OpenFOAMによる
電子機器シミュレーション
その2

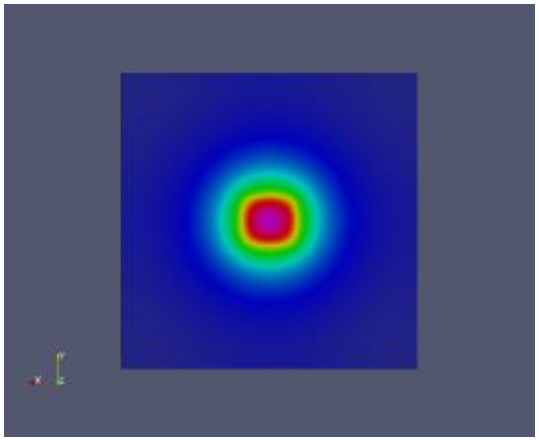
西　剛伺

前回確認したこと（検証数値実験）

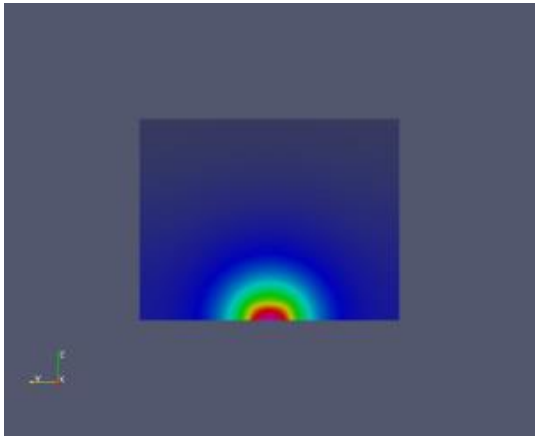
＜10秒後の結果＞　両者の結果が一致することを確認

ソルバA

底面の温度分布

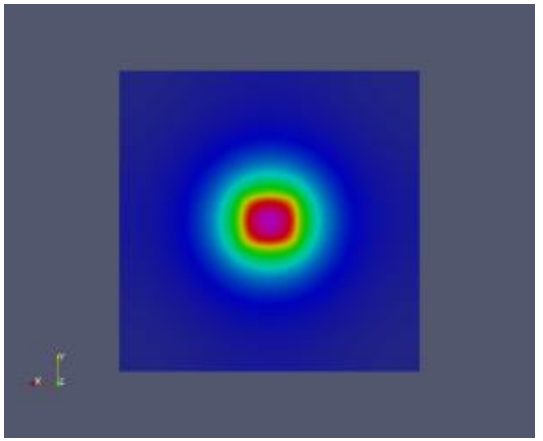


中央部断面の温度分布

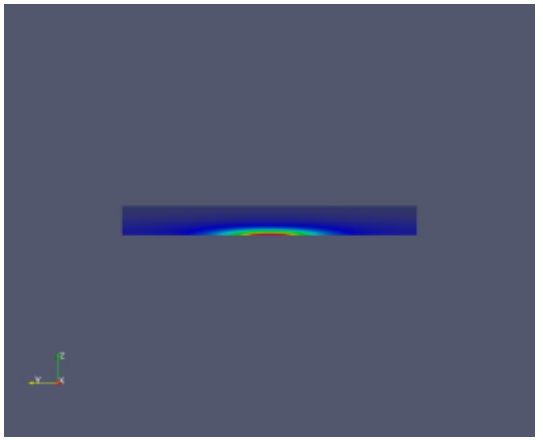


ソルバB

底面の温度分布



中央部断面の温度分布



温度値のチェック

特定点についてはprobeLocations
ユーティリティを使用.
実行後，probesフォルダが作成され，
0/Tファイルが生成される.

平均温度については
patchAverageユーティリティを使用.

probesDict

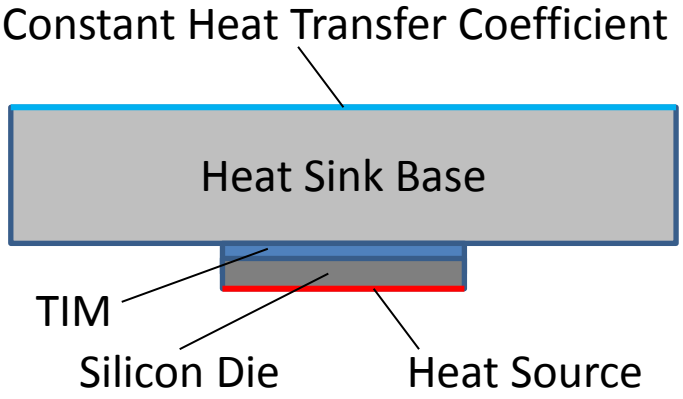
```
// Fields to be probed. runTime modifiable!
fields
(
    T
);

// Locations to be probed. runTime modifiable!
probeLocations
(
    (0.0 0.0 0.0)                    // Center
                :
);
```

probes/0/T

```
#        x       0.0          xxx2          xxx3
#        y       0.0          yyy2          yyy3
#        z       0.0          zzz2          zzz3
#      Time
           0       300           300           300
         0.1   302.521       302.452       302.292
         0.2   303.266       303.158       302.934
         0.3   303.722       303.594       303.338
                :
```

もう少し現実的な問題



シリコンダイにTIMを介してヒートシンクが装着されている
ケースを想定する.
サイズ及び熱物性値が部材ごとに異なる.
また，熱源は均一発熱でない場合もありうる.

異なる熱物性値はsetFieldsDictで指定できた.
あとは異なるサイズの部材についてメッシュを切ることだが,
実用的な精度で結果を得るには・・・

- モデル領域を一つのblockとして表現
- 複数のblockを組み合わせてそれぞれ個別にgridを切る
- snappyHexMeshを活用する

温度伝導率の設定

DTという“テンソル変数”を
createFields.Hに定義.
値をsystem/setFieldsDictで設定.

```
volScalarield DT
(
    volSymmTensorField DT
    (
        IOobject
        (
            "DT",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
);
```

transportPropertiesの
記述からはDTを削除.

setFieldsDict

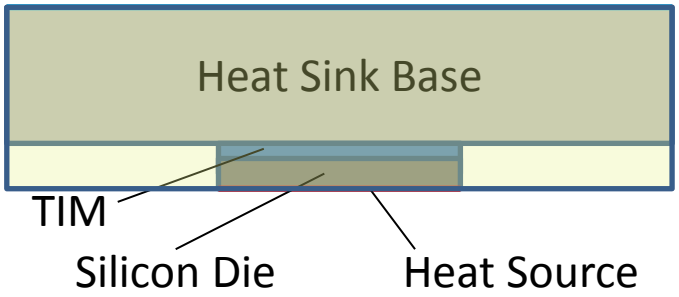
```
defaultFieldValues
(
        volScalarFieldValue ST 0
        volSymmTensorFieldValue DT (xxx 0 0 xxx 0 xxx)
);

regions
(
    boxToCell
    {
        box (-75 -75 0) (75 75 1.6);          // Motherboard

        fieldValues
        (
            volSymmTensorFieldValue DT (xxx 0 0 yyy 0 zzz)
        );
    }
        :
}
```

モデル領域を一つのblockとして表現

モデル領域を一つのblockとしてモデル化し，
空白部にはゼロの温度伝導率を与える（つまり，断熱）



＜利点＞
－　block間境界の整合性などを考慮する必要がない．

＜欠点＞
－　空白部も計算されるため，その分，計算時間が余計にかかる．
－　特定領域の都合でgridを細かく分割する際には，block全体にその分割を適用することになるため，実用的な精度で結果を得るためには大幅にcell数が増大する可能性あり．

＜余談＞1000万セル超への挑戦

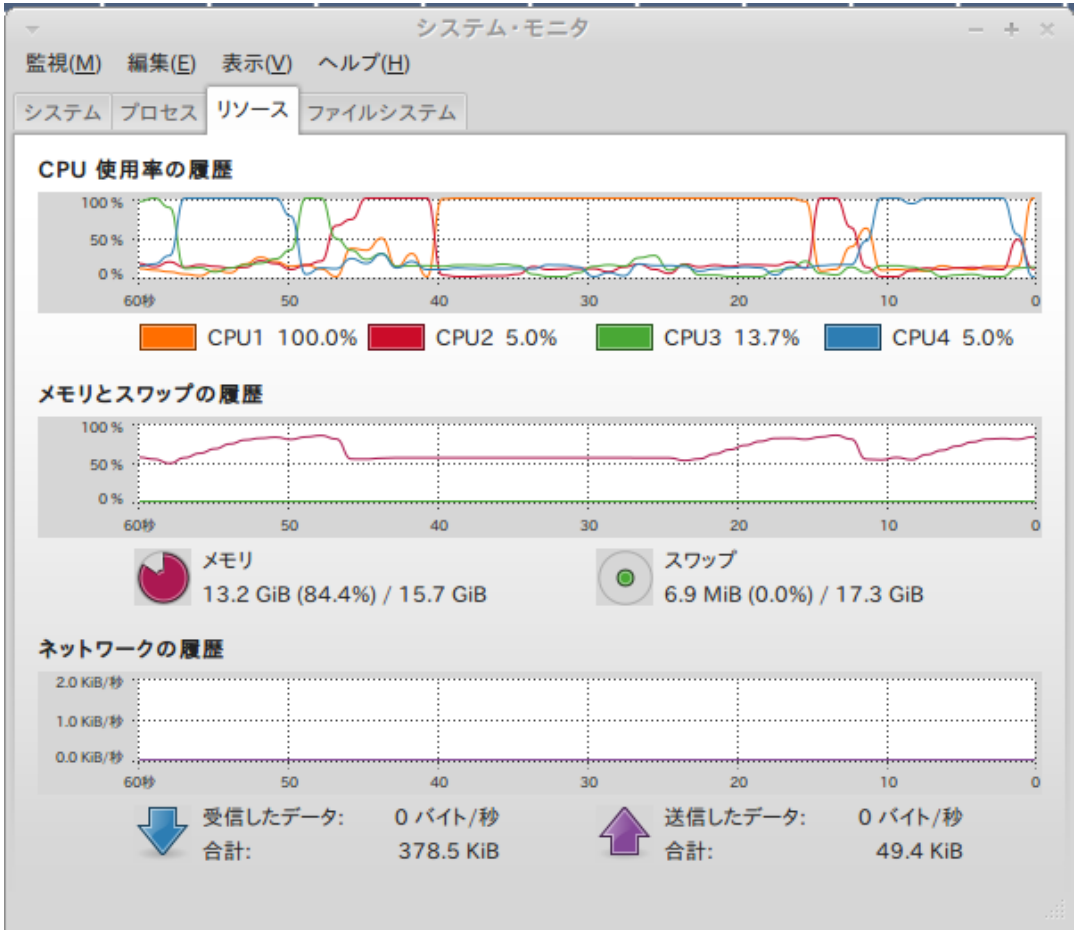
blockMeshを実行する．しかし，何時間経っても終わらない…

そうだ，リソースを確認しよう･･･

メインメモリが不足しており，ディスクスワップを繰り返していることが判明．
（このとき，CPUも数%しか使用されず，処理は明らかに進んでいない．）
このとき，メインメモリは4GB搭載に対し，

メインメモリサイズ（4GB）
＋ スワップされているデータ量が6GB強

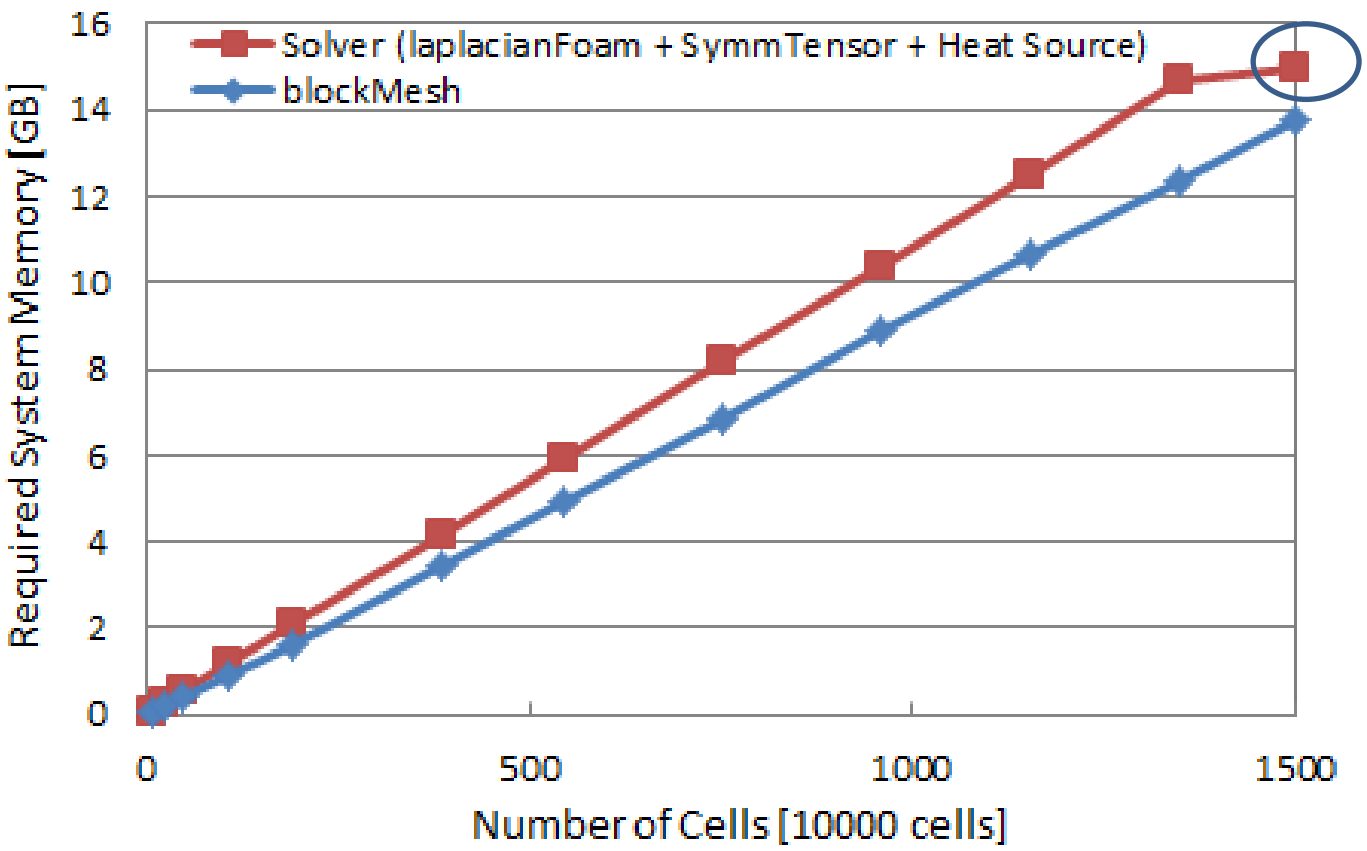
よって，10GB程度は必要ということになる…



そこで，秋葉原にて（2013年3月現在）1万円で買える16GB（8GB x 2）のDDR3-12800を購入．

→ blockMeshは数分で完了（メモリ使用量は10GB強）．その後，setFieldsも数分で終わる．
肝心のソルバに関しては，ピークで13GB強メモリを使用することを確認．

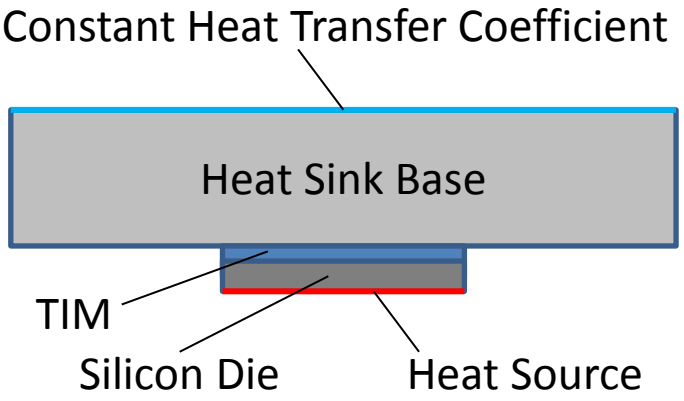
スワップなし実行に必要なメモリ容量



実行の一部で
スワップ発生

blockMeshのみであれば16GBのメインメモリで1500万セルまでO.K.
blockMesh/setFields実行後にソルバー（前回資料のソルバB，laplacianFoamにソース項を追加し，温度伝導率にSymmTensor（対象テンソル）を用いたソルバ）を実行した場合，1500万セルでは実行の一部でスワップが発生.
実行ソフトウェア環境：DEXCS2012 for OpenFOAM ® 2.1x (64bit)

複数のblockを組み合わせてそれぞれ個別にgridを切る



モデル領域を複数のblockとしてモデル化する.

＜利点＞
- 解きたい部材のみをモデル化できる.
- 空白部がなく，また，各部材ごとにgridを設定できるため，1つのblockでモデル化する場合よりcell数を節約できる.

＜欠点＞
- 依然として，キリの悪い分割が難しく，結果として，細かくgrid分割しなければならないケースが多く存在する.

複数のblockの定義とpatchのマージ

```
vertices
(
    // Silicon Die
    (-5.0 -5.0 0.0)      // 0
    ( 5.0 -5.0 0.0)      // 1
    ( 5.0  5.0 0.0)      // 2
    (-5.0  5.0 0.0)      // 3
                :
    // TIM Top
    (-5.0 -5.0 zzz2)     // 8
    ( 5.0 -5.0 zzz2)     // 9
    ( 5.0  5.0 zzz2)     // 10
    (-5.0  5.0 zzz2)     // 11

    // Heatsink Base
    (-25 -25 zzz3)       // 12
    ( 25 -25 zzz3)       // 13
    ( 25  25 zzz3)       // 14
    (-25  25 zzz3)       // 15
                :
);
```

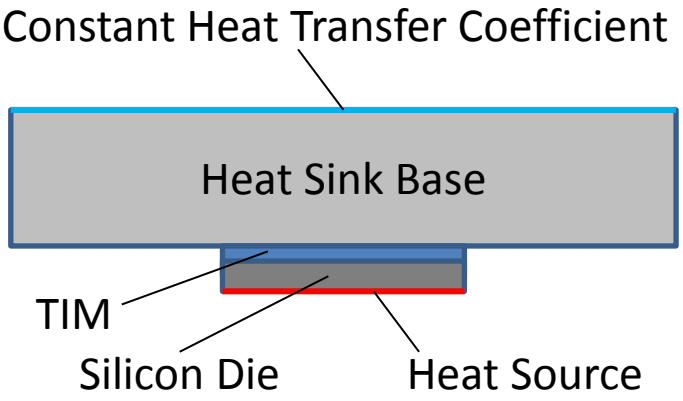
```
blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 8) simpleGrading (1 1 1)
    hex (4 5 6 7 8 9 10 11) (20 20 2) simpleGrading (1 1 1)
    hex (12 13 14 15 16 17 18 19) (50 50 20) simpleGrading (1 1 1)
);
```

```
patches
(
                :
    patch TIMTop
    (
        (8 9 10 11)
    )
    patch HeatSinkBottom
    (
        (12 13 14 15)
    )
                :
);
```

```
mergePatchPairs
(
    (HeatSinkBottom TIMTop)
);
```

0フォルダのT, ST, DTに
HeatSinkBottomの設定を記述

複数のblockを組み合わせてそれぞれ個別にgridを切る



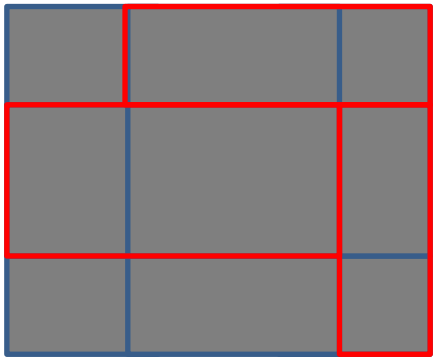
モデル領域を複数のblockとしてモデル化する.

＜利点＞
- 解きたい部材のみをモデル化できる.
- 空白部がなく，また，各部材ごとにgridを設定できるため，1つのblockでモデル化する場合よりcell数を節約できる.

＜欠点＞
- 依然として，キリの悪い分割が難しく，結果として，細かくgrid分割しなければならないケースが多く存在する.

熱源等に合ったセル分割を行いたい場合には，xy方向を複数のblockで表現することで対応できる.

＜利点＞ 効率の良いセル分割が可能
＜欠点＞ 複雑な構成をblockMeshDictに記述する必要あり.
→ 自前でツールを作ればそれほど手間ではなくなる！？



snappyHexMeshを活用する

今回の勉強会に期待！！