

	授業計画	課題
04/06	第1回 微分方程式の離散化	前進・後退・中心差分，高次の差分を用いて微分方程式を離散化し，誤差を評価できる
04/10	第2回 有限差分法	時間積分の安定性や高次精度の積分を理解し移流・拡散・波動方程式を解析できる
04/13	第3回 有限要素法	Galerkin 法，テスト関数，isoparametric 要素の概念を理解し，弾性方程式を解析できる
04/17	第4回 <del>スペクトル法</del> Python ctypes	Fourier・Chebyshev・Legendre・Bessel などの直交基底関数による離散化の利点を説明できる
04/20	第5回 <del>境界要素法</del> OpenMP	逆行列と $\delta$ 関数・Green 関数の関係を理解し境界積分方程式を用いた解析ができる
04/24	第6回 <del>分子動力学法</del> MPI	時間積分の symplectic 性や熱浴の概念を理解し分子間に働く保存力の動力学を解析できる
04/27	第7回 <del>Smooth particle hydrodynamics (SPH) 法</del> SIMD	微分演算子の動径基底関数による離散化とその保存性・散逸性を評価できる
05/01	第8回 <del>Particle mesh 法</del> GPU	粒子と格子の両方の離散化を組み合わせる場合の離散点からの補間法と高次モーメントの保存法

並列プログラミング言語: SIMD, OpenMP, MPI, GPU

並列計算ライブラリ: BLAS, LAPACK, FFTW

高性能計算支援ツール: Compiler flags, Profiler, Debugger

TSUBAME job submission

# MPIのインストール

Ubuntu/Debian

```
>sudo apt-get install mpich
```

CentOS/Fedora

```
>sudo yum install mpich
```

Mac OSX

```
>sudo port install mpich
```

```
>sudo brew install mpich
```

Windows

?

## step01.cpp

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    printf("rank: %d/%d\n", mpirank, mpisize);
    MPI_Finalize();
}
```

```
> mpicxx step01.cpp
> mpirun -np 2 ./a.out
```

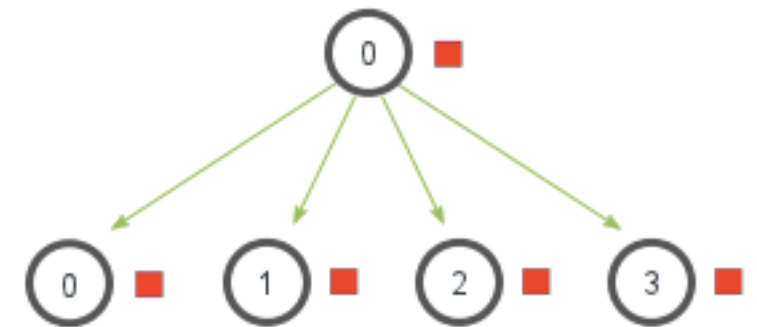
## step02.cpp: MPI\_Bcast

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int data[4] = {0,0,0,0};
    if(!mpirank) {
        for(int i=0; i<4; i++) data[i] = i+1;
    }
    printf("rank%d: before [%d %d %d %d]\n",
        mpirank,data[0],data[1],data[2],data[3]);
    MPI_Bcast(data, 4, MPI_INT, 0, MPI_COMM_WORLD);
    printf("rank%d: after  [%d %d %d %d]\n",
        mpirank,data[0],data[1],data[2],data[3]);
    MPI_Finalize();
}
```

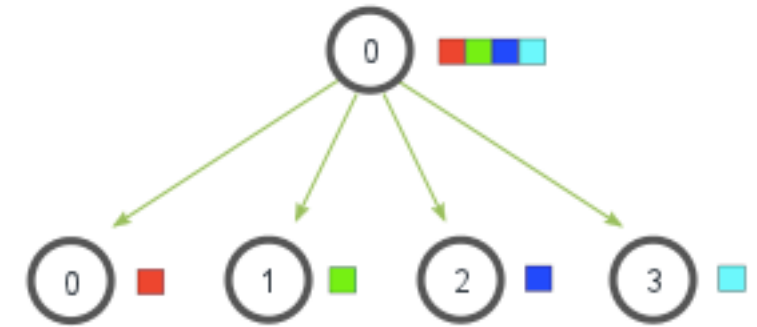
# step03.cpp: MPI\_Scatter

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
    if(!mpirank) {
        for(int i=0; i<4; i++) send[i] = i+1;
    }
    MPI_Scatter(send, 1, MPI_INT, recv, 1, MPI_INT, 0, MPI_COMM_WORLD);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n", mpirank,
        send[0], send[1], send[2], send[3],
        recv[0], recv[1], recv[2], recv[3]);
    MPI_Finalize();
}
```

MPI\_Bcast

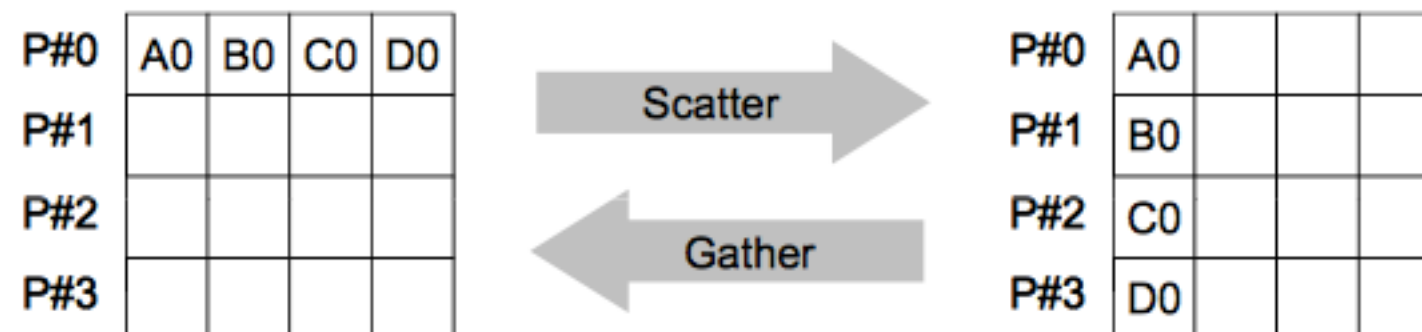


MPI\_Scatter



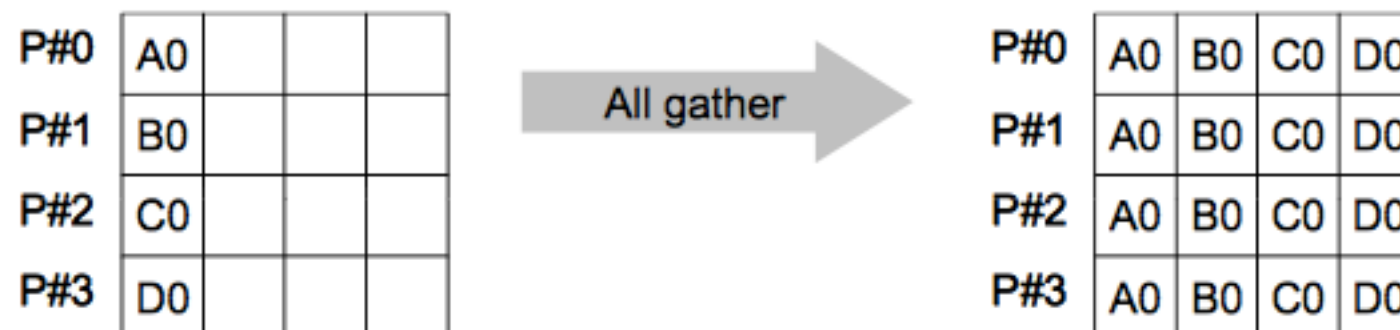
# step04.cpp: MPI\_Gather

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
    send[0] = mpirank+1;
    MPI_Gather(send, 1, MPI_INT, recv, 1, MPI_INT, 0, MPI_COMM_WORLD);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n", mpirank,
        send[0], send[1], send[2], send[3],
        recv[0], recv[1], recv[2], recv[3]);
    MPI_Finalize();
}
```



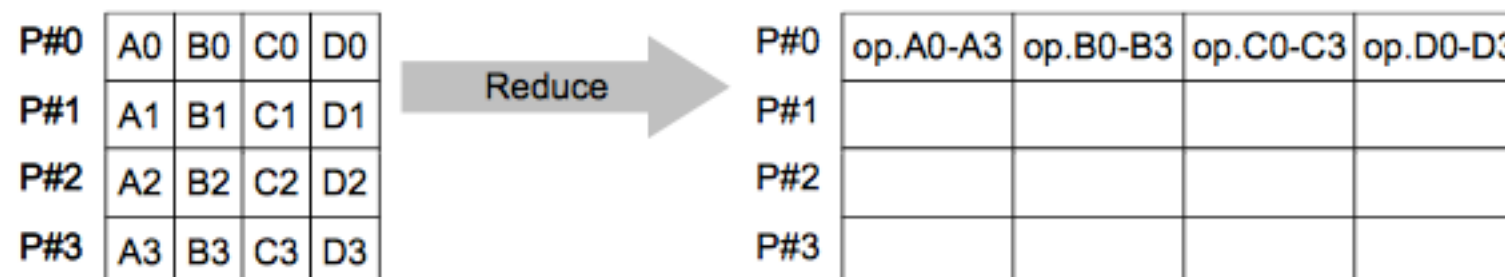
## step05.cpp: MPI\_Allgather

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
    send[0] = mpirank+1;
    MPI_Allgather(send, 1, MPI_INT, recv, 1, MPI_INT, MPI_COMM_WORLD);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n", mpirank,
        send[0], send[1], send[2], send[3],
        recv[0], recv[1], recv[2], recv[3]);
    MPI_Finalize();
}
```



# step06.cpp: MPI\_Reduce

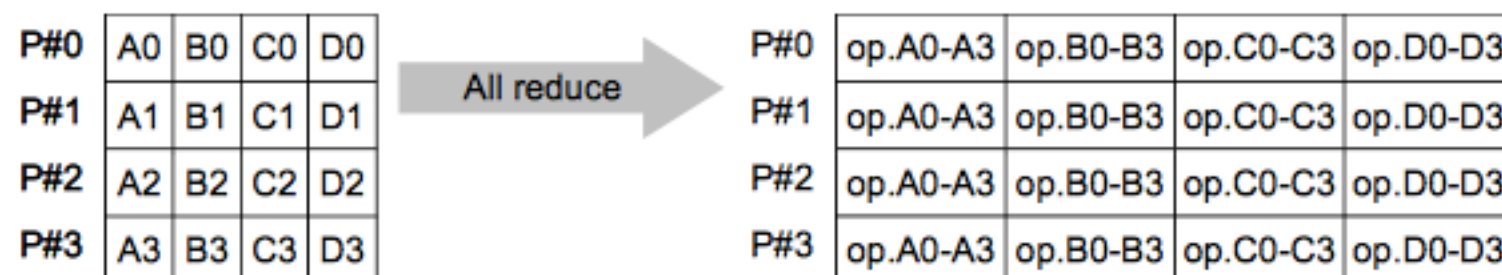
```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4], recv[4] = {0};
    for (int i=0; i<4; i++) send[i] = mpirank + i;
    MPI_Reduce(send, recv, 4, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n",
        mpirank, send[0], send[1], send[2], send[3],
        recv[0], recv[1], recv[2], recv[3]);
    MPI_Finalize();
}
```





# step07.cpp: MPI\_Allreduce

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4], recv[4] = {0};
    for (int i=0; i<4; i++) send[i] = mpirank + i;
    MPI_Allreduce(send, recv, 4, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n",
        mpirank, send[0], send[1], send[2], send[3],
        recv[0], recv[1], recv[2], recv[3]);
    MPI_Finalize();
}
```



# step08.cpp: MPI\_Alltoall

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
    for(int i=0; i<4; i++)
        send[i] = mpirank+10*i;
    MPI_Alltoall(send, 1, MPI_INT, recv, 1, MPI_INT, MPI_COMM_WORLD);
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n", mpirank,
        send[0], send[1], send[2], send[3],
        recv[0], recv[1], recv[2], recv[3]);
    MPI_Finalize();
}
```



# step09.cpp: MPI\_Send, MPI\_Recv

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
    for(int i=0; i<4; i++)
        send[i] = mpirank+10*i;
    if(mpirank==0) {
        MPI_Send(send, 4, MPI_INT, 1, 0, MPI_COMM_WORLD);
    } else if(mpirank==1) {
        MPI_Recv(recv, 4, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n",mpirank,
        send[0],send[1],send[2],send[3],recv[0],recv[1],recv[2],recv[3]);
    MPI_Finalize();
}
```

# step10.cpp: MPI\_Send, MPI\_Recv

```
#include "mpi.h"
#include <stdio>
int main(int argc, char ** argv) {
    MPI_Init(&argc, &argv);
    int mpisize, mpirank;
    MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
    MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
    int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
    for(int i=0; i<4; i++)
        send[i] = mpirank+10*i;
    if(mpirank==0) {
        MPI_Send(send, 4, MPI_INT, 1, 0, MPI_COMM_WORLD);
        MPI_Recv(recv, 4, MPI_INT, 1, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    } else if(mpirank==1) {
        MPI_Send(send, 4, MPI_INT, 0, 1, MPI_COMM_WORLD);
        MPI_Recv(recv, 4, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n",mpirank,
        send[0],send[1],send[2],send[3],recv[0],recv[1],recv[2],recv[3]);
    MPI_Finalize();
}
```