反応拡散方程式の数値計算並列処理

- 陽的差分法を用いた反応拡散方程式の数値解
  - 1次元の場合、空間の分割数 N に比例した計算量
  - 2次元の場合、N^2 に比例した計算量
  - 空間を分割して複数ノードを用いた並列処理は効果的か？

プログラムの骨格（1次元）

```
double u[SIZE+2], v[SIZE+2];

main()
{
  initialize();  // 初期条件の設定、ファイルのオープンなど
  process();     // 処理
  finish();      // ファイルのクローズなど、後始末
}

void process()
{
  int step;

  for(step=0; step<STEP; step++){
    if( step%INTV == 0 ) write_data();
    go_ahead(); // 時刻を dt 進める関数呼び出し
  }
}
```


処理を関数毎にまとめる

```
void go_ahead()
{
  double ud[SIZE+2], vd[SIZE+2];
  int i;

  /* 陽的差分法 */
  for(i=1; i<SIEZ+1; i++){
    ud[i]=u[i]+cu*(u[i-1]-2*u[i]+u[i+1])+fn(u[i], v[i])*DT;
    vd[i]=v[i]+cv*(v[i-1]-2*v[i]+v[i+1])+gn(u[i], v[i])*DT;
  }

  /* 境界条件の設定 */
  ud[0] = ud[1]; ud[SIZE+1] = ud[SIZE];
  vd[0] = vd[1]; vd[SIZE+1] = vd[SIZE];

  // ud[], vd[] と u[], v[] の入れ替え（無駄な処理）
  // ud, vd, u, v を入れ替えて計算。
  // go_aheadを呼び出すと時間は 2*DT だけ進む
}
```


1次元反応拡散方程式

$$\frac{\partial u}{\partial t} = D_u \frac{\partial^2 u}{\partial x^2} + f(u,v)$$

$$\frac{\partial v}{\partial t} = D_v \frac{\partial^2 v}{\partial x^2} + g(u,v)$$

Schnakenberg kinetics

$$f(u,v) = k_1 - k_2 u + k_3 u^2 v$$

$$g(u,v) = k_4 - k_3 u^2 v$$

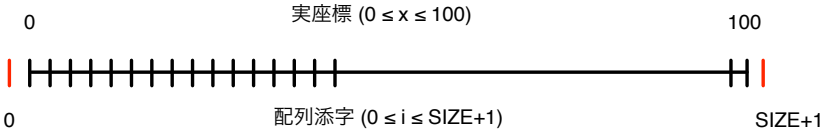
区間 0 ≤ x ≤ 100 を

1）N = 1,001 (DX = 0.1)で分割して 1ノードで計算

2）2ノードを用いて（N = 501, DX = 0.1）並列計算

陽的差分法

1ノードで実行する場合（これまでのやり方）

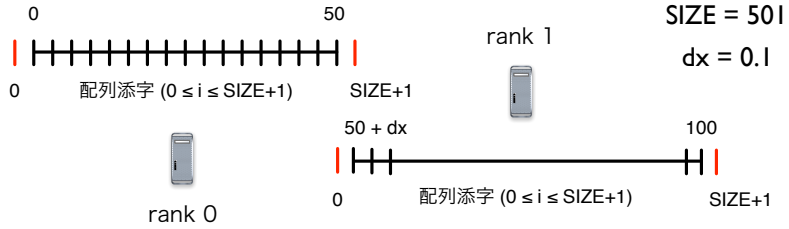



```
double u[SIZE+2], v[SIZE+2];      配列の添字 0, SIZE+1 は境界条件用のダミー

void go_ahead() /* 時刻を dt だけ進める */          SIZE = 1001
{                                                    DX = 0.1
  for(i=1; i<SIZE+1; i++){
    ud[i] = u[i] + ..... // 領域内部状態の更新
  }
  ud[0]=ud[1]; ... 境界条件の設定
}
```

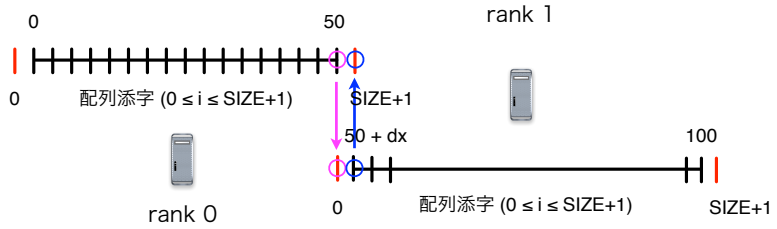

2ノードによる並列処理

2ノードで実行する場合（空間を2つに分割）



2つに分割した空間をうまく接続する必要がある
（rank 0 と 1 の間の通信が必要）

2ノードによる並列処理実装




```
double u[SIZE+2], v[SIZE+2];

void go_ahead(int rank)
{
  MPI により2つに分割した空間を接続する（send/receive）
  for(i=1; i<SIZE+1; i++){
    ud[i] = u[i] + .....
  }
  ud[0]=ud[1]; ...
}
```


2ノードによる並列処理実装

- 時刻を dt 進める関数 process(int rank) を定義
- rank 0 は左半分の区画、rank 1は右半分の区画を担当

```
void go_ahead(int rank)
{

  if( rank == 0 ) {
    1)MPI により右端の境界条件 u[SIZE+1], v[SIZE+1] を取得
    2)陽的差分 + 左端の境界条件設定
  }
  if( rank == 1 ){
    1)MPI により左端の境界条件 u[0], v[0] を取得
    2)陽的差分 + 右端の境界条件設定
  }
}
```

2ノードによる並列処理実装

- それぞれのノードの計算結果を、個別のファイル（ファイル名は異なる！）に書き出す。座標に注意！
- Mathematica で2つのファイルを連結して図示（別資料を参照）。

```
void write_data(int rank)
{
   int i;
   if( rank == 0){
          for(i=1; i<=SIZE; i+=INTV_THIN){
               fprintf(fp_u, "%f %f ", (i-1)*DX, u[i]);
               fprintf(fp_v, "%f %f ", (i-1)*DX, v[i]);
          }
          fprintf(fp_u, "\n"); fprintf(fp_v, "\n");

   }
   if( rank == 1 ){
          for(i=1; i<=SIZE; i+=INTV_THIN){
               fprintf(fp_u, "%f %f ", SIZE*DX + (i-1)*DX, u[i]);
               fprintf(fp_v, "%f %f ", SIZE*DX + (i-1)*DX, v[i]);
          }
          fprintf(fp_u, "\n"); fprintf(fp_v, "\n");
   }
}
```


2ノードによる並列処理実装

- 2つのノードが異なる名前のファイルを作成

```
FILE *fp_u, *fp_v;

....

void initialize(int rank)
{

   // ファイルを開く
   if( rank == 0 ){
          fp_u = fopen("data-u-0", "w");
          fp_v = fopen("data-v-0", "w");
   }

   if( rank == 1){
          fp_u = fopen("data-u-1", "w");
          fp_v = fopen("data-v-1", "w");
   }

   // 初期条件の設定など
   ...
}
```


並列処理の効果

- 時刻を dt 進めるために必要な計算量は分割数 N に比例
- ノード間通信に要する時間 Tc（通信には時間がかかる）
- 陽的差分の処理時間＋通信時間が実時間

1ノード処理	2ノード処理
T ∝ N	T ∝ N/2 + Tc

分割数 N が大きいほど 2ノード並列処理が効果的

課題

- Schnakenberg kinetics による1次元反応拡散方程式を 2 つのノードを用いて並列計算するプログラムを作成せよ。
- 処理速度（実行時間）を測定し、並列化の効果（2ノードを用いると2倍速くなるのか？）について調べよ。分割数 N を変化させ得ると並列化の効果はどうなるか？
- パラメータは次の値を用いること。

$$k_1 = 0.2, k_2 = 1.0, k_3 = 1.0, k_4 = 0.5$$
$$D_u = 1.0, D_v = 14.0$$
$$0 \leq x \leq 100$$