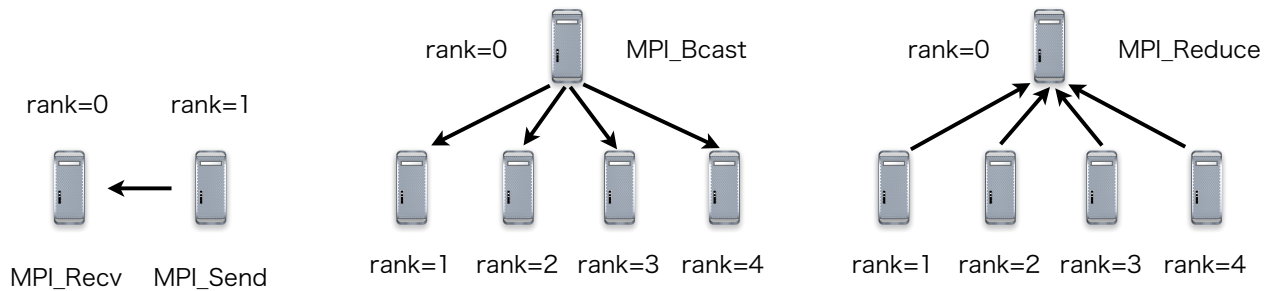


プロセッサ間通信

- ・ 1対1通信を行う関数 MPI_Send と MPI_Recv
- ・ 複数ノードへの一斉送信・受信を行う MPI_Bcast
- ・ 複数ノードのデータのあるノードへ集約する MPI_Reduce



ノード数が増えると、1対1通信は
処理が面倒になる

MPI_Bcast

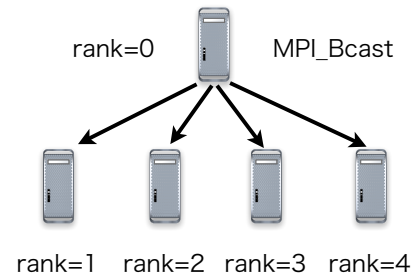
ブロードキャスト通信関数 MPI_Bcast

データ受信

```
MPI_Bcast(  
    void *buf,                受信データが格納されるアドレス  
    int count,                受信するデータの数  
    MPI_Datatype datatype,    受信するデータの型 MPI_INT, MPI_DOUBLE  
    int source,               送信元プロセッサ rank  
    MPI_Comm comm             送受信グループ MPI_COMM_WORLD  
)
```

MPI_Bcast 例

rank0 から他のノードへ
整数値 int data を送る



```
if (rank == 0){
    printf("I am rank %d. Enter an integer: ", rank);    rank0 で整数値を入力
    scanf("%d", &data);
}
```

```
source = 0;    ノード 0 から他のノードへブロードキャスト
MPI_Bcast(&data, 1, MPI_INT, source, MPI_COMM_WORLD);    他のノードでは受信が行われることに注意
```

```
if(rank == 0)
    printf("Rank %d has broadcasted an integer %d\n", rank, data);
else
    printf("Rank %d has received an integer %d\n", rank, data);
```

MPI_Reduce

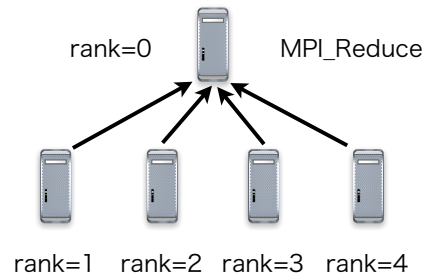
集約関数 MPI_Reduce

データ受信

```
MPI_Reduce(
    void *buf_local,    受信するデータが格納されるアドレス
    void *buf_result,    集約結果を格納するアドレス
    int count,    受信するデータの数
    MPI_Datatype datatype,    受信するデータの型 MPI_INT, MPI_DOUBLE
    MPI_Op operation,    集約処理の指定
    int destination,    送信先プロセッサ rank
    MPI_Comm comm    送受信グループ MPI_COMM_WORLD
)
```

Reduce operation

rank0 へ各ノードの結果を集約する例



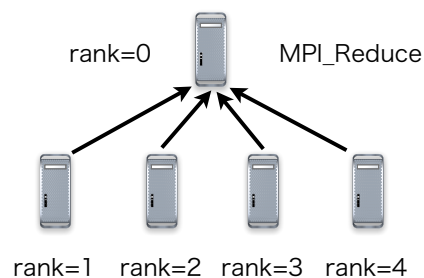
各ノードのデータをどう集約するかを集約 operation で指定

オペレータ	操作
MPI_SUM	和
MPI_PROD	積
MPI_MAX	最大値
MPI_MIN	最小値

MPI_SUM を指定すると、各ノード（自分自身を含む）のデータを足し合わせた結果が送信先の buf_result に格納される。

MPI_Reduce 例

ノード数 5 の場合



各ノードの変数 data に自分のランクを格納し、これをランク 0 の変数 data_reduced に集約処理 MPI_SUM（合計を計算）で格納するプログラム

```
data = rank;
printf("Rank %d has an integer %d\n", rank, data);

dest = 0;
MPI_Reduce(&data, &data_reduced, 1, MPI_INT, MPI_SUM, dest, MPI_COMM_WORLD);

if( rank == dest )
    printf("Reduced data is %d at rank %d\n", data_reduced, rank);
```

処理時間の測定

- 並列処理をする場合、ノード間の通信コスト（ネットワークを介したデータのやり取りにかかる時間）が無視できない
- 処理時間を測定して並列処理の効果を見積もる必要がある。
MPI_Wtime()

```
double start_time, finish_time;
```

```
start_time = MPI_Wtime();  開始時刻を測定
```

```
.....          .....   で挟まれた箇所の処理にかかる時間を測定したい。
```

```
finish_time = MPI_Wtime();  終了時刻を測定
```

```
if( rank == 0 )  
    printf("It took %lf second.\n", finish_time - start_time);
```

Rank 0 のノードが処理時間（終了時刻 - 開始時刻）を表示

課題

- 下記積分をリーマン和として m 台のプロセッサを用いて並列計算するプログラムを作れ。 m はプログラム実行時に `$ mpirun -np m ...` で指定する。
- プロセッサ数の増加 ($m=1, 2, 3, \dots$) に応じて処理時間がどの程度短縮するかを調べてレポートにまとめよ。

$$\int_0^1 \frac{4}{1+x^2} dx = \pi$$