Programação Paralela com MPI

ELC139 - Programação Paralela

João Vicente Ferreira Lima (UFSM)

Universidade Federal de Santa Maria
jvlima@inf.ufsm.br
http://www.inf.ufsm.br/~jvlima

2023/1

Outline

Outline

Comunicações coletivas

- O que poderia melhorar com relação a versão vista anteriormente?

```
1 /* Input: a, b, n */
2 h = (b - a)/n;
3 approx = (f(a) + f(b))/2.0;
4 for (i = 1; i <= n - 1; i++) {
5   x i = a + i*h;
6   approx += f(x i);
7 }
8 approx = h*approx;
```

Aproximação trapezoidal com MPI

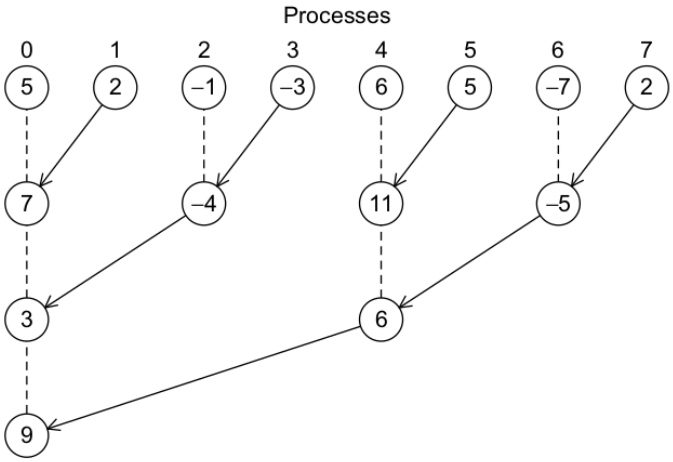
```
1   /* Add up the integrals calculated by each process */
2   if (my_rank != 0) {
3       MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0,
4             MPI_COMM_WORLD);
5   } else {
6       total_int = local_int;
7       for (source = 1; source < comm_sz; source++) {
8         MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0,
9             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
10        total_int += local_int;
11      }
12  }
```

Comunicações coletivas

- Processo mestre faz todo o trabalho de soma!!
- Como distribuir esse trabalho entre os processos?

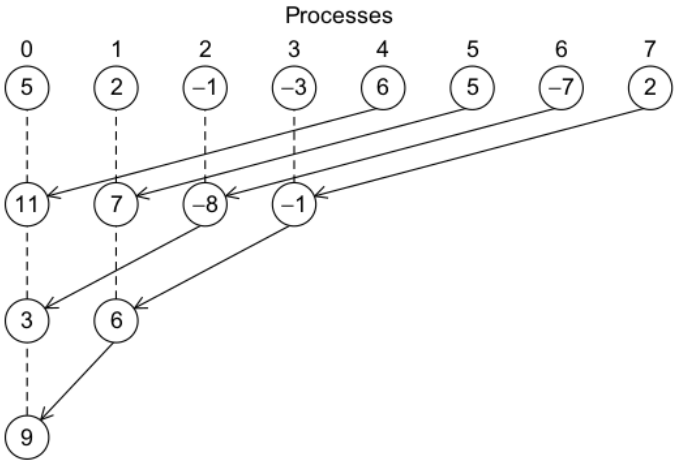
```
1  /* Input: a, b, n */
2  h = (b - a)/n;
3  approx = (f(a) + f(b))/2.0;
4  for (i = 1; i <= n - 1; i++) {
5    x i = a + i*h;
6    approx += f(x i);
7  }
8  approx = h*approx;
```

Soma estruturada em árvore



An Introduction to Parallel Programming, Peter Pacheco, 2011.

Soma estruturada em árvore



An Introduction to Parallel Programming, Peter Pacheco, 2011.

Aproximação trapezoidal com MPI

Uma versão com comunicação coletiva.

```
1     MPI_Reduce( &local_int, &total_int,
2       1, MPI_DOUBLE,
3       MPI_SUM,
4       0, MPI_COMM_WORLD );
```

Comunicações coletivas

- Todos os processos precisam chamar a mesma função coletiva no mesmo comunicador.
- Os argumentos precisam ser compatíveis. Se um processo coloca como destino 0 e outro 1, o programa trava ou aborta.
- O(s) parâmetro(s) de saída são usados apenas no processo destino.
- As comunicações ponto-a-ponto usam tags e comunicadores como Identificador. Por outro lado, comunicações coletivas usam apenas o comunicador e *ordem* de execução.

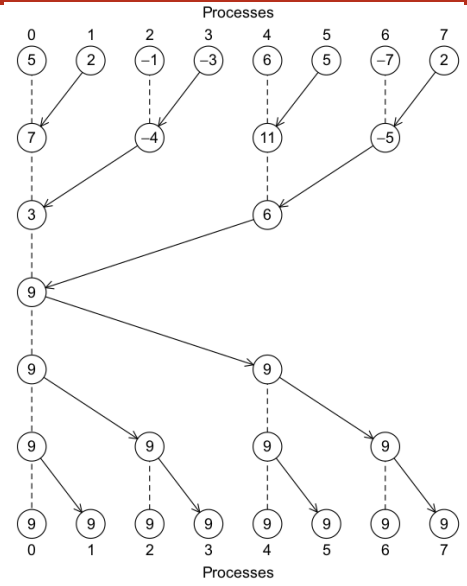
Outline

MPI_Allreduce

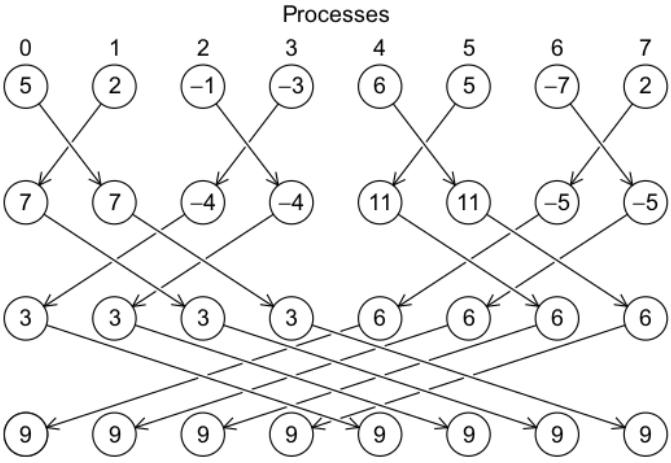
Quando todos os processos precisam do resultado.

```
1  int MPI_Allreduce(
2    const void *sendbuf,          /* in */
3    void *recvbuf,                /* out */
4    int count,                    /* in */
5    MPI_Datatype datatype,        /* in */
6    MPI_Op op,                    /* in */
7    MPI_Comm comm                 /* in */
8  )
```

MPI_Allreduce



MPI_Allreduce



An Introduction to Parallel Programming, Peter Pacheco, 2011.

informática
Universidade Federal de Santa Maria

Universidade Federal de Santa Maria
1960

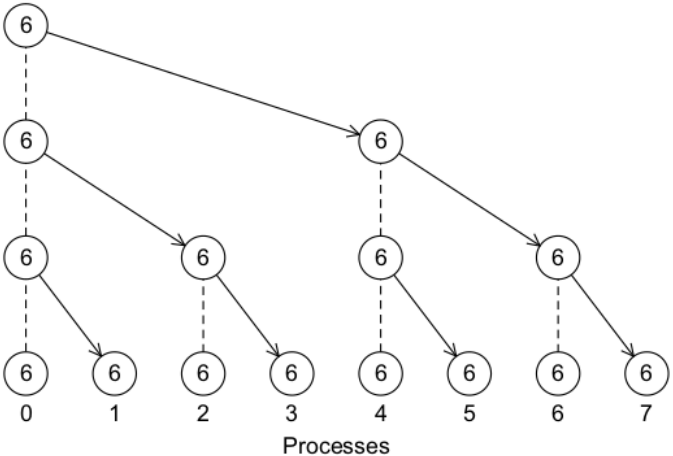
Outline

Broadcast

Dados de um processo são enviados a todos os outros do comunicador.

```
1  int MPI_Bcast(
2      void *buffer,
3      int count,
4      MPI_Datatype datatype,
5      int root,
6      MPI_Comm comm
7    )
```

Broadcast



An Introduction to Parallel Programming, Peter Pacheco, 2011.

Outline

Soma vetorial

Podemos usar broadcast de vetores?

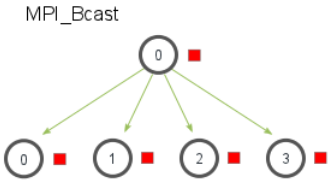
```
1  void Parallel_vector_sum(
2        double  local_x[]  /* in  */,
3        double  local_y[]  /* in  */,
4        double  local_z[]  /* out */,
5        int     local_n    /* in  */) {
6     int local_i;
7
8     for (local_i = 0; local_i < local_n; local_i++)
9        local_z[local_i] = local_x[local_i] + local_y[local_i];
10 }  /* Parallel_vector_sum */
```

Scatter

Scatter divide os dados entre os processos e envia um pedaço para cada processo.

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

https://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/



Soma vetorial

Scatter divide os dados entre os processos e envia um pedaço para cada processo.

```
1  if (my_rank == 0) {
2    /* lê dados de algum lugar */
3    MPI_Scatter(a, local_n, MPI_DOUBLE, local_a, local_n,
4      MPI_DOUBLE, 0, comm);
5  } else {
6    MPI_Scatter(a, local_n, MPI_DOUBLE, local_a, local_n,
7      MPI_DOUBLE, 0, comm);
8  }
```

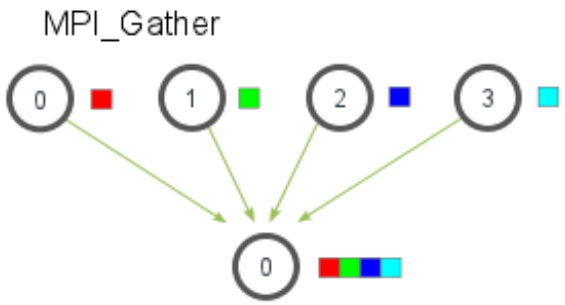
Outline

Gather

Gather coleta dados de todos os processos e agrupa.

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

https://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/



informática
Universidade Federal de Santa Maria

Universidade Federal de Santa Maria
1960

Soma vetorial

Gather coleta dados de todos os processos e agrupa.

```
1 if (my_rank == 0) {
2   MPI_Gather(local_b, local_n, MPI_DOUBLE, b, local_n,
3     MPI_DOUBLE, 0, comm);
4 } else {
5   MPI_Gather(local_b, local_n, MPI_DOUBLE, b, local_n,
6     MPI_DOUBLE, 0, comm);
7 }
```

Outline

Allgather

Multiplicação vetor matriz – $y = Ax$

```
1  for (i = 0; i < m; i++) {
2    y[i] = 0.0;
3    for (j = 0; j < n; j++)
4      y[i] += A[i][j]*x[j];
5    }
```

Allgather

Multiplicação vetor matriz – $y = Ax$

a_{00}	a_{01}	$\cdots$	$a_{0,n-1}$
a_{10}	a_{11}	$\cdots$	$a_{1,n-1}$
$\vdots$	$\vdots$		$\vdots$
a_{i0}	a_{i1}	$\cdots$	$a_{i,n-1}$
$\vdots$	$\vdots$		$\vdots$
$a_{m-1,0}$	$a_{m-1,1}$	$\cdots$	$a_{m-1,n-1}$

x_0
x_1
$\vdots$
x_{n-1}

$=$

y_0
y_1
$\vdots$
$y_i = a_{i0}x_0 + a_{i1}x_1 + \cdots a_{i,n-1}x_{n-1}$
$\vdots$
y_{m-1}

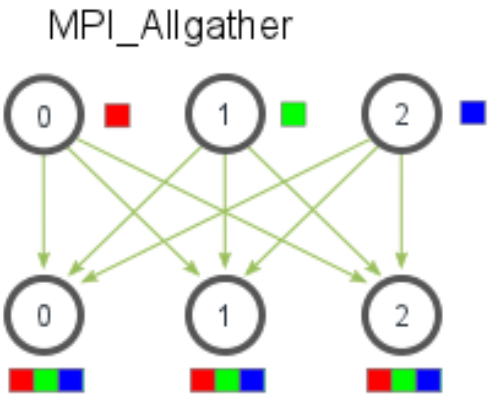
Allgather

```
1  void Mat_vect_mult(
2        double  A[]  /* in  */,
3        double  x[]  /* in  */,
4        double  y[]  /* out */,
5        int     m    /* in  */,
6        int     n    /* in  */) {
7    int i, j;
8
9    for (i = 0; i < m; i++) {
10     y[i] = 0.0;
11     for (j = 0; j < n; j++)
12       y[i] += A[i*n+j]*x[j];
13     }
14   }  /* Mat_vect_mult */
```

Allgather

Agrupa dados de todos os processos e faz *broadcast*.

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



https://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/

Allgather

```
1  double* x;
2  int local_i, j;
3  int local_ok = 1;
4
5  x = malloc(n*sizeof(double));
6  MPI_Allgather(local_x, local_n, MPI_DOUBLE,
7        x, local_n, MPI_DOUBLE, comm);
8
9  for (local_i = 0; local_i < local_m; local_i++) {
10   local_y[local_i] = 0.0;
11   for (j = 0; j < n; j++)
12       local_y[local_i] += local_A[local_i*n+j]*x[j];
13 }
14 free(x);
```

https://joao-ufsm.github.io/par2023a/