Universidade de Brasília
Faculdade do Gama
Engenharia de Software

PSPD – Programação MPI
Comunicação coletiva

Fernando W Cruz

Introdução à operações coletivas em MPI

- Operações coletivas são chamadas por processos em um comunicador
- MPI_Bcast distribui dados de um processo (raiz) para todos os demais no comunicador
- MPI_Reduce combina dados de todos os processos do comunicador e retorna-os para um processo
- Em muitos algoritmos numéricos, as operações SEND/RECV podem ser trocadas por BCAST/REDUCE para melhorar simplicidade e eficiência

Comunicação coletiva no MPI

- Refere-se à comunicação e à computação coordenada entre um grupo de processos em um comunicador
- Tags não são usadas; comunicadores diferentes produziriam uma funcionalidade similar
- Geralmente as operações coletivas são não bloqueantes
- Podem ser divididas em três tipos de operações: sincronização, movimento dos dados e computação coletiva

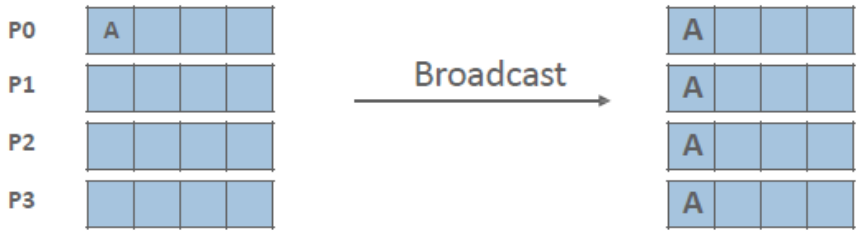
Sincronização

- MPI_Barrier (comunicador)
  - Bloqueia até que todos os processos no grupo do comunicador chamem essa primitiva
  - Um processo não sai fora da barreira até que todos os processos tenham alcançado a barreira

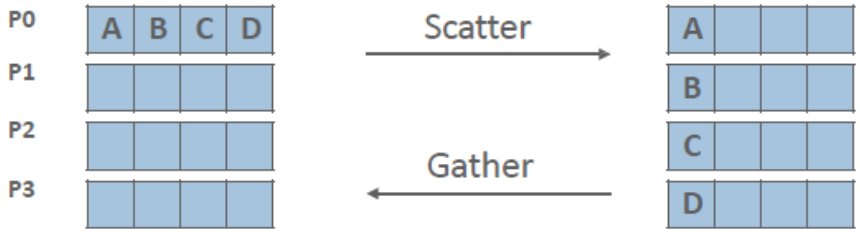
Exemplo: Elaborar um programa MPI para imprimir o rank do processo em ordem crescente, usando MPI_Barrier

Movimentação de dados coletiva (I)

```
int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm )
```



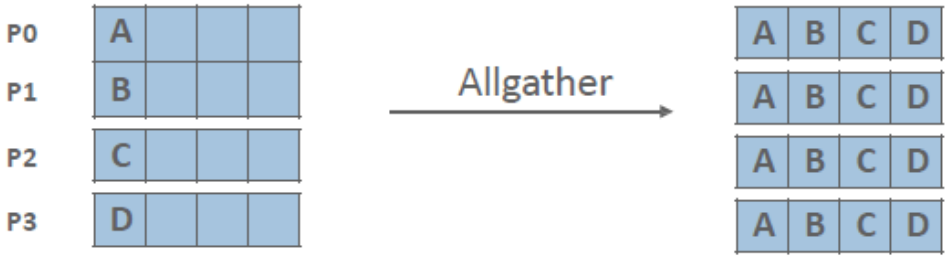

```
int MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount,
                MPI_Datatype recvtype, int root, MPI_Comm comm)
```




```
int MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount,
               MPI_Datatype recvtype, int root, MPI_Comm comm)
```

Movimentação de dados coletiva (II)

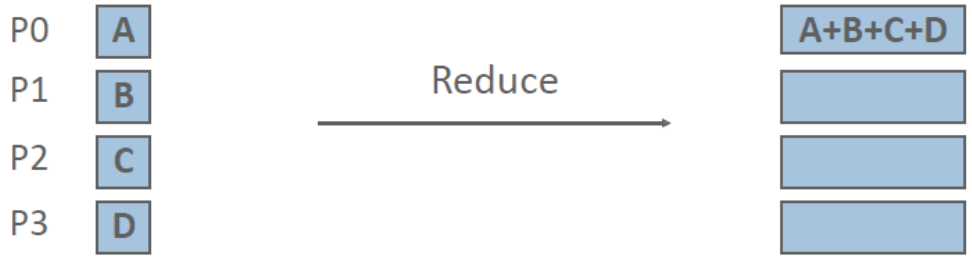
```
int MPI_Allgather(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount,
                  MPI_Datatype recvtype, MPI_Comm comm)
```



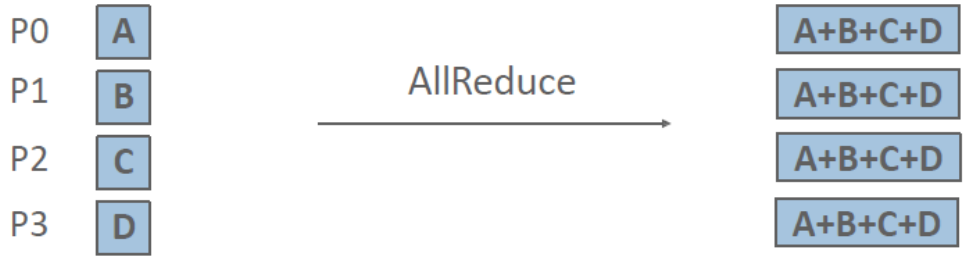

```
int MPI_Alltoall(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount,
                 MPI_Datatype recvtype, MPI_Comm comm)
```

Computação coletiva

```
int MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root,
               MPI_Comm comm)
```




```
int MPI_Allreduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op,
                  MPI_Comm comm)
```



Exemplo com MPI_Scatter

```c
#include <stdio.h>
#include <mpi.h>
#define MASTER 0

int main(int argc, char *argv[]) {
        int rank, size;
        int dados[4]={39, 42, 78, 129};
        int dado_espalhado;

        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        MPI_Scatter(&dados, 1, MPI_INT, &dado_espalhado, 1, MPI_INT, MASTER, MPI_COMM_WORLD);
        printf("Processo %d/%d recebeu %d\n", rank, size, dado_espalhado);

        MPI_Finalize();
        return 0;
} /*fim-main */
```

O que acontece se esse programa for executado por uma quantidade de processos maior ou menor do que o tamanho do vetor?

Exemplo com MPI_Scatter

```
int size, rank;
float *rand_nums=NULL;
if (argc != 2) exit(1);
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
int elementos_por_proc=atoi(argv[1]);
float *sub_rand_nums = malloc(sizeof(float) * elementos_por_proc);
if (rank == MASTER) {
        rand_nums = (float *) malloc(sizeof(float)*(elementos_por_proc*size));
        for (int i=0; i<(elementos_por_proc*size); i++)
                rand_nums[i]=i;
} /* fim-if */
MPI_Scatter(rand_nums, elementos_por_proc, MPI_FLOAT,
            sub_rand_nums, elementos_por_proc, MPI_FLOAT, MASTER, MPI_COMM_WORLD);
float media = calcula_media(sub_rand_nums, elementos_por_proc);
float *sub_avgs=NULL;
if (rank == MASTER)
        sub_avgs=malloc(sizeof(float)*size);
MPI_Gather(&media, 1, MPI_FLOAT, sub_avgs, 1, MPI_FLOAT, MASTER, MPI_COMM_WORLD);
if (rank == MASTER) {
        float mediafinal=0;
        for (int i=0; i<size; i++) mediafinal+=sub_avgs[i];
        printf("A media é igual a %.2f\n", mediafinal/size);
}
MPI_Finalize();
```

Operações válidas

- MPI_MAX – máximo de um domínio
- MPI_MIN – mínimo de um domínio
- MPI_PROD – produto
- MPI_SUM – soma
- MPI_LAND – And lógico
- MPI_LOR – Or lógico
- MPI_LXOR – Exclusive-or lógico
- MPI_BAND – Bitwise and
- MPI_BOR – Bitwise or
- ...

Exemplo: Elaborar um programa MPI para calcular e imprimir a média dos valores dos ranks dos processos, usando função MPI_Reduce

Exemplo: Elaborar um programa MPI de modo que os processos calculem, colaborativamente a média dos valores de um vetor de 8 posições. Nesse caso, dividir uma quantidade de posições para cada processo usando MPI_Scatter e recolha as parciais usando MPI_Gather.

Universidade de Brasília
Faculdade do Gama
Engenharia de Software

PSPD – MPI Comunicação coletiva

Fernando W Cruz