

# How to Parallelize a given application with a simple example

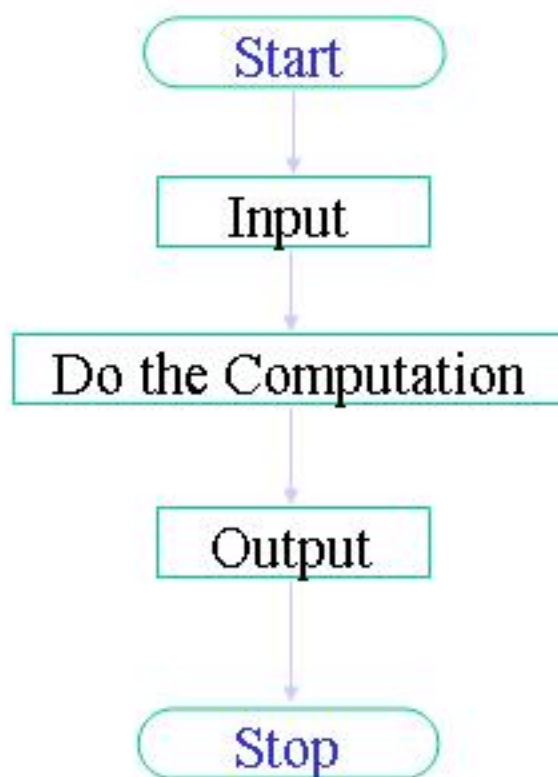
V. Sundararajan  
Thanks to Dr. S.S. Kadam  
C-DAC, Pune

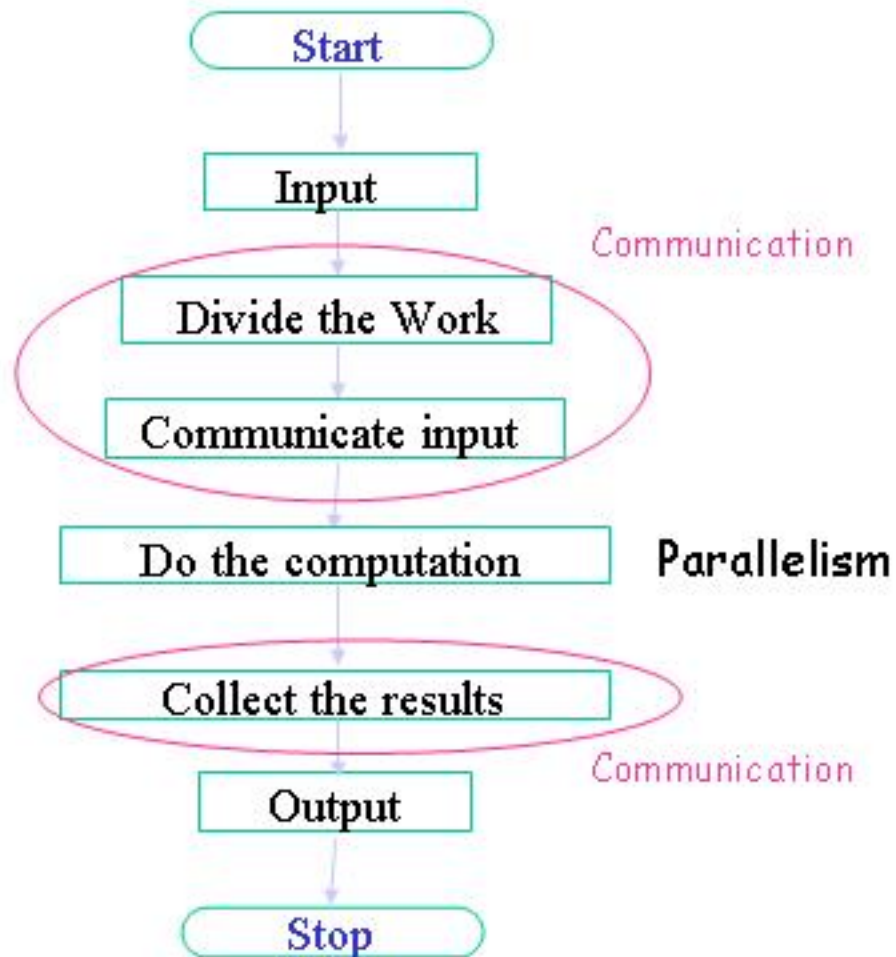
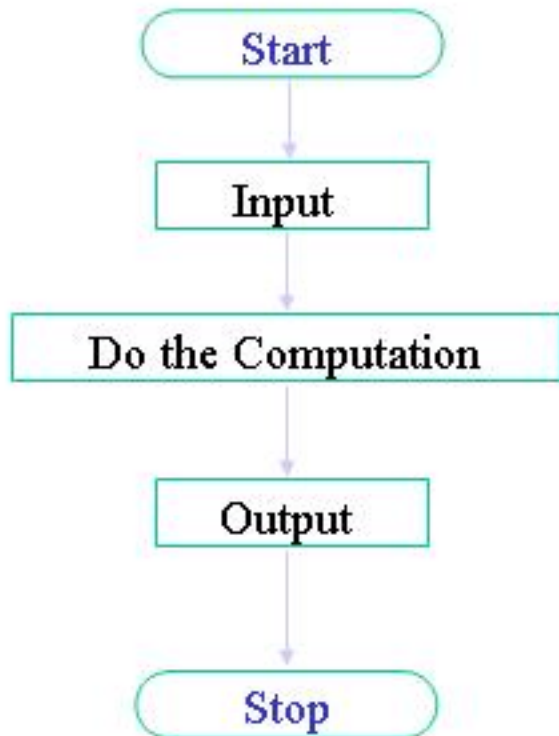
- **New Development**
- **Parallelisation of existing program**
- **Whether parallelisation is required or not**
  - **Parallelise**
    - Application that consumes lot of time and also to be repeatedly used (ab initio molecular dynamics, accurate quantum chemistry, N-body simulations)
    - Application that cannot be fit in to PC because of memory requirement or computation complexity (large scale CFD codes, multi-scale modeling )
    - Less time consuming but frequently used codes (daily forecast of weather with less resolution, Engineering simulations for more parameter sets)
  - **Don't parallelise**
    - Time consuming but not going to be used more than few times
    - Sequential part of the algorithm is dominant (more discussion with Amdahl's law)

- One needs to judge the parallelism from the algorithm that is going to be used.
- Possible to develop an efficient parallel code

- Compile and link the code with `-pg` option
- Execute it to know its profile (for at least two or three (small, medium and large) data sizes)
- Judge the most time consuming part
- Design the parallel method for that part
- Decide the programming model
- Implement
- Measure its performance
- Tune the parallel code based on the performance
- Don't attempt if the underlying algorithm is not understood

- Let us consider summation of  $10^{12}$  real numbers between 0 and 1.
  - To compute  $\sum x_i$  for  $(i=1,2,\dots, 10^{12})$
- This may sound trivial but, useful in illustrating the practical aspects of parallel program development
- Total computation of  $10^{12}$  operations to be done
- On a PC, it would take roughly 500 secs assuming 2 GFLOPS sustained performance
- Can we speed this up and finish in  $1/8^{\text{th}}$  on 8 processors





## Data Parallel

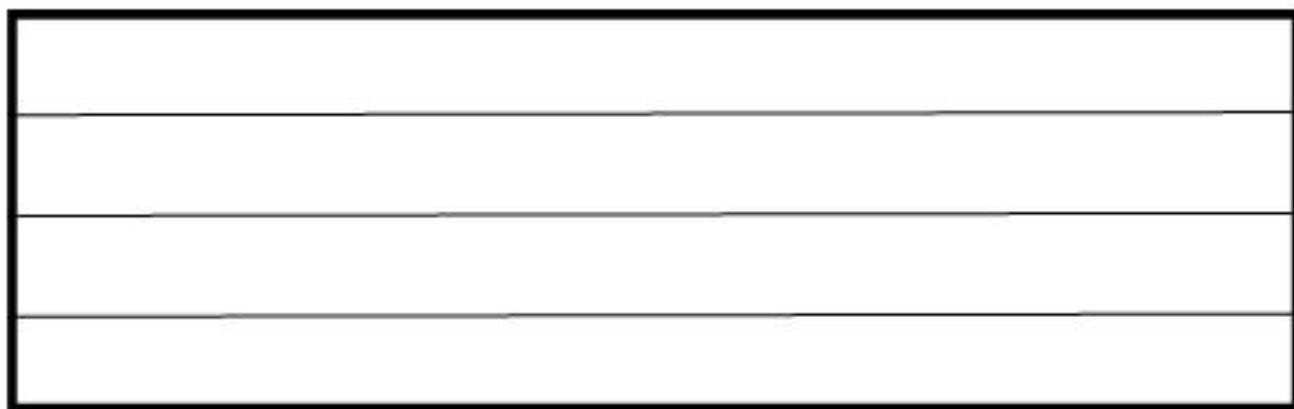
- Teachers correcting same subject papers
- Data division, domain decomposition

P4

P3

P2

P1



## Pipelining

- **Assembly line in a manufacturing unit**
- **Correction of answers to specific questions by each teacher**
- **Instruction fetch, decode, operand fetch, execute and write.**  
eg. **printing and editing.**

P1

P2

P3

P4

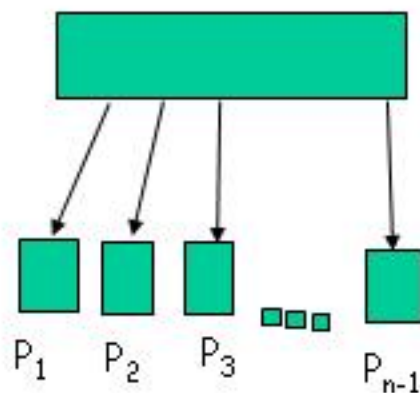
P5

5  
4  
3  
2  
1

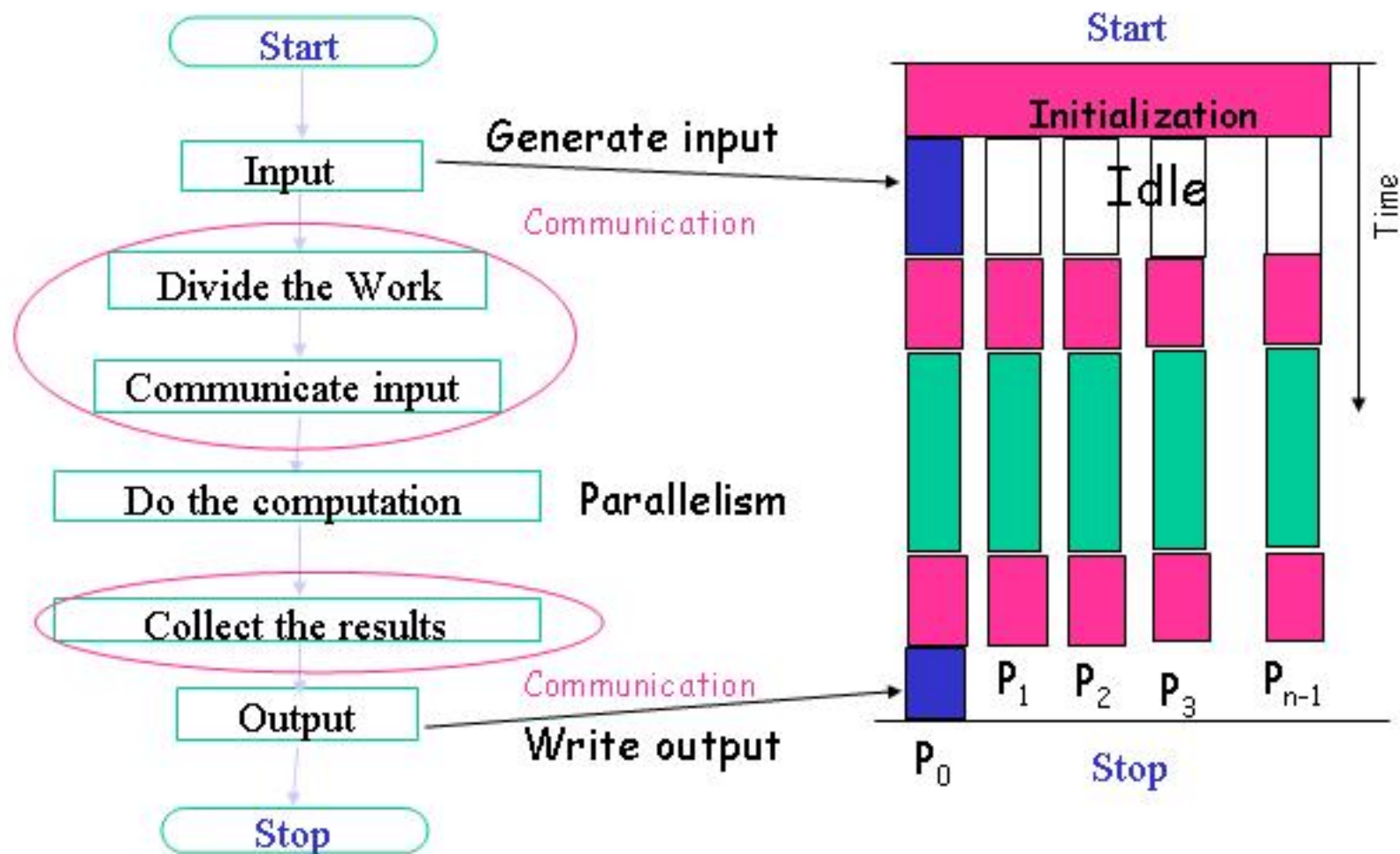

- **Generation of data or reading input**
- **Distribution of work and corresponding input**
- **Implementation of the method by using MPI**
  - **Whether the communication required frequently?**
  - **Whether it is nearest neighbour communication?**
  - **Whether synchronisation required?**

- **Generation of data or reading input**
  - **Option 1: generate on master and send part data to each**

Process 0 : Master either generates or reads as input

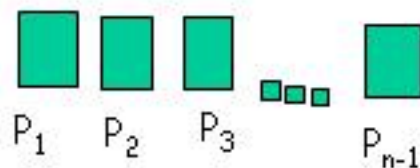


Disadvantage: Requires communication



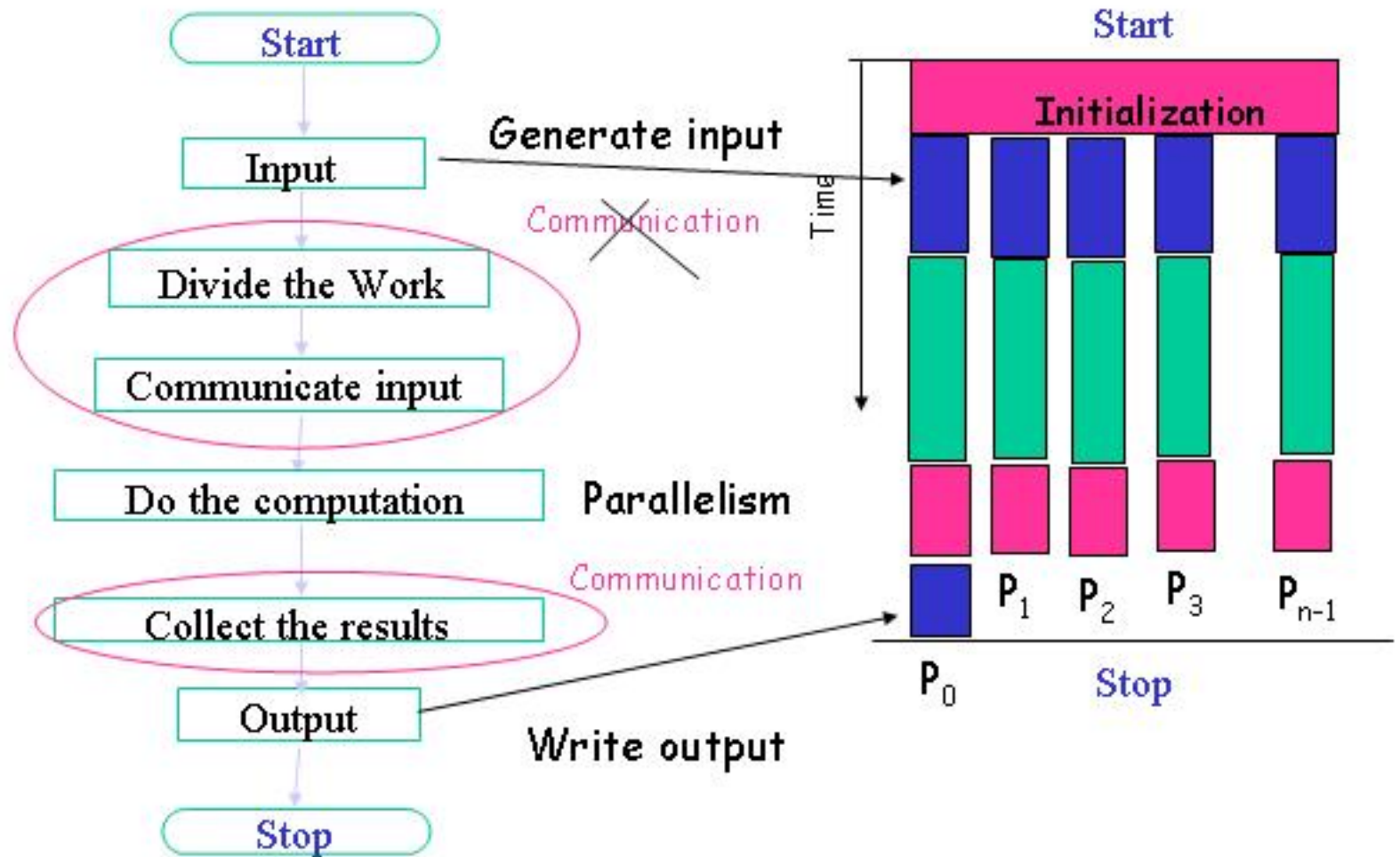
- **Generation of data or reading input**
  - **Option 2: generate simultaneously on each process**

All processes generate inputs



Advantage: No communication

## Option 2: Generate inputs simultaneously



## Fortran

```
program summation
double precision x,xsum
integer i
xsum=0.0

do i=1,10000000
    x=float(mod(i,10))/10.0
    xsum=xsum+x
enddo

print *,'xsum= ',xsum

stop
end
```

## C

```
main()
{
double x,xsum;
int i;
xsum=0.0;

for ( i=1; i<10000000;i++)
{
    x=float(mod(i,10))/10.0;
    xsum=xsum+x;
}
printf(" xsum=  %f \n",xsum);
}
```

## Fortran

```
program summation
double precision x,xsum
integer i

xsum=0.0
do i=1,100000000
  x=float(mod(i,10))/10.0
  xsum=xsum+x
enddo

print *, 'xsum= ',xsum
stop
end
```

```
program summation
include "mpif.h"
double precision x,xsum,tsum
integer i
Integer myid,nprocs,IERR
Call MPI_INIT(IERR)
Call MPI_COMM_RANK(MPI_COMM_WORLD,myid,IERR)
Call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,IERR)
xsum=0.0
do i=myid+1,100000000,nprocs
  x=float(mod(i,10))/10.0
  xsum=xsum+x
enddo
Call MPI_REDUCE(xsum,tsum,1,MPI_DOUBLE_PRECISION,
               MPI_SUM,0,MPI_COMM_WORLD,IERR)
If(myid.eq.0) print *, 'tsum= ',tsum
Call MPI_Finalize(IERR)
stop
end
```

C

```
main()
{
double x,xsum;
int l;

xsum=0.0;
for ( i=1; i<100000000;i++)
{
x=float(mod(i,10))/10.0;
xsum=xsum+x;
}

Printf(" xsum= %f \n",xsum);
}
```

```
#include "mpif.h"
main()
{
double x,xsum,tsum;
int l;
Int myid,nprocs,IERR;
MPI_Init(argc, *argv);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
xsum=0.0;
for ( i=myid+1; i<100000000,i=i+nprocs)
{
x=float(mod(i,10))/10.0;
xsum=xsum+x;
}
MPI_Reduce(&xsum,&tsum,1,MPI_Double,
MPI_SUM,0,MPI_COMM_WORLD);
If(myid == 0) printf(" tsum= %f \n",tsum);
MPI_Finalize()
}
```

Sequential	Parallel
Design algorithm step by step and write the code assuming single process to be executed	Redesign the same algorithm with the distribution of work assuming many processes to be executed simultaneously
<code>f90 -o sum.exe sum.f</code> <code>gcc -o sum.exe sum.c</code> Only one executable created	<code>mpif77 -o sum.exe sum.f</code> <code>mpicc -o sum.exe sum.f</code> Only one executable created
<code>sum.exe</code> Executes a single process	<code>mpirun -np 8 sum.exe</code> Executes 8 processes <b>simultaneously</b>
Single point of failure Good tools available for debugging	Cooperative operations but, multiple points of failure; Tools are still evolving towards debugging parallel executions.

1	2	3	4	5	6	7	8	9	10
2									
3									
4									
5									
6									
7									
8									

↑  
nrem

		Endcol		Endcol		Endcol		Endcol	
		3		6		8		10	
1	2	3	4	5	6	7	8	9	10
2									
3									
4									
5									
6									
7									
8									
		Startcol		Startcol		Startcol		Startcol	
		1		4		7		9	

SEC G/2007

**! Division of work that would be done by each process**

```
ncolpp(1)=NCB/NPROCS
nrem=NCB - ncolpp(1)*NPROCS
do i=1,nrem
    ncolpp(i)= ncolpp(1) +1
enddo
startcol(1)=1
endcol(1)=ncolpp(1)
do i=2,nprocs-1
    startcol(i) = endcol(i-1) + 1
    endcol(i) = startcol(i) + ncolpp(i) - 1
enddo
```

! Do matrix multiply sharing iterations on outer loop

```
DO J=startcol(MYID+1),endcol(MYID+1)
```

```
DO I=1, NRA
```

```
DO K=1, NCA
```

```
    C(I,J) = C(I,J) + A(I,K) * B(K,J)
```

```
    ENDDO
```

```
  ENDDO
```

```
ENDDO
```

! End of parallel region

MYID	0	1	2	3	
Index	1	2	3	4	MYID+1

## Collection of Results

```
If(MYID.EQ.0) then
  DO i=2,NPROCS
    CALL MPI_RECV(C(1,startcol(i)), ncolpp(i)*NRA, &
      mpi_double_precision,i-1,tag,MPI_COMM_WORLD,STATUS, IERR)
  ENDDO
Else
  CALL MPI_SEND(C(1,startcol(MYID+1)),ncolpp(MYID+1)*NRA, &
    mpi_double_precision,0,tag,MPI_COMM_WORLD,IERR)
endif
```