
# GPUコンピューティング No.10

# 応用事例　拡散方程式

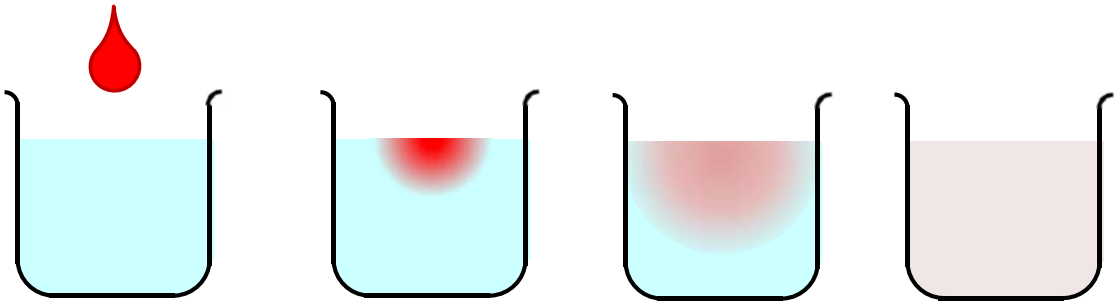
*東京工業大学　　学術国際情報センター*

*青木　尊之*

## 拡散現象

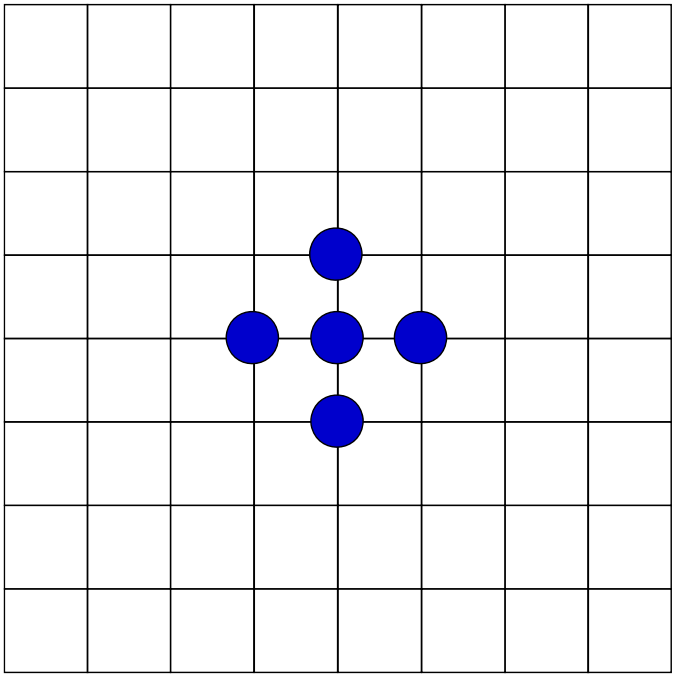

コップの中の水に赤インクを落す



次第に拡散して赤インクは拡がって行き、最後は均一な色になる

# 平均操作(1)



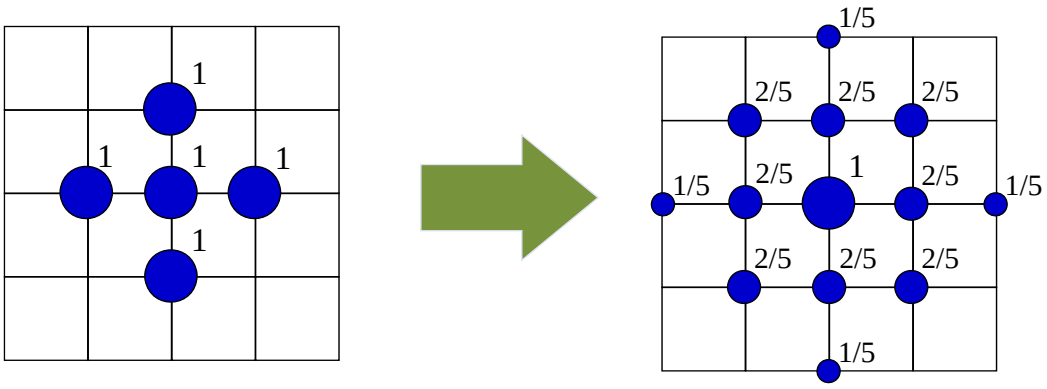
自分自身の値と上下左右で隣接する点の値を足して5で割り、新しい値とする。

最初の状態

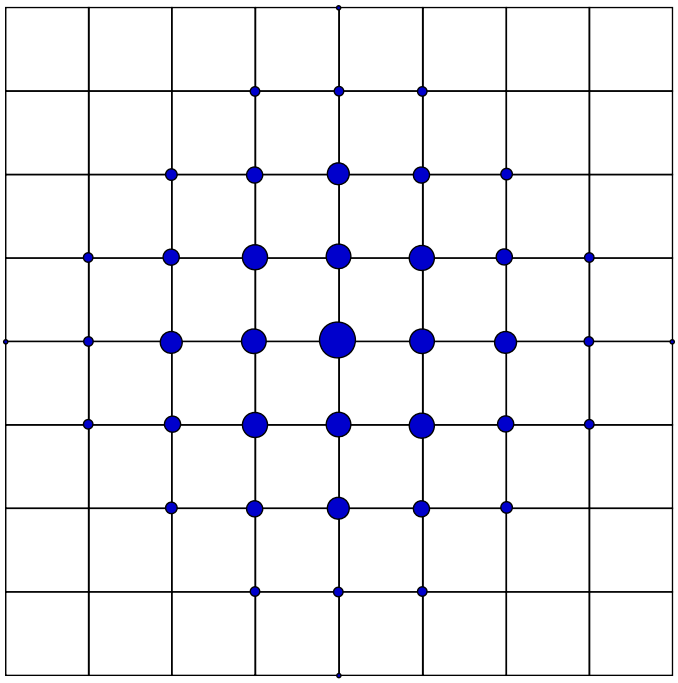
# 平均操作(2)

自分自身の値と上下左右で隣接する点の値を足して5で割り、新しい値とする。



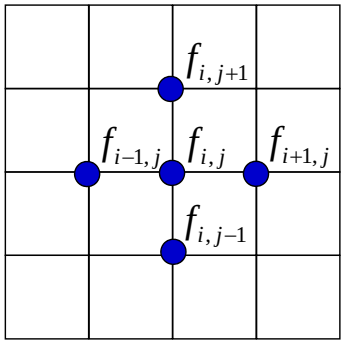

 

# 平均操作(3)





# 平均操作(4)




自分自身の値と上下左右で隣接する点の値を足して5で割り、新しい値とする。

***i*** : $x$ 方向の格子点番号

***j*** : $y$ 方向の格子点番号

平均操作の手続き

$$f^*_{i,j} = \frac{f_{i,j} + f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1}}{5}$$

# 平均操作(5)


自分自身の値と上下左右で隣接する点の値を足して5で割り、新しい値とする。

$$f_{i,j}^{n+1} = \frac{f_{i,j}^{n} + f_{i+1,j}^{n} + f_{i-1,j}^{n} + f_{i,j+1}^{n} + f_{i,j-1}^{n}}{5}$$

$n$ : 現在の値
$n+1$ : 平均操作後の値

両辺から $f_{i,j}^{n}$ を引く

$$f_{i,j}^{n+1} - f_{i,j}^{n} = \frac{f_{i+1,j}^{n} - 2f_{i,j}^{n} + f_{i-1,j}^{n} + f_{i,j+1}^{n} - 2f_{i,j}^{n} + f_{i,j-1}^{n}}{5}$$

# 拡散方程式へ帰着


$\Delta t = 1.0,\ \Delta x = \Delta y = 1.0,\ \kappa = 1/5$ とする

$$\underbrace{\frac{f_{i,j}^{n+1} - f_{i,j}^{n}}{\Delta t}} = \kappa \underbrace{\frac{f_{i+1,j}^{n} - 2f_{i,j}^{n} + f_{i-1,j}^{n}}{\Delta x^2}} + \kappa \underbrace{\frac{f_{i,j+1}^{n} - 2f_{i,j}^{n} + f_{i,j-1}^{n}}{\Delta y^2}}$$

有限差分法の $\frac{\partial f}{\partial t}$　　有限差分法の $\frac{\partial^2 f}{\partial x^2}$　　有限差分法の $\frac{\partial^2 f}{\partial y^2}$

2次元拡散方程式

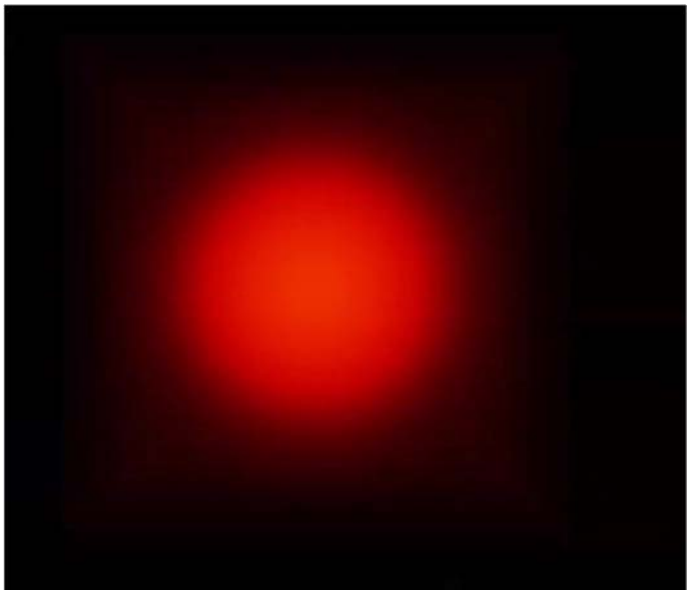
$$\frac{\partial f}{\partial t} = \kappa\left(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\right) = \kappa \nabla^2 f$$

# 3次元拡散方程式


$$\frac{\partial f}{\partial t} = \Delta \cdot \kappa \nabla f$$
$$= \kappa \nabla^2 f$$
$$= \kappa \left( \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \right)$$

κ：拡散係数（物理定数）



典型的なステンシル計算



# 差分法による離散化


四則演算による偏微分方程式の近似計算を可能にする

$$\frac{\partial f}{\partial t} \rightarrow \frac{f_{i,j,k}^{n+1} - f_{i,j,k}^{n}}{\Delta t}$$ （時間方向の1次精度前進差分）

$$\frac{\partial^2 f}{\partial x^2} \rightarrow \frac{f_{i+1,j,k}^{n} - 2 f_{i,j,k}^{n} + f_{i-1,j,k}^{n}}{\Delta x^2}$$ （$x$方向の2次精度中心差分）

$$\frac{\partial^2 f}{\partial y^2} \rightarrow \frac{f_{i,j+1,k}^{n} - 2 f_{i,j,k}^{n} + f_{i,j-1,k}^{n}}{\Delta y^2}$$ （$y$方向の2次精度中心差分）

$$\frac{\partial^2 f}{\partial z^2} \rightarrow \frac{f_{i,j,k+1}^{n} - 2 f_{i,j,k}^{n} + f_{i,j,k-1}^{n}}{\Delta z^2}$$ （$z$方向の2次精度中心差分）

# 離散化式


$$f_{i,j,k}^{n+1} = f_{i,j,k}^{n} + \kappa\Delta t\left( \frac{f_{i+1,j,k}^{n} - 2f_{i,j,k}^{n} + f_{i-1,j,k}^{n}}{\Delta x^2} + \frac{f_{i,j+1,k}^{n} - 2f_{i,j,k}^{n} + f_{i,j-1,k}^{n}}{\Delta y^2} + \frac{f_{i,j,k+1}^{n} - 2f_{i,j,k}^{n} + f_{i,j,k-1}^{n}}{\Delta z^2} \right)$$

$$= c_0 f_{i,j,k}^{n} + c_1 f_{i+1,j,k}^{n} + c_2 f_{i-1,j,k}^{n} + c_3 f_{i,j+1,k}^{n} + c_4 f_{i,j-1,k}^{n} + c_5 f_{i,j,k+1}^{n} + c_6 f_{i,j,k-1}^{n}$$

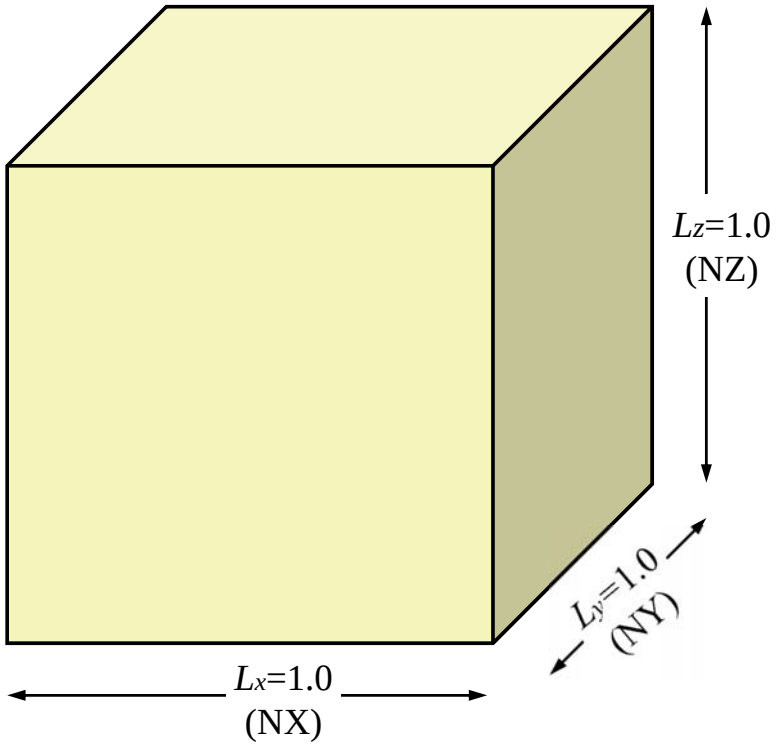
$$c_0 = 1 - \kappa\Delta 2\left( \frac{2}{\Delta x^2} + \frac{2}{\Delta y^2} + \frac{2}{\Delta z^2} \right) \qquad c_1 = c_2 = \frac{\kappa\Delta t}{\Delta x^2} \qquad c_3 = c_4 = \frac{\kappa\Delta t}{\Delta y^2} \qquad c_5 = c_6 = \frac{\kappa\Delta t}{\Delta z^2}$$

**平均操作＝拡散方程式**

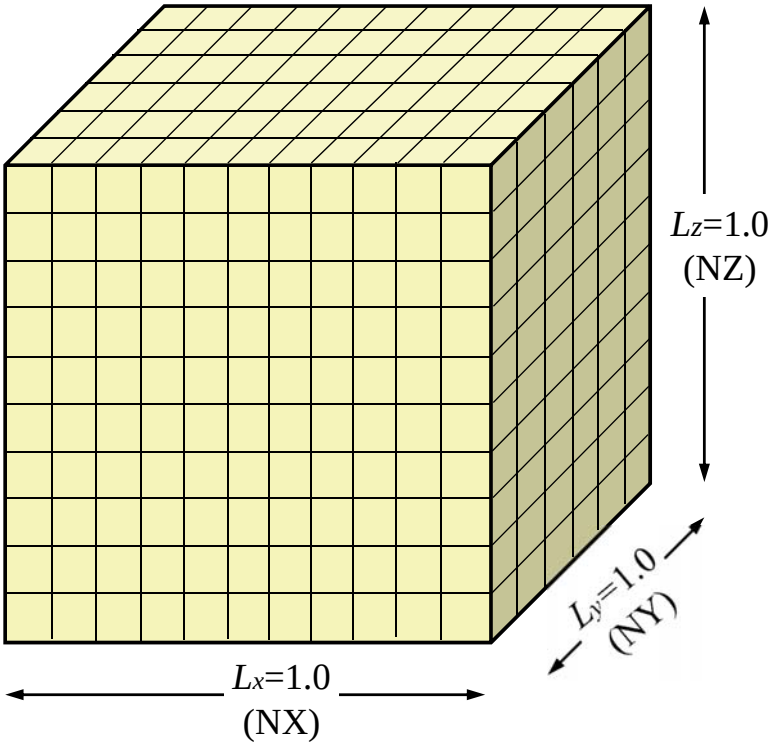
$$c_0 = c_1 = c_2 = c_3 = c_4 = c_5 = c_6 = \frac{1}{7}$$



# 計算領域

# 計算領域




$$\Delta x = \frac{L_x}{NX}$$

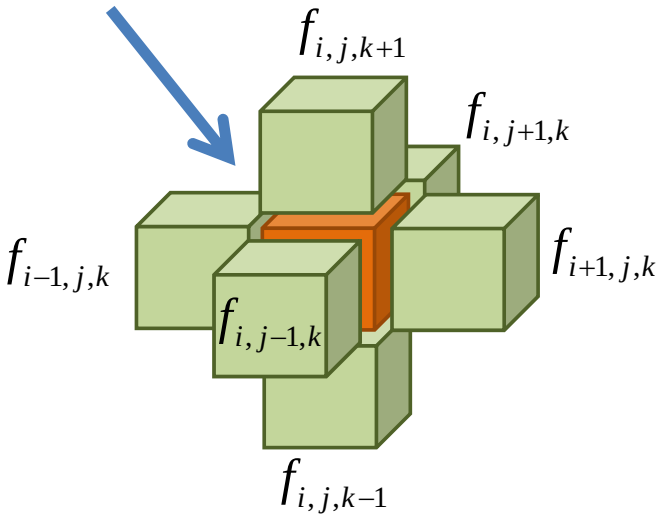
$$\Delta y = \frac{L_y}{NY}$$

$$\Delta z = \frac{L_z}{NZ}$$



# 7点ステンシル

$f_{i,j,k}$




**1次元配列**

$$f_{i,j,k} = f[j]$$

$$f_{i+1,j,k} = f[j+1]$$

$$f_{i-1,j,k} = f[j-1]$$

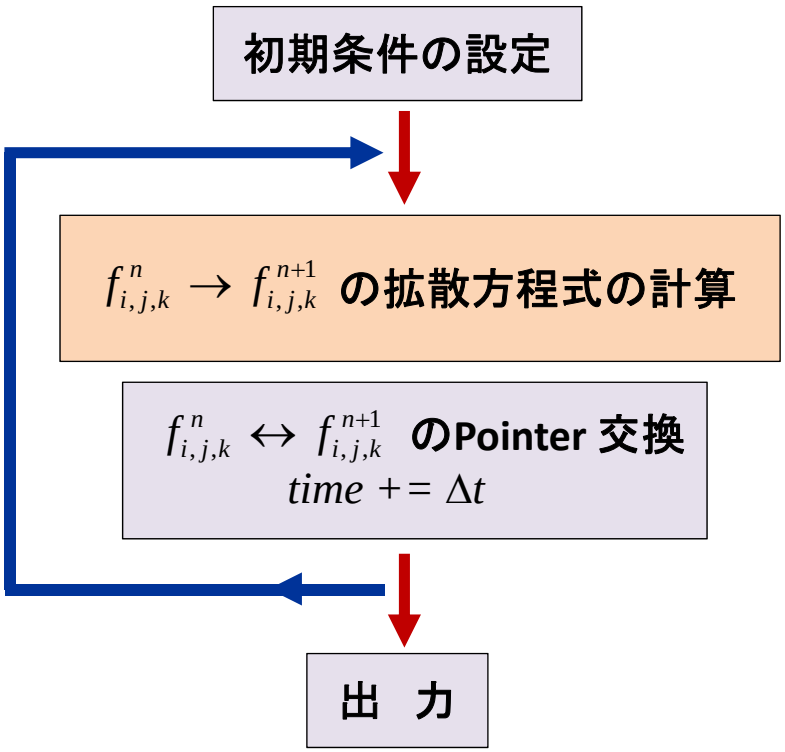
$$f_{i,j+1,k} = f[j+nx]$$

$$f_{i,j-1,k} = f[j-nx]$$

$$f_{i,j,k+1} = f[j+nx*ny]$$

$$f_{i,j,k-1} = f[j-nx*ny]$$

# 時間積分






# 初期条件と境界条件

**初期条件： Cosine Bell プロファイル**

$$f(x,y,z) = (1-\cos(k_x x))(1-\cos(k_y y))(1-\cos(k_z z))$$

**境界条件： ノイマン境界**

$$x = 0,\quad L_x \qquad \frac{\partial f}{\partial x} = 0$$ （$x$ 方向の壁から物質が漏れない条件）

$$y = 0,\quad L_y \qquad \frac{\partial f}{\partial y} = 0$$ （$y$ 方向の壁から物質が漏れない条件）

$$z = 0,\quad L_z \qquad \frac{\partial f}{\partial z} = 0$$ （$z$ 方向の壁から物質が漏れない条件）

# 初期条件のプログラム


```
f_h = (double *) malloc(sizeof(double)*n);

for(jz=0 ; jz < nz; jz++) {
    for(jy=0 ; jy < ny; jy++) {
        for(jx=0 ; jx < nx; jx++) {
            j = nx*ny*jz + nx*jy + jx;
            x = dx*((double)jx + 0.5);
            y = dy*((double)jy + 0.5);
            z = dz*((double)jz + 0.5);

            f_h[j] = 0.125*(1.0 - cos(kx*x))
                          *(1.0 - cos(ky*y))
                          *(1.0 - cos(kz*z));
        }
    }
}

cudaMemcpy(f, f_h, n*sizeof(double), cudaMemcpyHostToDevice);
```

Source Code #18



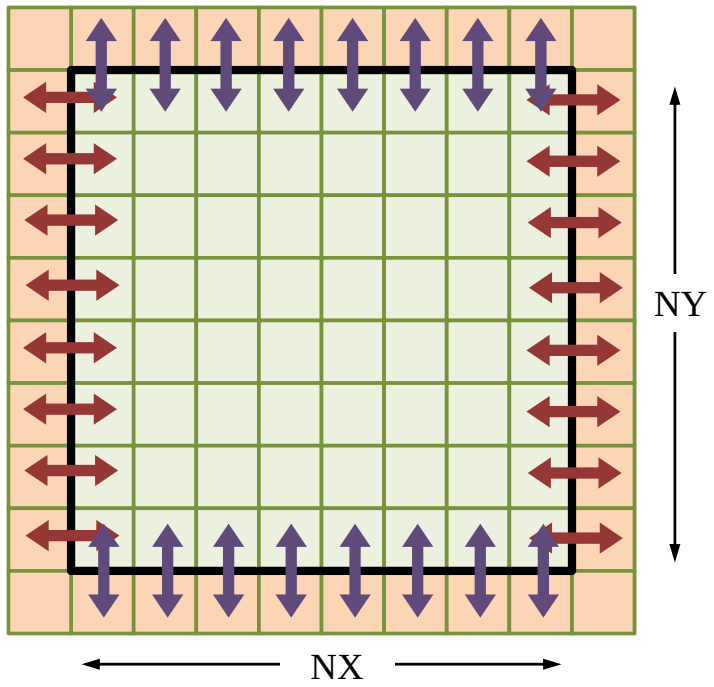
# 数値計算における境界条件


境界の外側の格子点を同じ値とする： ノイマン境界条件

$$x = 0: \quad \frac{f_{0,j,k} - f_{-1,j,k}}{\Delta x} = 0$$

$$y = 0: \quad \frac{f_{i,0,k} - f_{i,-1,k}}{\Delta y} = 0$$

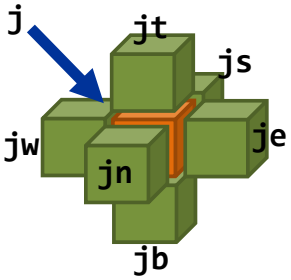
$$z = 0: \quad \frac{f_{i,j,0} - f_{i,j,-1}}{\Delta z} = 0$$



# CPUの拡散方程式プログラム


```
for(jz=0; jz < nz; jz++) {
    for(jy=0; jy < ny; jy++) {
        for(jx=0; jx < nx; jx++) {
            j = nx*ny*jz + nx*jy + jx;
            je = j+1; jw = j-1;  jn = j+nx; js =j-nx;
            jt = j + nx*ny;  jb = j - nx*ny;

            if (jx == nx-1) je = j;
            if (jx == 0   ) jw = j;
            if (jy == ny-1) jn = j;
            if (jy == 0   ) js = j;
            if (jz == nz-1) jt = j;
            if (jz == 0   ) jb = j;

            fn[j] = c0*f[j]
                  + c1*f[je] + c2*f[jw]
                  + c3*f[jn] + c4*f[js]
                  + c5*f[jt] + c6*f[jb];
        }
    }
}
```

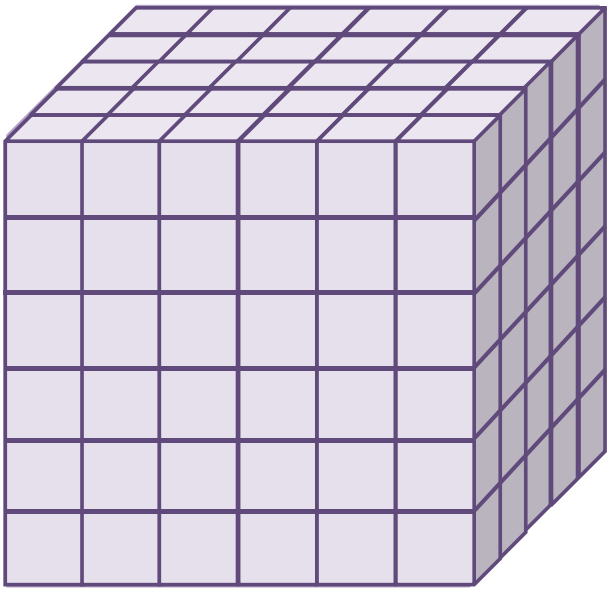
境界条件



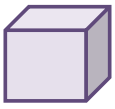

Source Code #18



# block分割（1）

3次元thread配置



# GPUカーネル関数

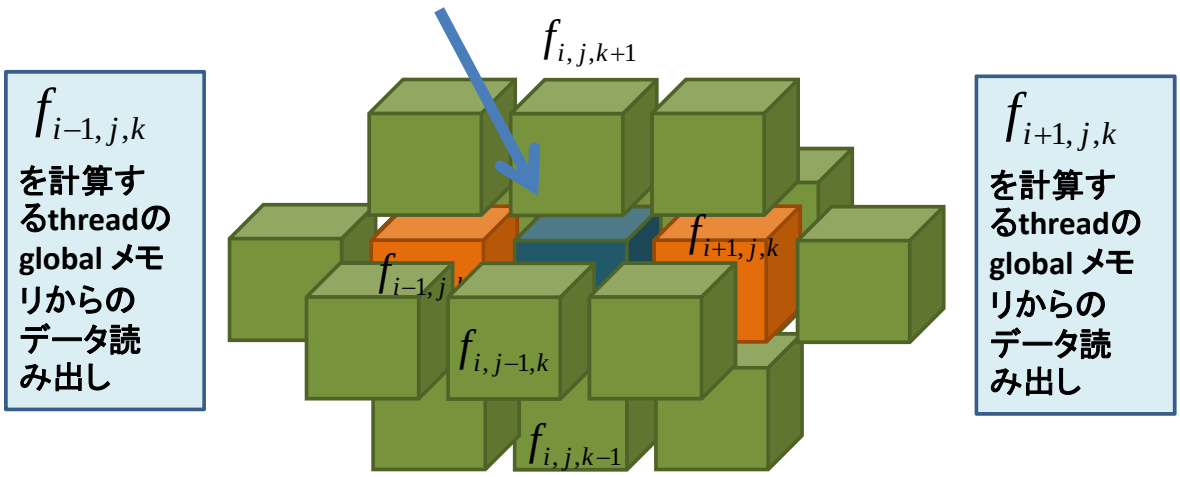

```
__global__  void  gpu_diffusion3d(double *f, double *fn,
int nx, int ny, int nz, double c0, double c1, double c2, double
c3, double c4, double c5, double c6)
{
    int jx = ........ ;  /* x方向のindex計算をする */
    int jy = ........ ;  /* y方向のindex計算をする */
    int jz = ........ ;  /* z方向のindex計算をする */

    int j,  je,  jw,  jn,  js,  jt,  jb;

            /* この部分を完成させる */

}
```

Source Code #18



# Cache によるデータの再利用


$f_{i,j,k}$ を計算するthreadのglobal メモリからのデータ読み出し




$f_{i-1,j,k}$ を計算するthreadのglobal メモリからのデータ読み出し

$f_{i+1,j,k}$ を計算するthreadのglobal メモリからのデータ読み出し

1つのthread がglobal メモリから読込んだdata を隣接threadが利用する

# Fermi L1-cache の効果


■ Default では、自動的に Fermi コアのL1 cache (16kB) が有効になっている。Default の Performance は、58 GFLOPS

■ Compiler Option : –Xptxas –dlcm=cg
--ptxas-options="-dlcm=cg"
で L1 cache を無効にする

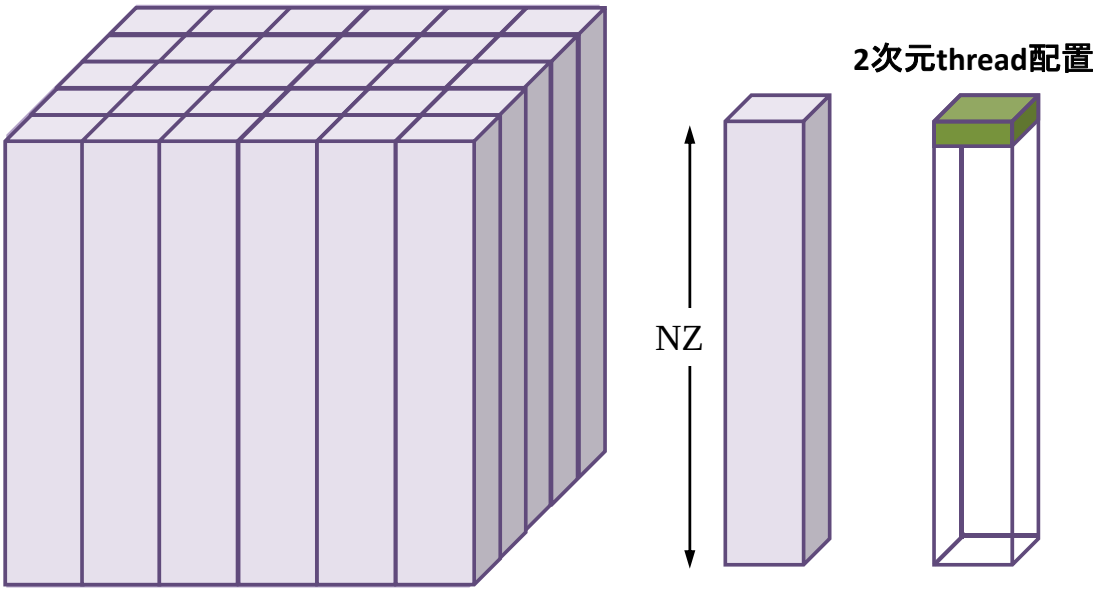
58 GFLOPS 程度 → 62 GFLOPS
（何故か、性能が向上している。L2 cache が効いている。）

■ cudaThreadSetCacheConfig( ***, cudaFuncCachePreferL1 );
Cache 48kB + Shared メモリ 16 kB にする

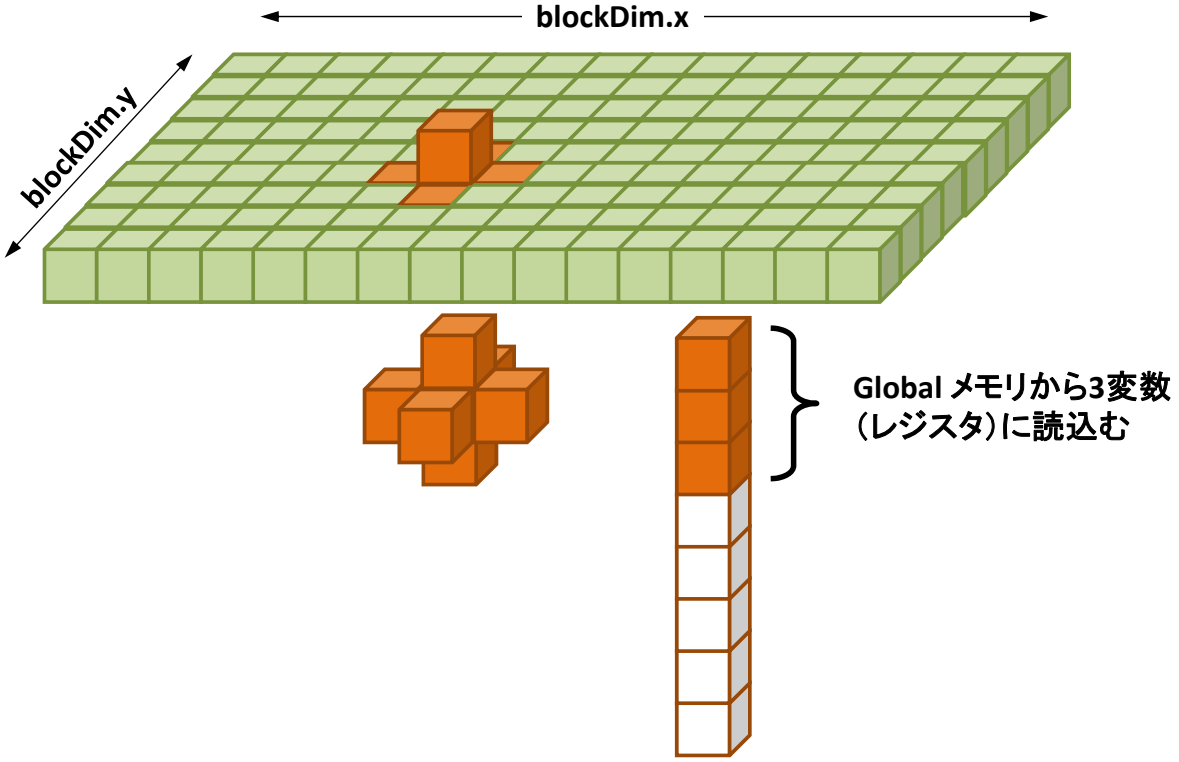
58 GFLOPS → 59 GFLOPS（余り変わらず、この場合は L1 cache は16kB で充分）



# block分割（2）

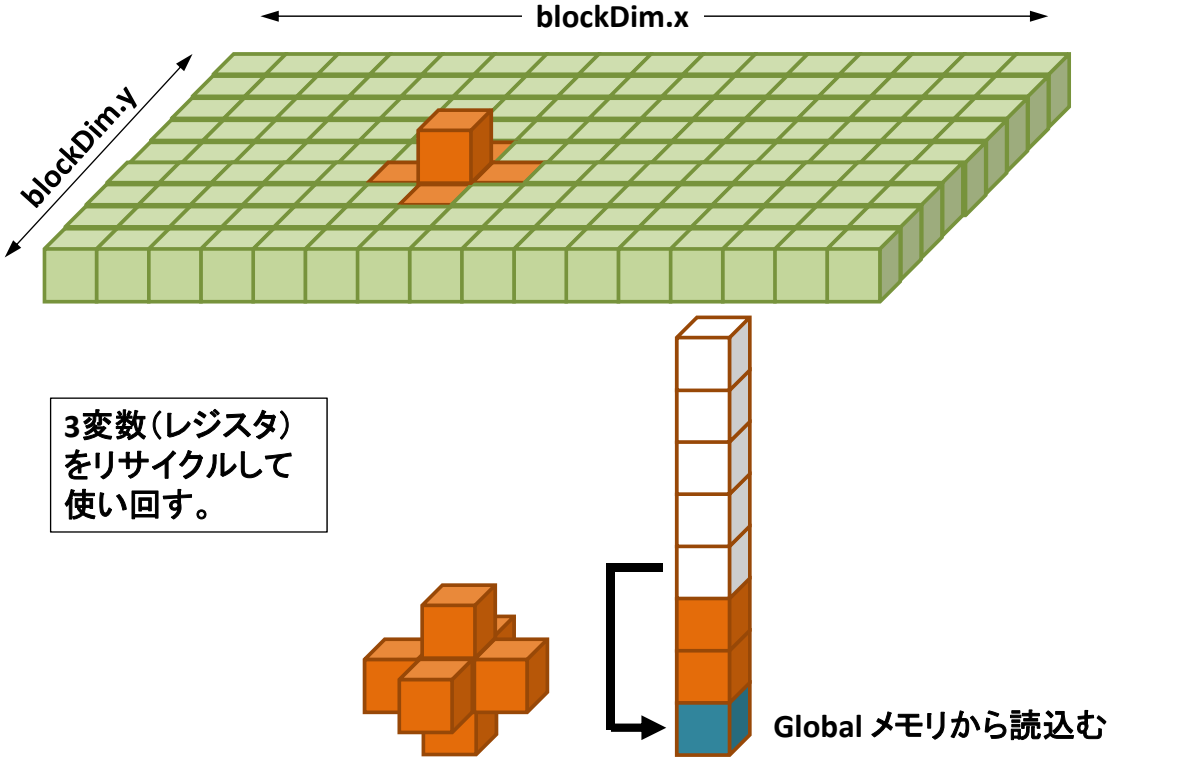



# Registerによるデータの再利用




# Registerによるデータの再利用

# GPUカーネル関数


```
__global__  void  gpu_diffusion3d(double *f, double *fn,
int nx, int ny, int nz, double ce, double cw, double cn, double
cs, double ct, double cb, double cc)
{
    int jx = ........ ;  /* complete the index calculation */
    int jy = ........ ;  /* complete the index calculation */
    int j,  jz,  je,  jw,  jn,  js,  jt,  jb;

    for(jz = 0; jz < nz; jz++) {

      /* make a program to solve the 3-D diffusion equation */

    }
}
```



# レポート課題 5


## 『拡散方程式計算の高速化』

**Source Code #18** の 拡散方程式の1タイムステップの時間積分を計算する GPU カーネル関数を完成させ、また、様々なチューニングを行うことにより、どのように高速化できるかを試みよ。よ。レポートには何をどのして、どのくらい高速化できかかを記述すること。

期限：2013年 7月4日（木） 17:00
場所：学術国際情報センター・国際棟 1F のI7-3メールボックスに提出。または Subject:「拡散方程式計算の高速化」とし、電子メールの宛先：gpu_report2013@sim.gsic.titech.ac.jp
に上記の内容を pdf ファイルとして提出すること。