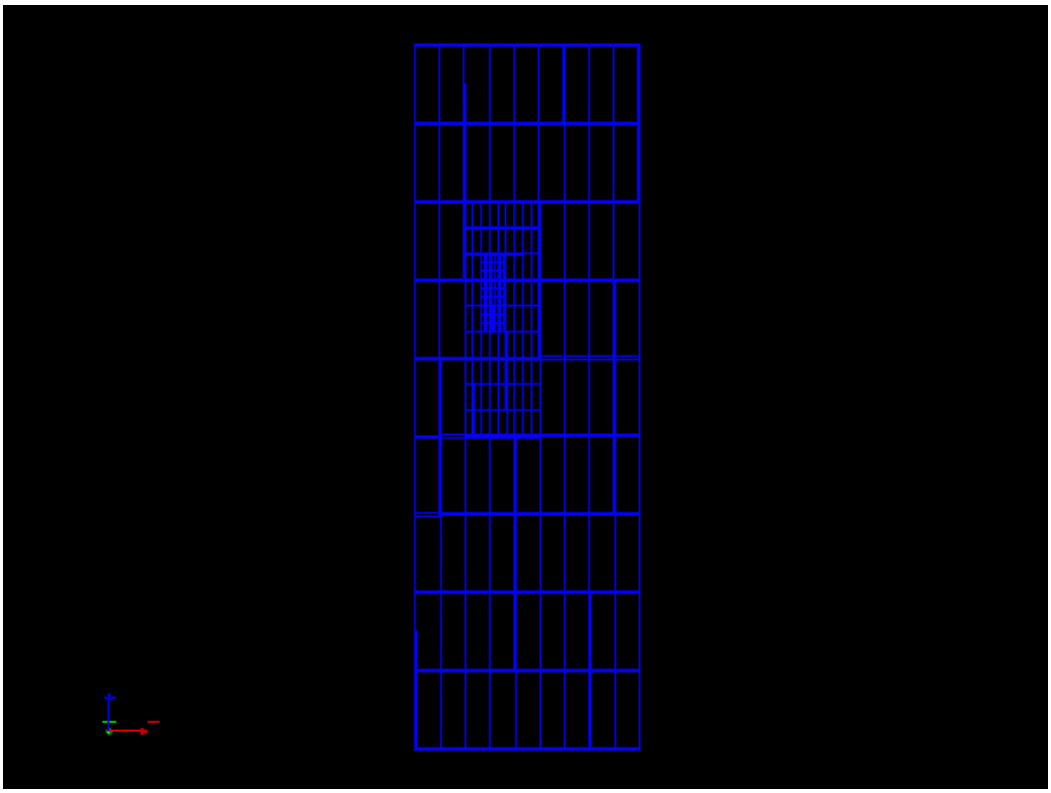
レベル０、１、２の重ね合わせ（Z-Y 平面）：メッシュ分割図



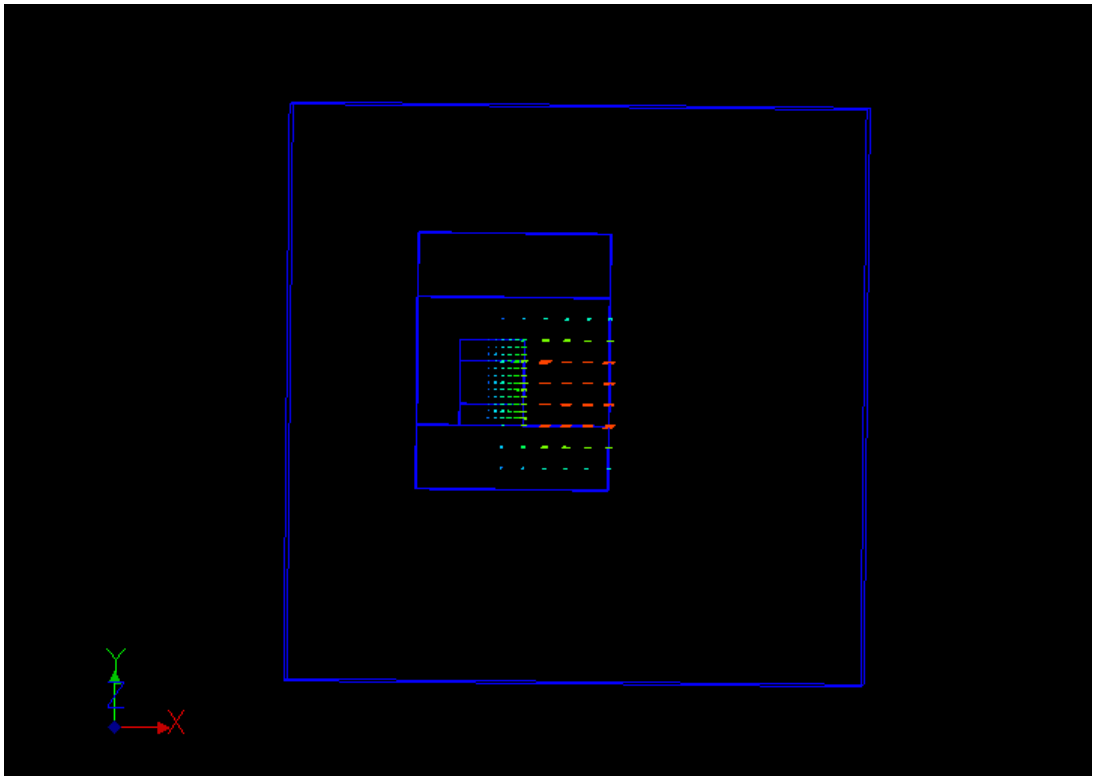
レベル０、１、２の重ね合わせ（X-Z 平面）



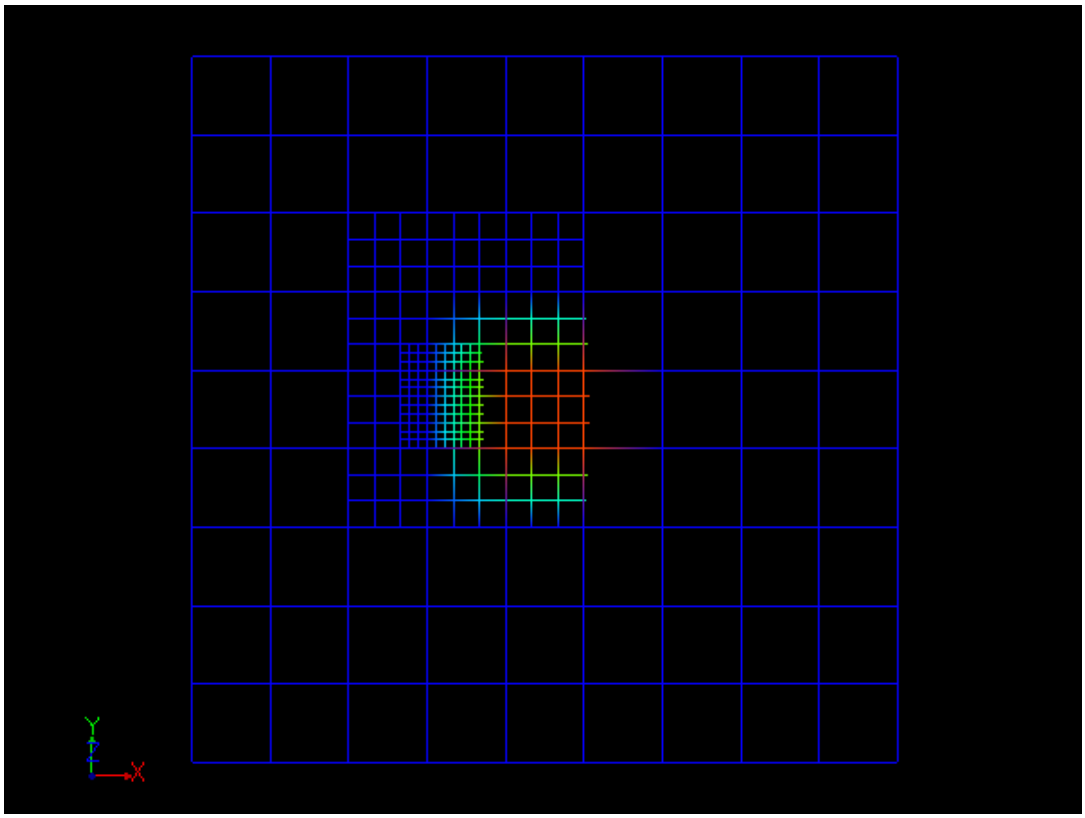
レベル０、１、２の重ね合わせ（X-Z 平面）：メッシュ分割図



レベル０、１、２の重ね合わせ（X-Y 平面）



レベル０、１、２の重ね合わせ（X-Y 平面）：メッシュ分割図



5　プログラムに関するメモ

（１）　２分割および３分割の指定フラグ

２分割および３分割の選択用にフラグを用意した。

module_system_domainsize.f90

```
!@ame@ 2008.7 AMR(二分割)orAMR(三分割)
  integer, parameter:: flg_AMRdiv = 2
```

（２）　作成サブルーチン

新規に作成したサブルーチンは、すべて module_grids_NS3DComp.f90 に記述した。

サブルーチン名
guard_fill_div2
guard_fill_div3
copy_parent2current_div2
copy_parent2current_div3
make_grids_div2
make_grids_div3
copy_current2parent_div2
copy_current2parent_div3
copy_parent2current_div2
copy_parent2current_div3

（３）　強制的な分割指定部分

最初にレベル０を初期化する際に、分割領域を１つ指定する。

subroutine march (rogram_cNS3D_duct_OR.f90)

```
!@ame@
!      Grid_pointer(0)%Ptr%icoord =1
  do iloop = 0, IGdomain_active
     Grid_pointer(iloop)%Ptr%level = 0
                    Grid_pointer(iloop)%Ptr%idomain_parent(1) = iloop
                    Grid_pointer(iloop)%Ptr%idomain_parent(2) = IGdomain_map_NW_T
  end do
```

誤差を使って判断する分岐があるので、テスト用に err_max(ib) = 0.d0 として誤差は分割を考慮するパラメータから除く。

subroutine error_test(Ptr) (rogram_cNS3D_duct_OR.f90)

```
if( err9 .gt. err_max(ib) ) then
   err_max(ib) = err9
end if
   err_max(ib) = 0.d0 !@ame@@test@ always 0
```

２分割の場合、IGdomain_active が新規に作成されるたびに、分割位置を以下の位置に指定する。

subroutine refine_check_detail_div2：rogram_cNS3D_duct_OR.f90

```
!@@@
! フラグは、error_test で設定、現在テスト用
      Grid_pointer(IGdomain_active)%Ptr%refine_NW_T = 1  !@ame@ test setting
      Grid_pointer(IGdomain_active)%Ptr%refine_SW_T = 1
!@@@
```

３分割の場合、IGdomain_active が新規に作成されるたびに、分割位置を以下の位置に指定する。

subroutine refine_check_detail_div3：rogram_cNS3D_duct_OR.f90

```
!@@@
! フラグは、error_test で設定、現在テスト用
      Grid_pointer(IGdomain_active)%Ptr%refine_NW_T = 1  !@ame@ test setting
      Grid_pointer(IGdomain_active)%Ptr%refine__W_T = 1
!@@@
```

（４） 分割部分に関連しないところでの不具合部分

赤字のサブルーチンは直接分割サブルーチンに関係しないのでコメントアウトしている。しないと無限大の解が部分的に発生する。

subroutine march: program_cNS3D_duct_OR.f90

```
    do istep = 1, 4   ! RKGのステップ実行
         write(iuLog,cf5010) '[march] istep = ', istep, ' <<<<<<<<<'

       if(i_debug_printall .gt. 0) &
    &     write(iuLog,cf5010) '[march] istep = ', istep, ' <<<<<<<<<'

!       call rhs_grids_NS3DComp  ! 基本部分はmodule variables に所属

!       call impose_BC_RHS    ! プログラム毎に違う可能性
                              ! 頻出なら module_variables に処属させても良い

        call step(istep)      ! 時間発展（ここではRKG）の各領域についての作用


!       call impose_BC        ! プログラム毎に違う可能性
                              ! 頻出なら module_variables に処属させても良い
      if(flg_AMRdiv == 2) then
      elseif(flg_AMRdiv == 3) then
        call guard_fill_div3   ! 全ての階層、領域について境界糊代のコピー、未完成
      else
      end if

    end do  !istep
```

（４）検証用の格子サイズ指定

検証用に格子サイズを少なくして、分割結果を可視化して判断できるようにしてある。

module system_domainsize:　module_system_domainsize.f90

```
integer, parameter:: Nx =  3, NGx1 = 3, NGx2 = 3, NWx = (Nx+NGx1+NGx2+1)
integer, parameter:: Ny =  3, NGy1 = 3, NGy2 = 3, NWy = (Ny+NGy1+NGy2+1)
integer, parameter:: Nz =  3, NGz1 = 3, NGz2 = 3, NWz = (Nz+NGz1+NGz2+1)
```

（５）　ファイル

デバッグ出力用ファイルを open 文で指定してある。

```
program NS3D_KH   :   program_cNS3D_duct_OR.f90


!@ame@2008.4
  open(iuLog,file='debug.log')
!@ame@2008.4
```

subroutine march :　program_cNS3D_duct_OR.f90

```
!@ame@ 結果を打ち出す => start
!毎回ファイルをcloseして最後のマーチングの結果を掃き出す
         do idom = 0, IGdomain_active
           if(idom == 0) then
              open (78, file= 'res00.out')
           else if(idom == 1) then
              open (78, file= 'res01.out')
           else if(idom == 2) then
              open (78, file= 'res02.out')
           else if(idom == 3) then
              open (78, file= 'res03.out')
           else if(idom == 4) then
              open (78, file= 'res04.out')
           else if(idom == 5) then
              open (78, file= 'res05.out')
           end if

         write(78,'(5x,3i5)')  NWx, NWy, NWz
!         write(78,'(a)')  '  i   j   k      x      y     z     u     v  w   den  enet'
           do k = -NGz1, Nz + NGz2
            do j = -NGy1, Ny + NGy2
             do i = -NGx1, Nx + NGx2
               write(78, '(3i5,3f15.6,999e25.16)') &
                         i,j,k, Grid_pointer(idom)%Ptr%cord1(i),Grid_pointer(idom)%Ptr%cord2(j), &
                         Grid_pointer(idom)%Ptr%cord3(k),Grid_pointer(idom)%Ptr%amo1(i,j,k), &
                         Grid_pointer(idom)%Ptr%amo2(i,j,k),Grid_pointer(idom)%Ptr%amo3(i,j,k), &
                         Grid_pointer(idom)%Ptr%den(i,j,k),Grid_pointer(idom)%Ptr%enet(i,j,k)

             end do
            end do
           end do
        close (78)

         end do
!@ame@ 結果を打ち出す <= end
```

（６） デバッグ用変数

分割後の格子を保存するために、配列 cord1, cord2, cord3 および分割セルの基準位置 upos1, upos2, upos3 を用意した。必要ないなら関連する箇所をすべて削除して問題ない。

```
                    module variables_NS3Dcomp   : module_variables_NS3DComp.f90

!    非直交一般座標系であれば、type(Grid_irregular)を利用して座標系を定義

     real(kind=pd):: u1, u2, u3 ! 非一様座標の場合の格子空間原点座標
     real(kind=pd):: upos1, upos2, upos3 !@ame@
     real(kind=pd):: du1, du2, du3 ! 非一様座標の場合の格子空間内格子点間隔
     real(kind=pd):: Lu1, Lu2, Lu3 ! 3方向への長さ（システムサイズ）
!@ame@
  real(kind=pd), dimension(-NGx1:Nx+NGx2):: cord1
  real(kind=pd), dimension(-NGy1:Ny+NGy2):: cord2
  real(kind=pd), dimension(-NGz1:Nz+NGz2):: cord3
!@ame@
```

cord1, cord2, cord3 の定義

```
                    subroutine refine_check   : module_grids_NS3DComp.f90


!@ame@ check用 分割領域の座標を保存 => start
           do k = -NGz1, Nz + NGz2
            do j = -NGy1, Ny + NGy2
             do i = -NGx1, Nx + NGx2
                Grid_pointer(iloop)%Ptr%cord1(i) = Grid_pointer(iloop)%Ptr%upos1 +
(i+NGx1)*Grid_pointer(iloop)%Ptr%du1
                Grid_pointer(iloop)%Ptr%cord2(j) = Grid_pointer(iloop)%Ptr%upos2 +
(j+NGy1)*Grid_pointer(iloop)%Ptr%du2
                Grid_pointer(iloop)%Ptr%cord3(k) = Grid_pointer(iloop)%Ptr%upos3 +
(k+NGz1)*Grid_pointer(iloop)%Ptr%du3
             end do
            end do
           end do
!@ame@ check用 分割領域の座標を保存 <= end
```

upos1, upos2, upos3 の定義

```
                    subroutine make_grids_div2   : module_grids_NS3DComp.f90  および
                        subroutine make_grids_div3   : module_grids_NS3DComp.f90

      if(i_shift .eq. 0) then
         Ptr2%upos1 = Ptr1%upos1 + Ptr1%du1*dble(NGx1)*0.5d0  !@ame@
         Ptr2%u1 = Ptr1%u1
      else
         Ptr2%upos1 = Ptr1%upos1 + Ptr1%Lu1 * 0.5d0 + Ptr1%du1*dble(NGx1)*0.5d0  !@ame@
         Ptr2%u1 = Ptr1%u1 + Ptr1%Lu1 * 0.5d0
      end if

      if(j_shift .eq. 0) then
         Ptr2%upos2 = Ptr1%upos2 + Ptr1%du2*dble(NGy1)*0.5d0  !@ame@
         Ptr2%u2 = Ptr1%u2
      else
         Ptr2%upos2 = Ptr1%upos2 + Ptr1%Lu2 * 0.5d0 + Ptr1%du2*dble(NGy1)*0.5d0  !@ame@
         Ptr2%u2 = Ptr1%u2 + Ptr1%Lu2 * 0.5d0
      end if

      if(k_shift .eq. 0) then
         Ptr2%upos3 = Ptr1%upos3 + Ptr1%du3*dble(NGz1)*0.5d0  !@ame@
         Ptr2%u3 = Ptr1%u3
      else
         Ptr2%upos3 = Ptr1%upos3 + Ptr1%Lu3 * 0.5d0 + Ptr1%du3*dble(NGz1)*0.5d0  !@ame@
         Ptr2%u3 = Ptr1%u3 + Ptr1%Lu3 * 0.5d0
      end if
```

６．　コンパイルと実行

コンパイルに関しては、受け取ったファイル名以外に新しくファイルを作成していないので、もとの Makefile でコンパイルできるはずである。

検証用に実行する際のファイル指定は、ファイル基盤番号を用いたファイル名を用いている。例えば F_FF01 に対応して fort.1 としている。