Sample codes are available on Ada at

/general/public/training/mpi/Fall2017/part1
/general/public/training/mpi/Fall2017/part2

Introduction to Code Parallelization Using MPI (Part II)

Ping Luo

TAMU HPRC

October 26, 2017

HPRC Short Course – Fall 2017

HIGH PERFORMANCE RESEARCH COMPUTING
TEXAS A&M UNIVERSITY

Outline

- Exploring parallelism
  - Task-parallelism
  - Data-parallelism
- Data distribution
- Self-scheduling
- Matrix-vector multiplication
- Solving the 2D Poisson equation
- Domain decomposition
- MPI/OMP hybrid programming
- MPI/OMP: A Case Study

Important Note On Using MPI

- All parallelism is explicit: the programmer is responsible for correctly identifying parallelism and implementing parallel algorithms using MPI constructs

Exploring Parallelism

- Task-parallelism: The programmer identifies different tasks of a program and distribute the tasks among different processors
- Data-parallelism: The programmer partitions the data used in a program and distribute them among different processors, each performing similar operations on the subset of data assigned.

3 chefs need to prepare a three-course menu for 12 guests

salad steak desert

Preparing 12 salads

Preparing 12 steaks

Preparing 12 deserts

Task 1

Task 2

Task 3

Task parallelism

4 meals

salad
steak
desert

4 meals

salad
steak
desert

4 meals

salad
steak
desert

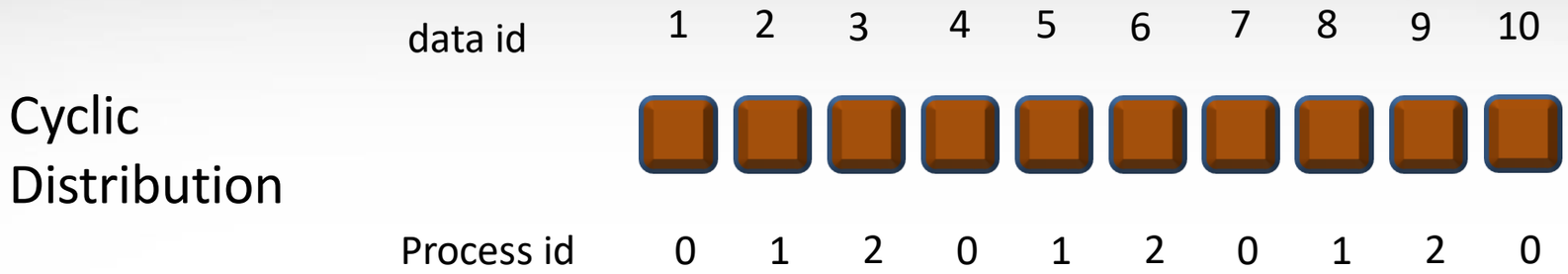
Data parallelism

https://101clipart.com/wp-content/uploads/02/Clipart%20Chef%20Cooking%2012.jpg

https://classroomclipart.com/images/gallery/Clipart/Culinary/chef-thumbs-up-holding-plate-hot-food-clipart-622.jpg

http://www.marbellafamilyfun.com/images/chef-and-waiter-wanted-for-summer-job-from-14th-august-for-2-weeks-21516737.jpg

Example 6: Data Distribution



Data is distributed in a round robin manner among the processes.

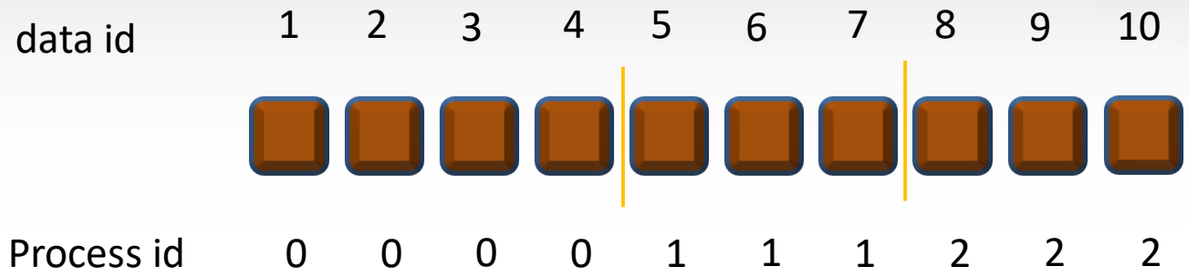
C	Fortran

`for (i=myid+1; i<=N; i+=nprocs){`
`  x = h*(i-0.5);`
`  sum += 4.0/(1.0+x*x);`
`}`
`sum = sum*h;`
	`do i=myid+1, N, nprocs`
	`  x = h*(i-0.5d0)`
	`  sum = sum+4.0d0/(1.0d0+x*x)`
	`enddo`
	`sum = sum*h;`

calc_PI_cyclic.c

Example 6: Data Distribution

Block Distribution



Data is partitioned into *n* contiguous parts, where *n* is equal to the number of processes. Each process will take one part of the data.

C

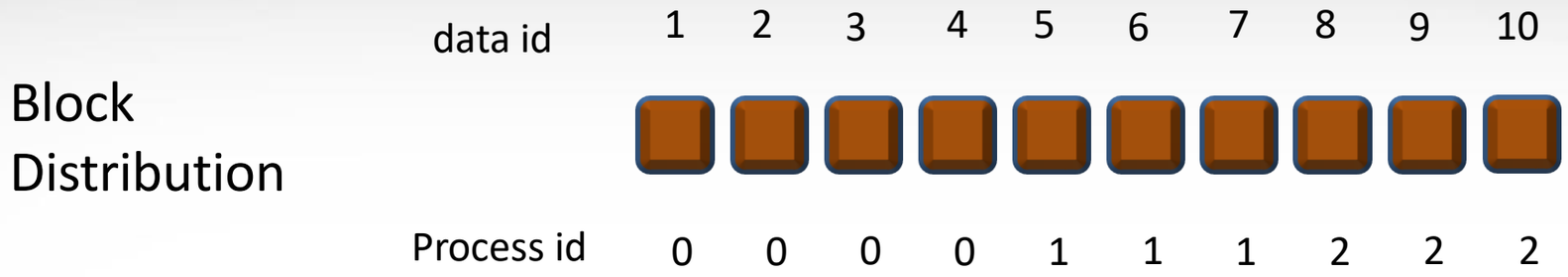
```
block_map(1,N,nprocs,myid,&l1,&l2);
for (i=l1; i<=l2; i++){
   x = h*(i-0.5);
   sum += 4.0/(1.0+x*x);
}
sum = sum*h;
```

Fortran

```
call block_map(1,N,nprocs,myid,l1,l2)
do i=l1, l2
   x = h*(i-0.5d0)
    sum = sum + 4.0d0/(1.0d0+x*x)
enddo
sum = sum*h
```

calc_PI_block.c

Example 6: Data Distribution



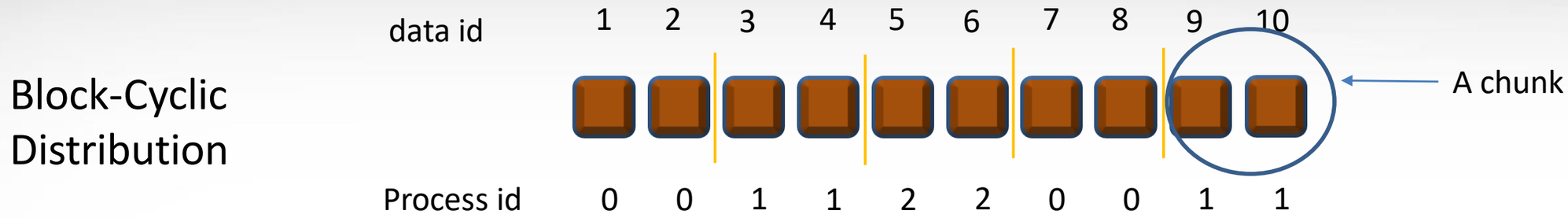
C

```
void block_map(int n1, int n2,int nprocs,
               int myid, int *l1, int *l2)
{
  int block, rem;
  block = (n2-n1+1)/nprocs;
  rem   = (n2-n1+1)%nprocs;
  if (myid < rem){
    block++;
    *l1 = n1+myid*block;
  }else
    *l1 = n1+rem+block*myid;

  *l2 = *l1+block-1;
}
```

Fortran

```
subroutine block_map(n1,n2, nprocs, myid, l1, l2)
Integer n1, n2, nprocs, myid, l1,l2

integer block, rem
block = (n2-n1+1)/nprocs
rem = mod(n2-n1+1, nprocs)
if (myid < rem) then
    block = block+1
    l1 = n1+myid*block
else
    l1 = n1+rem+block*myid
end if
l2 = l1+block-1
end subroutine block_map
```

Example 6: Data Distribution



Data is divided into chunks of contiguous blocks and the chunks distributed in a round-robin manner

Loop through chunks

Loop through blocks inside a chunk

C

```
for (i=myid*BLK+1; i<=N; i+=nprocs*BLK){
    for (j=i; j<=MIN(N,i+BLK-1); j++){
        x = h*(j-0.5);
        sum += 4.0/(1.0+x*x);
    }
}
sum = sum*h;
```

Fortran

```
do i=myid*BLK+1, N, nprocs*BLK
   do j=i, MIN(N,i+BLK-1)
       x = h*(j-0.5d0)
       sum = sum+4.0d0/(1.0d0+x*x)
  enddo
enddo
sum = sum*h
```

calc_PI_bc.c

Example 7: Self-Scheduling

$$A\vec{b} = \begin{pmatrix} \vec{a}_1 \\ \vdots \\ \vec{a}_n \end{pmatrix} \vec{b} = \begin{pmatrix} \vec{a}_1 \cdot \vec{b} \\ \vdots \\ \vec{a}_n \cdot \vec{b} \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \vec{c}$$

The algorithm distributes the computation of the dot product to any process. Each dot product is independent of the other.

Master process

"owns" A and b, and collects $c = A \times b$

1. Broadcast b each worker
2. Use tag to signify row number; send a row to each worker.
3. Loop:
   -receive a dot_product entry from worker_i
   -send a new row to worker_i from whom we just received the new dot_product;

Until all dot_products are received.

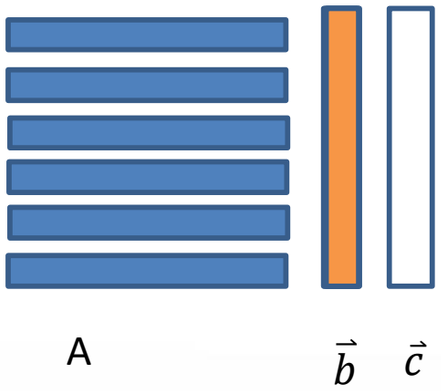
Worker process

1. Receive b
2. Loop:
   - receive a new row from master
   - compute dot product
   - send dot product to master

Until master sends termination notice.

Self scheduling: also called work load distribution scheme whereby a "master" process hands work to an idle worker process.

Example 7: Self-Scheduling



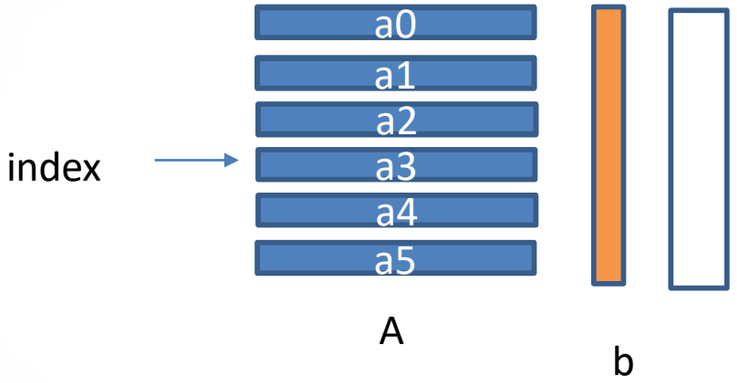
Worker 1

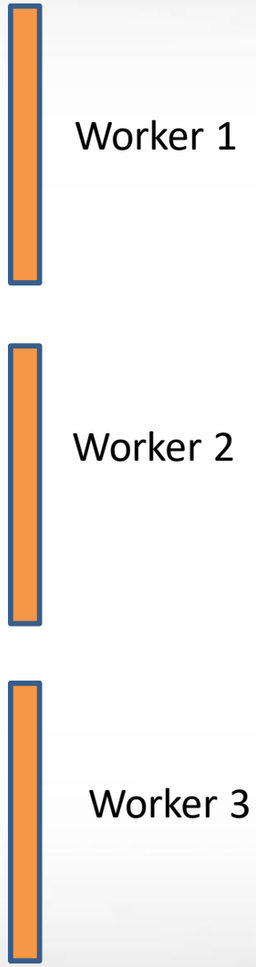
Worker 2

Worker 3

MPI_Bcast(b, n, MPI_DOUBLE, master, 0, MPI_COMM_WORLD)

Example 7: Self-Scheduling



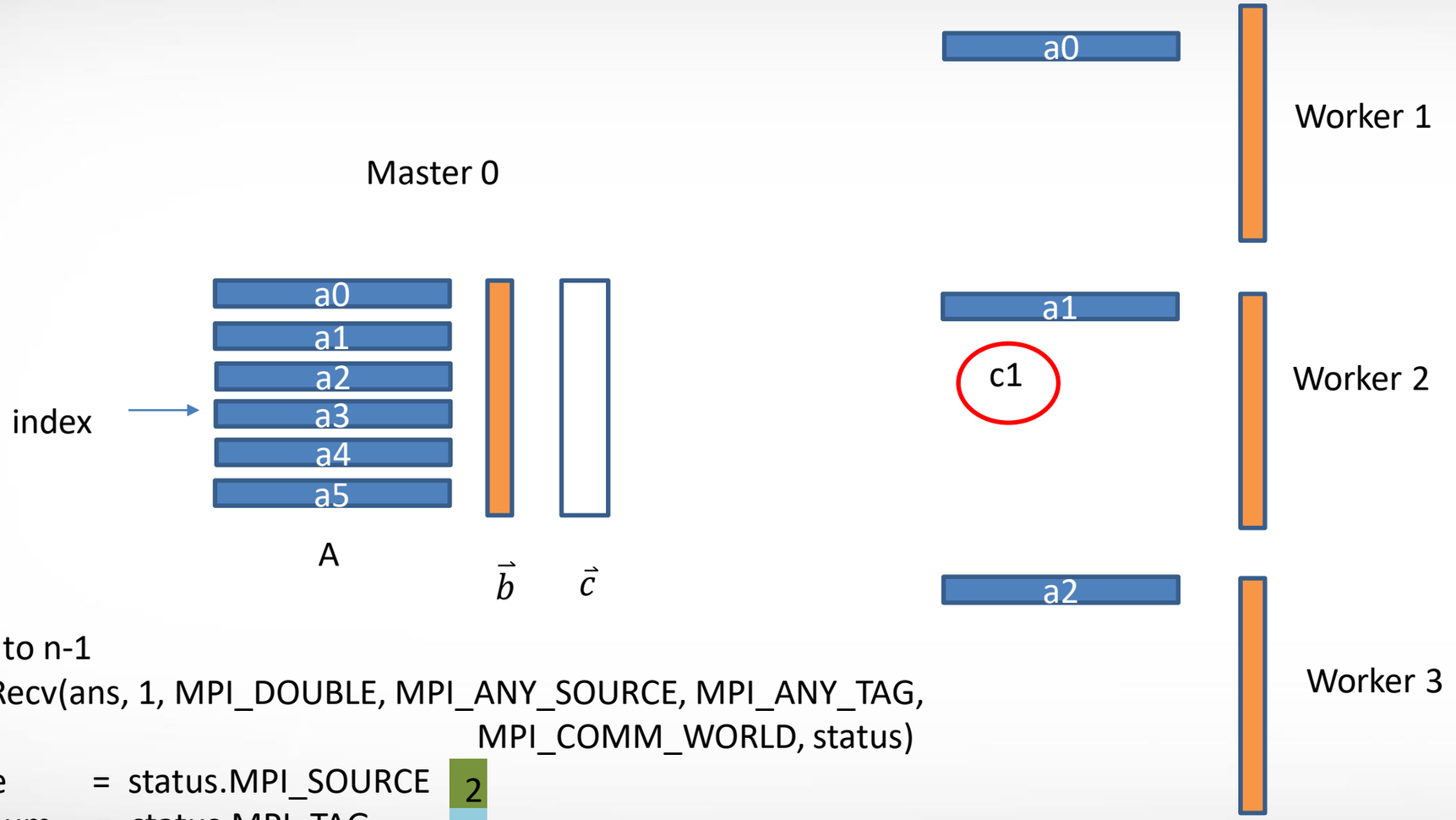


MPI_Send(a(i-1), n, MPI_DOUBLE, i, i-1, MPI_COMM_WORLD)
I = 1, 3

The tag holds the row number

MPI_Recv(a_row, n, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD)
row_num = status.MPI_TAG

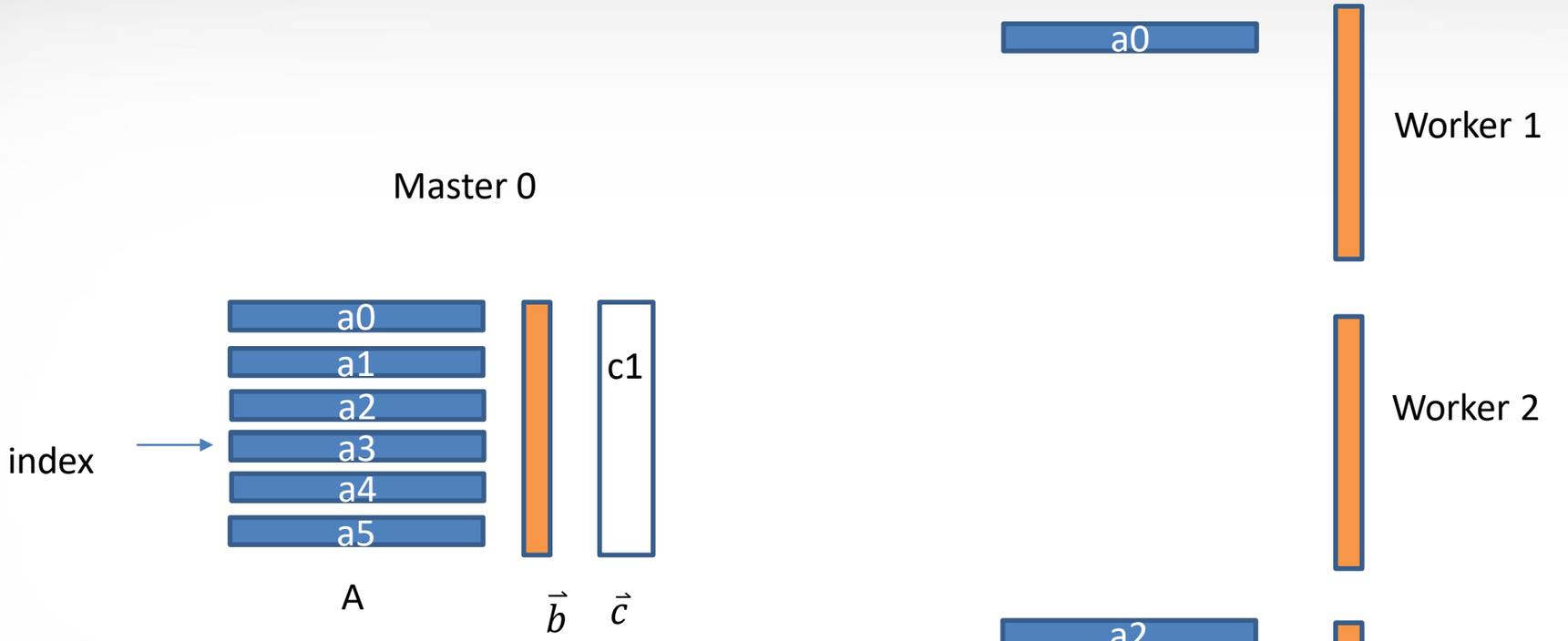
Example 7: Self-Scheduling




```
For i = 0 to n-1
   MPI_Recv(ans, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,
                                              MPI_COMM_WORLD, status)
   source      = status.MPI_SOURCE   2
   row_num     = status.MPI_TAG      1
   c(row_num) =  ans
```

```
MPI_Recv(a_row, n, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD)
row_num = status.MPI_TAG; do dot product;
MPI_Send(ans, 1, MPI_DOUBLE, 0, r 1 _num, MPI_COMM_WORLD)
```

Example 7: Self-Scheduling



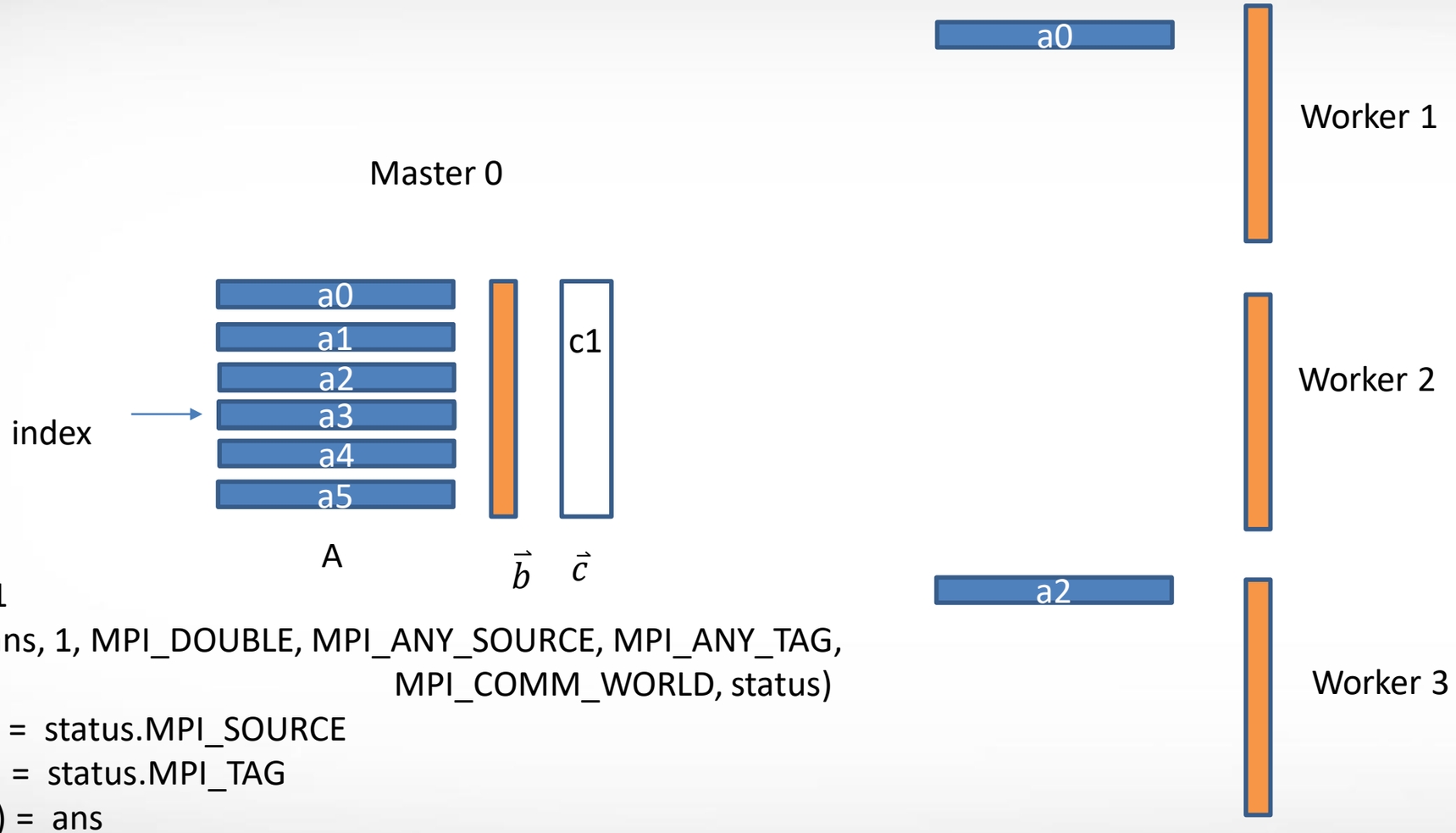

```
For i = 0 to n-1
   MPI_Recv(ans, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,
                                MPI_COMM_WORLD, status)
   source      = status.MPI_SOURCE
   row_num     = status.MPI_TAG
   c(row_num) =  ans
```

2
1

Worker 2 just finished and need more work

```
MPI_Recv(a_row, n, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD)
row_num = status.MPI_TAG; do dot product;
MPI_Send(ans, 1, MPI_DOUBLE, 0, row_num, MPI_COMM_WORLD)
```

Example 7: Self-Scheduling



For i = 0 to n-1

MPI_Recv(ans, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, status)

source = status.MPI_SOURCE

row_num = status.MPI_TAG

c(row_num) = ans

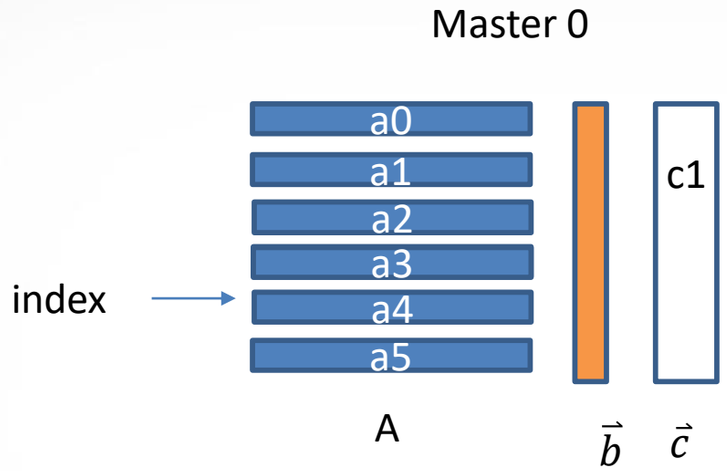
MPI_Send(a(index), n, MPI_DOUBLE, source, index, MPI_COM_WORLD)

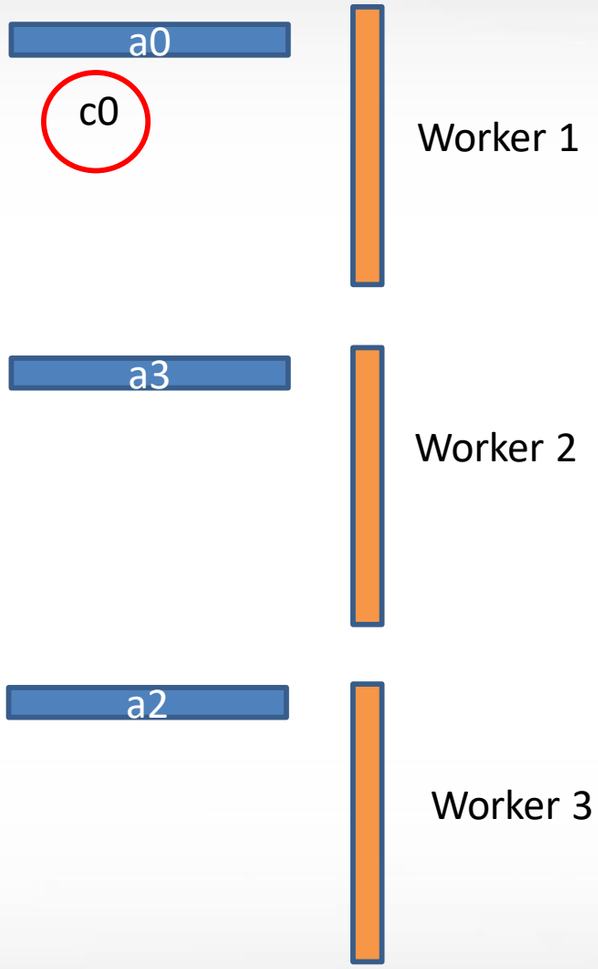
MPI_Recv(a_row, n, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD)

row_num = status.MPI_TAG; do dot product;

MPI_Send(ans, 1, MPI_DOUBLE, 0, row_num, MPI_COMM_WORLD)

Example 7: Self-Scheduling

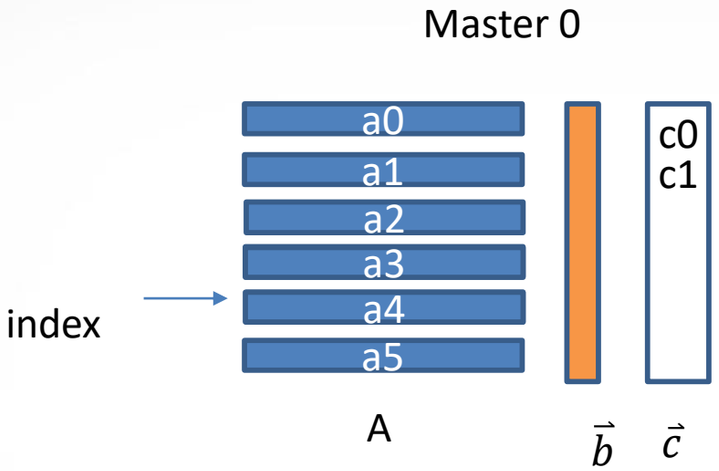


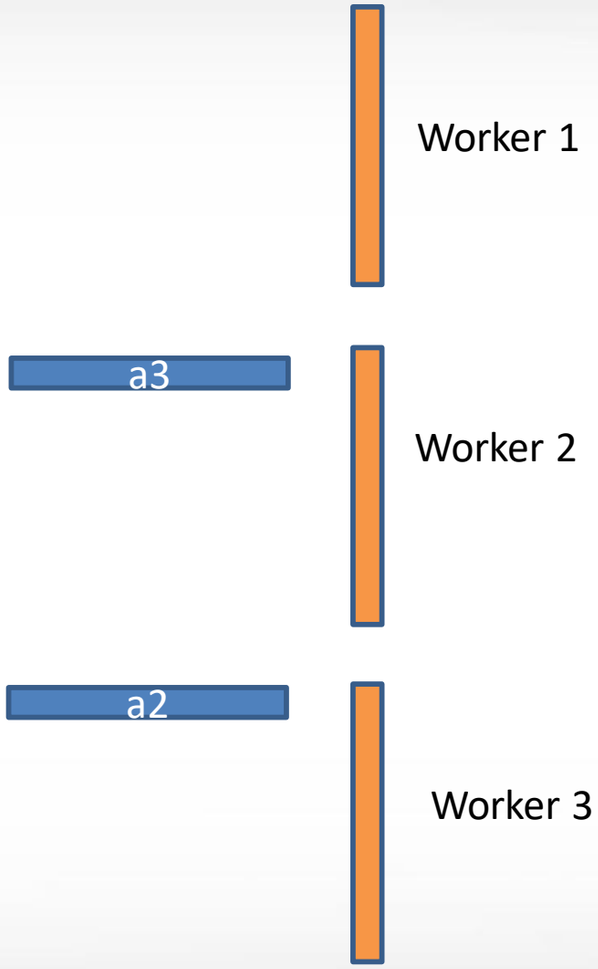



```
For i = 0 to n-1
   MPI_Recv(ans, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,
                                          MPI_COMM_WORLD, status)
   source      = status.MPI_SOURCE   1
   row_num     = status.MPI_TAG      0
   c(row_num) =  ans
   MPI_Send(a(index), n, MPI_DOUBLE, source, index, MPI_COM_WORLD)
```

```
MPI_Recv(a_row, n, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD)
row_num = status.MPI_TAG; do dot product;
MPI_Send(ans, 1, MPI_DOUBLE, 0, row_num, MPI_COMM_WORLD)
```

Example 7: Self-Scheduling

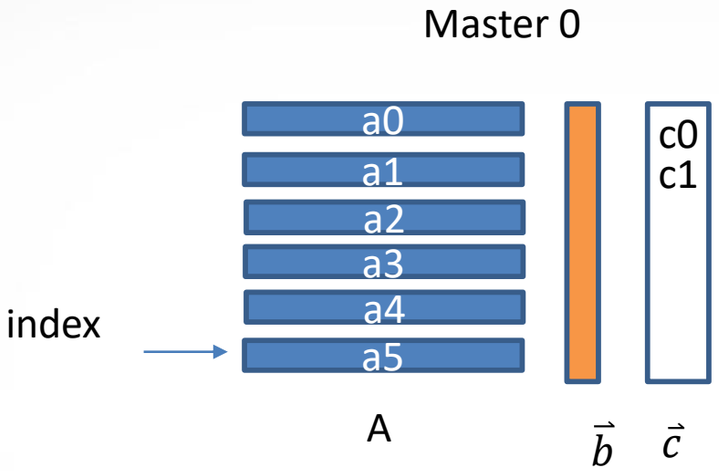


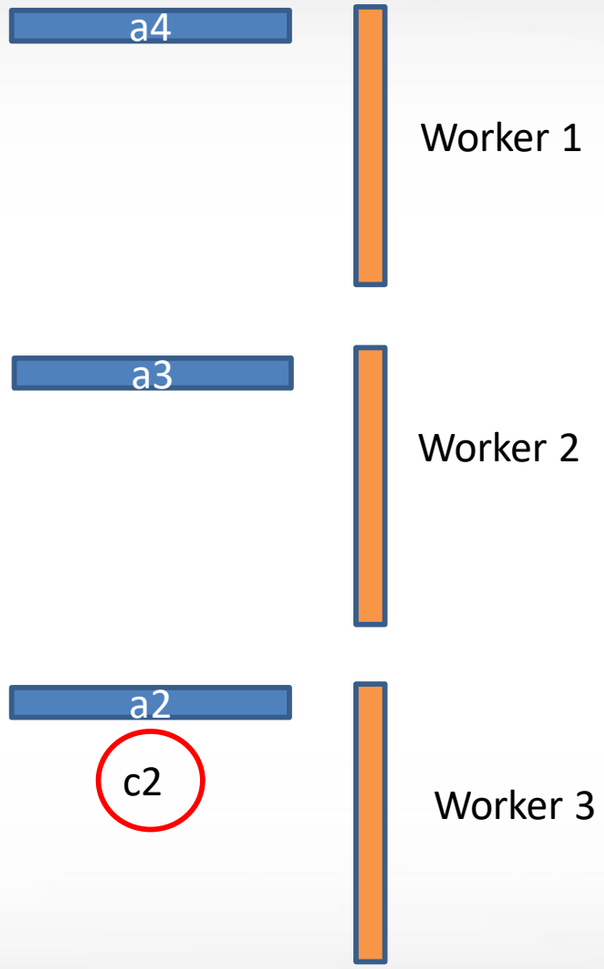



```
For i = 0 to n-1
   MPI_Recv(ans, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,
                                          MPI_COMM_WORLD, status)
   source      = status.MPI_SOURCE   1
   row_num     = status.MPI_TAG      0
   c(row_num) =  ans
   MPI_Send(a(index), n, MPI_DOUBLE, source, index, MPI_COM_WORLD)
```

```
MPI_Recv(a_row, n, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD)
row_num = status.MPI_TAG; do dot product;
MPI_Send(ans, 1, MPI_DOUBLE, 0, row_num, MPI_COMM_WORLD)
```

Example 7: Self-Scheduling

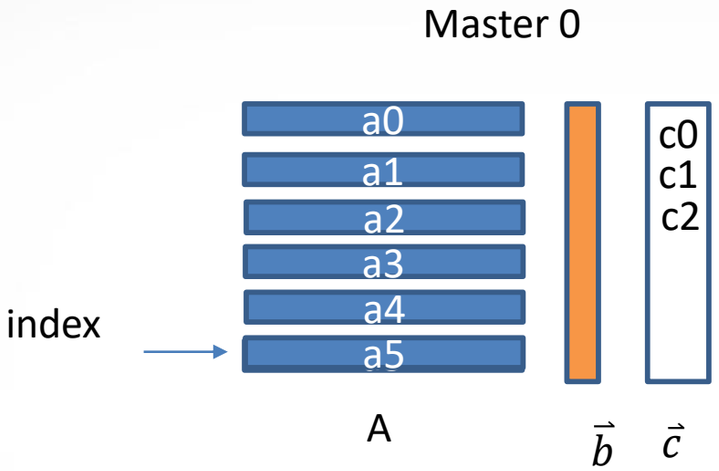


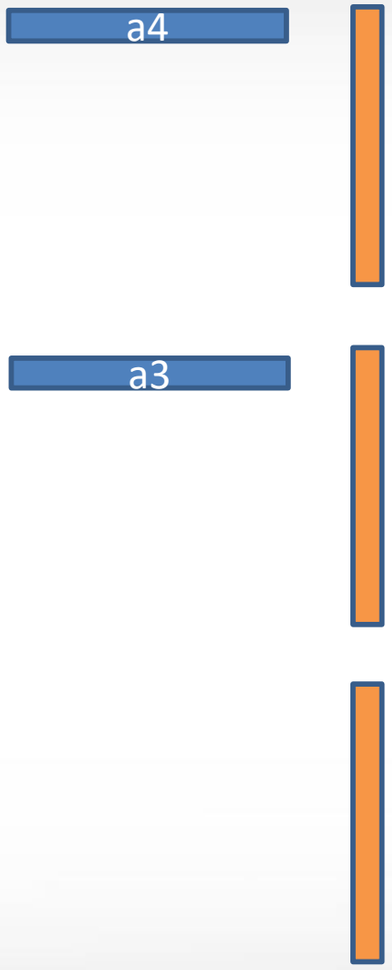



```
For i = 0 to n-1
   MPI_Recv(ans, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,
                                          MPI_COMM_WORLD, status)
   source       =  status.MPI_SOURCE
   row_num    =  status.MPI_TAG
   c(row_num) =  ans
   MPI_Send(a(index), n, MPI_DOUBLE, source, index, MPI_COM_WORLD)
```

```
MPI_Recv(a_row, n, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD)
row_num = status.MPI_TAG; do dot product;
MPI_Send(ans, 1, MPI_DOUBLE, 0, row_num, MPI_COMM_WORLD)
```

Example 7: Self-Scheduling

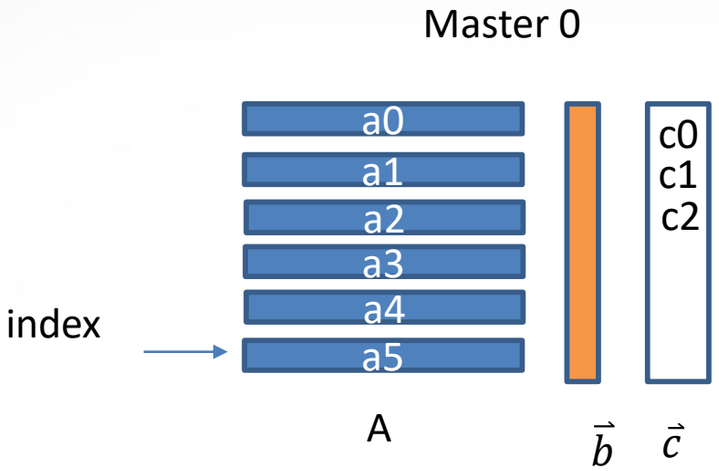


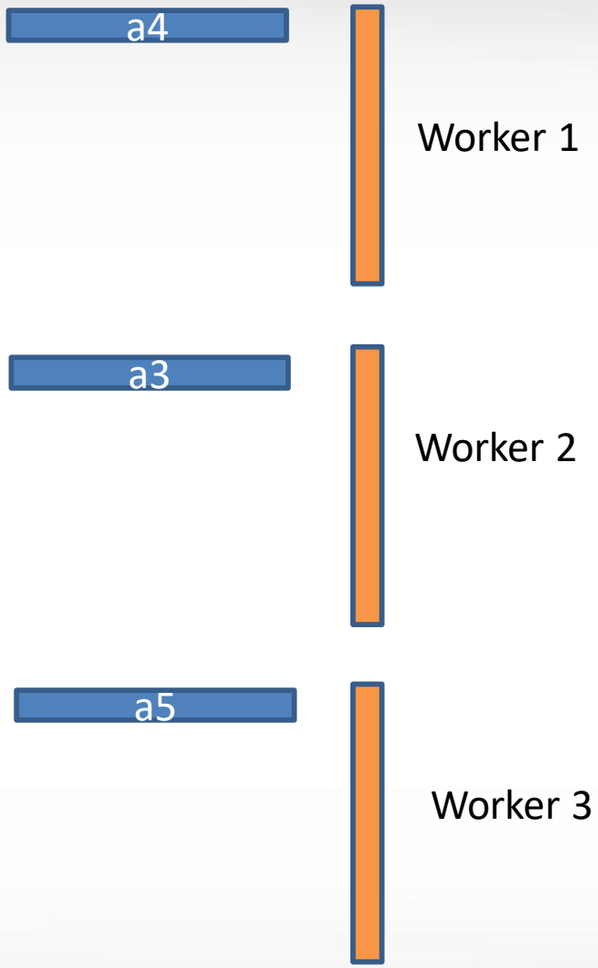



```
For i = 0 to n-1
   MPI_Recv(ans, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,
                                        MPI_COMM_WORLD, status)
   source      = status.MPI_SOURCE   3
   row_num     = status.MPI_TAG      2
   c(row_num) =  ans
   MPI_Send(a(index), n, MPI_DOUBLE, source, index, MPI_COM_WORLD)
```

```
MPI_Recv(a_row, n, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD)
row_num = status.MPI_TAG; do dot product;
MPI_Send(ans, 1, MPI_DOUBLE, 0, row_num, MPI_COMM_WORLD)
```

Example 7: Self-Scheduling






```
For i = 0 to n-1
   MPI_Recv(ans, 1, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,
                                        MPI_COMM_WORLD, status)
   source      = status.MPI_SOURCE
   row_num     = status.MPI_TAG
   c(row_num) = ans
   index = index+1
   if (index == n)
      MPI_Send(MPI_BOTTOM,0,MPI_DOUBLE,source,index,MPI_COMM_WORLD)
   else
      MPI_Send(a(index), n, MPI_DOUBLE, source, index, MPI_COMM_WORLD)
```

```
MPI_Recv(a_row, n, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD)
row_num = status.MPI_TAG
If (row_num < n)
   do dot product
   MPI_Send(ans, 1, MPI_DOUBLE, 0, row_num, MPI_COMM_WORLD)
```

Example 8: matvec-scatterv

$$A\vec{b} = (\vec{a}_1 \quad \ldots \quad \vec{a}_n)\vec{b} = b_1\vec{a}_1 + b_2\vec{a}_2 + \cdots + b_n\vec{a}_n = \vec{c}$$

$\vec{a}_i$ is a column vector.

In this program, strips of consecutive columns of A are distributed to all processes. Each process carries out a part of the linear vector sum

$$b_i\vec{a}_i + \cdots + b_j\vec{a}_j$$

Example 9: Solving the x-y Poisson Equation

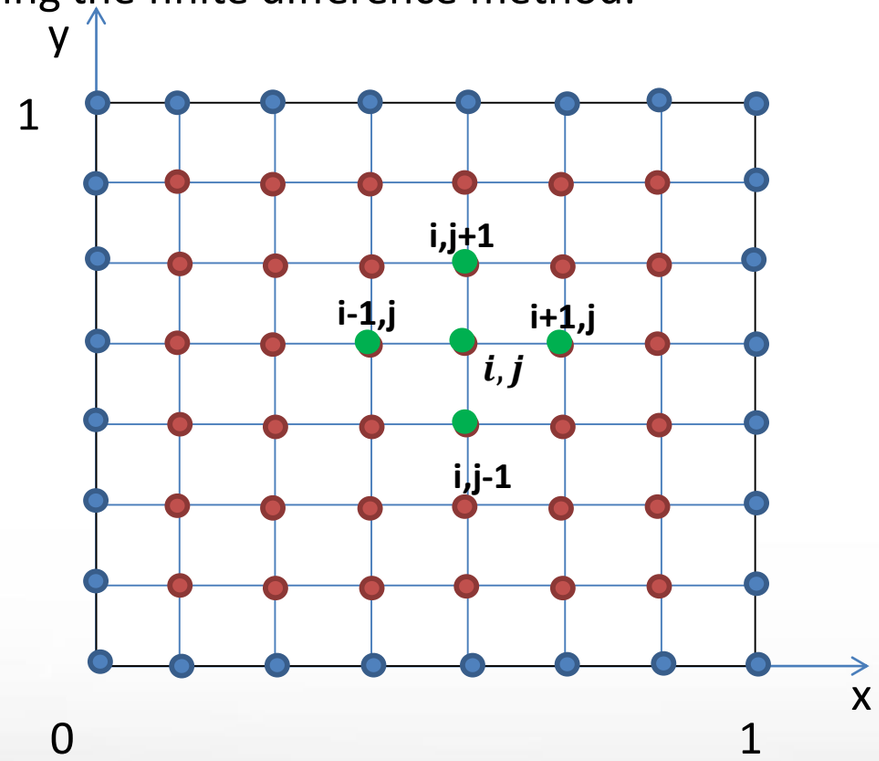
Solve the partial differential equation

$$\frac{\partial^2 u}{\partial^2 x} + \frac{\partial^2 u}{\partial^2 y} = f(x,y)$$

where $x, y \in [0,1]$

and $u = g(x,y)$ on boundary

using the finite difference method.



Discretize the domain along x and y using n internal points in each direction.

The increment is

$h = 1/(n+1)$

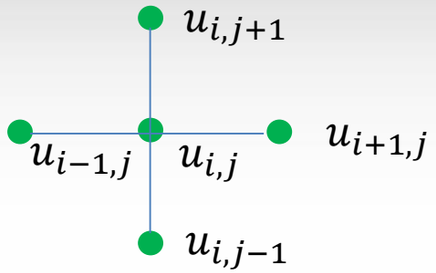
$x_i = ih, y_j = jh$

$(0 \le i, j \le n+1)$

$u_{ij} = u(x_i, y_j) = u(ih, jh)$

$(0 < i, j < n+1)$

Example 9: the x-y Poisson Equation



5-point finite difference stencil approximation:

$$u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j} = h^2 f_{i,j} \quad (0 < i, j < n+1)$$

$$u_{i,j} = 0.25 * (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - h^2 f_{i,j})$$

k+1 Jacobi iteration step at $x_i = ih;\ y_j = jh; i, j = 1{:}n$

$$u^{k+1}{}_{i,j} = 1/4(u^k{}_{i-1,j} + u^k{}_{i,j+1} + u^k{}_{i,j-1} + u^k{}_{i+1,j}) - h^2 f_{i,j}$$

Jacobi iteration across all points:

```
do j=1, n
   do i = 1, n
      unew(i, j) = 0.25*(u(i-1,j) + u(i, j+1) + u(i+1,j) + i(i, j-1)) – f(i,j)*h^2
   end do
end do
```

Example 9: Solving the x-y Poisson Equation

Boundary conditions (stay fixed):

$$u(x_i, 0) = \frac{\cos(\pi x_i) - \pi^2}{\pi^2} \qquad (0 \le x \le 1)$$

$$u(x_i, 1) = \frac{\cos(\pi x_i) - \pi^2 \cos(x_i)}{\pi^2} \qquad (0 \le x \le 1)$$

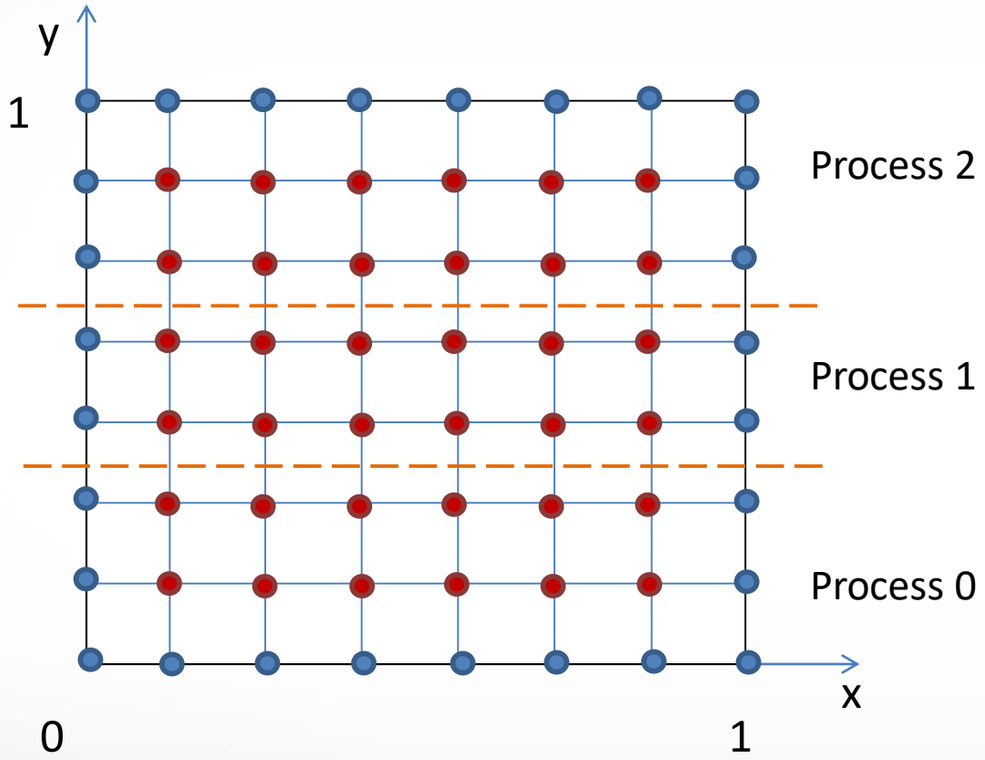
$$u(0, y_j) = \frac{1}{\pi^2} - 1 \qquad (0 \le y \le 1)$$

$$u(1, y_j) = -\left(\frac{1}{\pi^2} + \cos(y_j)\right) \qquad (0 \le y \le 1)$$

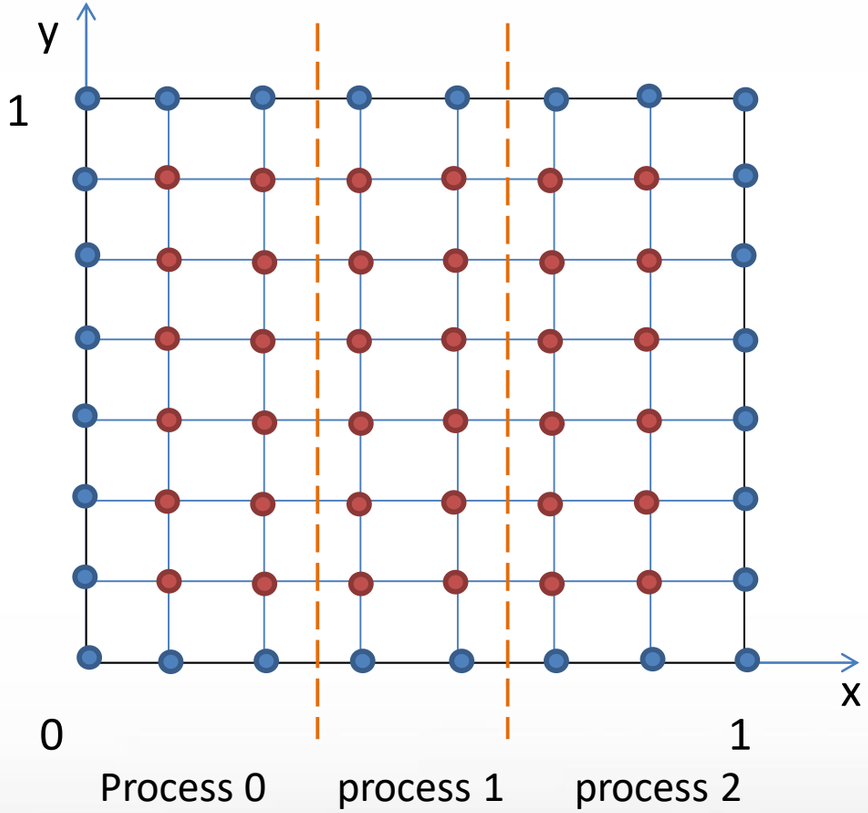
RHS: $f(x_i, y_j) = ({x_i}^2 + {y_j}^2)\cos(x_i y_j) - \cos(\pi x_i)$

1d-Domain Decomposition

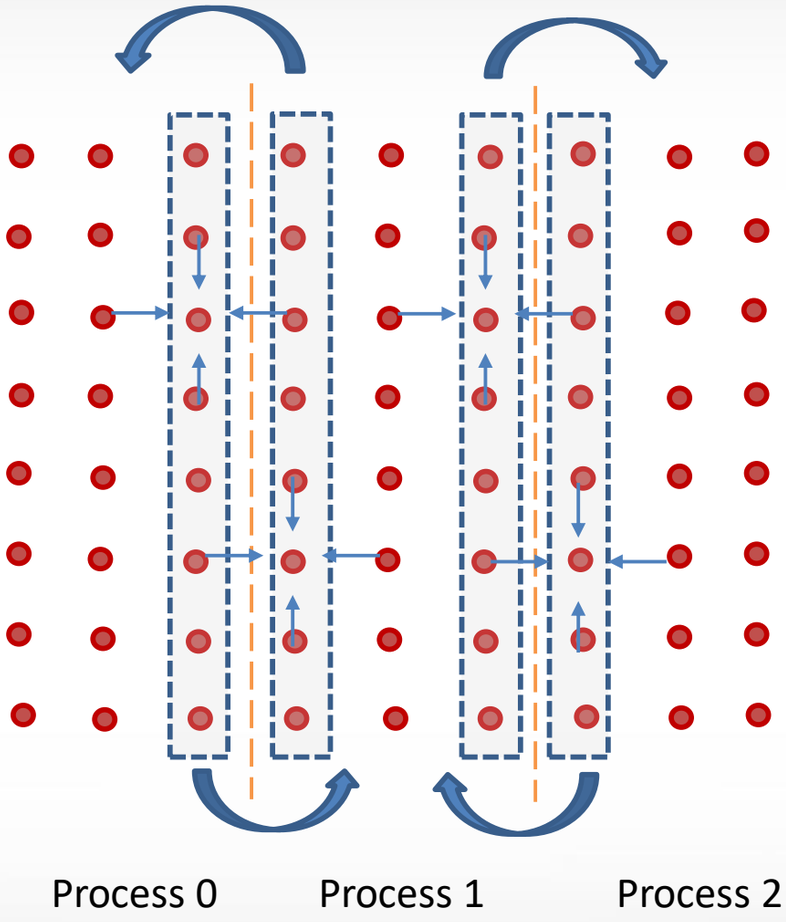
C: decompose along y axis



Fortran: decompose along x axis



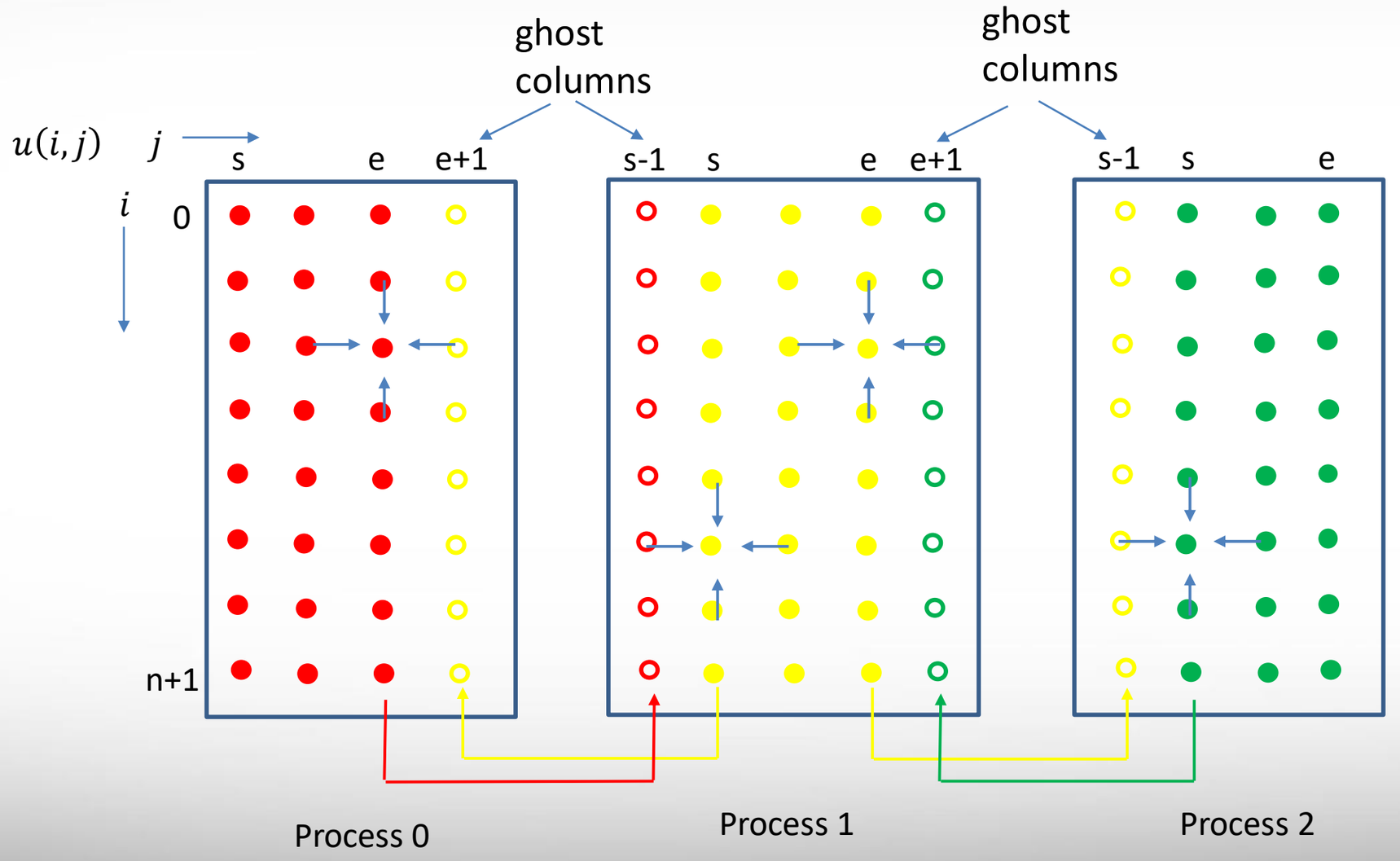
1d-Domain Decomposition



Border columns need to be copied into the memory space of the neighboring processes for the five-point stencil calculation. Same for row-wise decomposition.

1d-Domain Decomposition

Exchanged border columns are stored in ‘ghost’ columns in each process to be used in five-point stencil calculation



Example 10: Hybrid Programming

Int MPI_Init_thread(int *argc, char **argv, int required, int *provided)

MPI_INIT_THREAD(required, provided, ierr)

- Four possible values for the parameter required:
  - MPI_THREAD_SINGLE — ex2_single.c
  - MPI_THREAD_FUNNELD — ex2_funnel.c
  - MPI_THREAD_SERIALIZED — ex2_serialized.c
  - MPI_THREAD_MULTIPLE — ex2_multiple.c
- To compile

  mpiicc -qopenmp [options] prog.c –o prog.exe

  mpiifort –qopenmp [options] prog.f90 –o prog.exe

MPI+OMP Hybrid Programming

- Not all program will benefit from hybrid programming

poisson_1d performance comparison: pure MPI vs hybrid

	Pure MPI Single-node		Pure MPI 4-node		Hybrid OMP_NUM_THREADS=2		Hybrid OMP_NUM_THREADS=4
np=2	22.8697 seconds						
np=4	11.9441 seconds	np=4	13.5297 seconds	np=2	25.0954 seconds	np=1	49.7846 seconds
np=8	6.5824 seconds	np=8	7.7067 seconds	np=4	13.3360 seconds	np=2	25.0660 seconds
np=16	3.6141 seconds	np=16	4.6635 seconds	np=8	7.5272 seconds	np=4	13.3054 seconds

Hybrid is worse

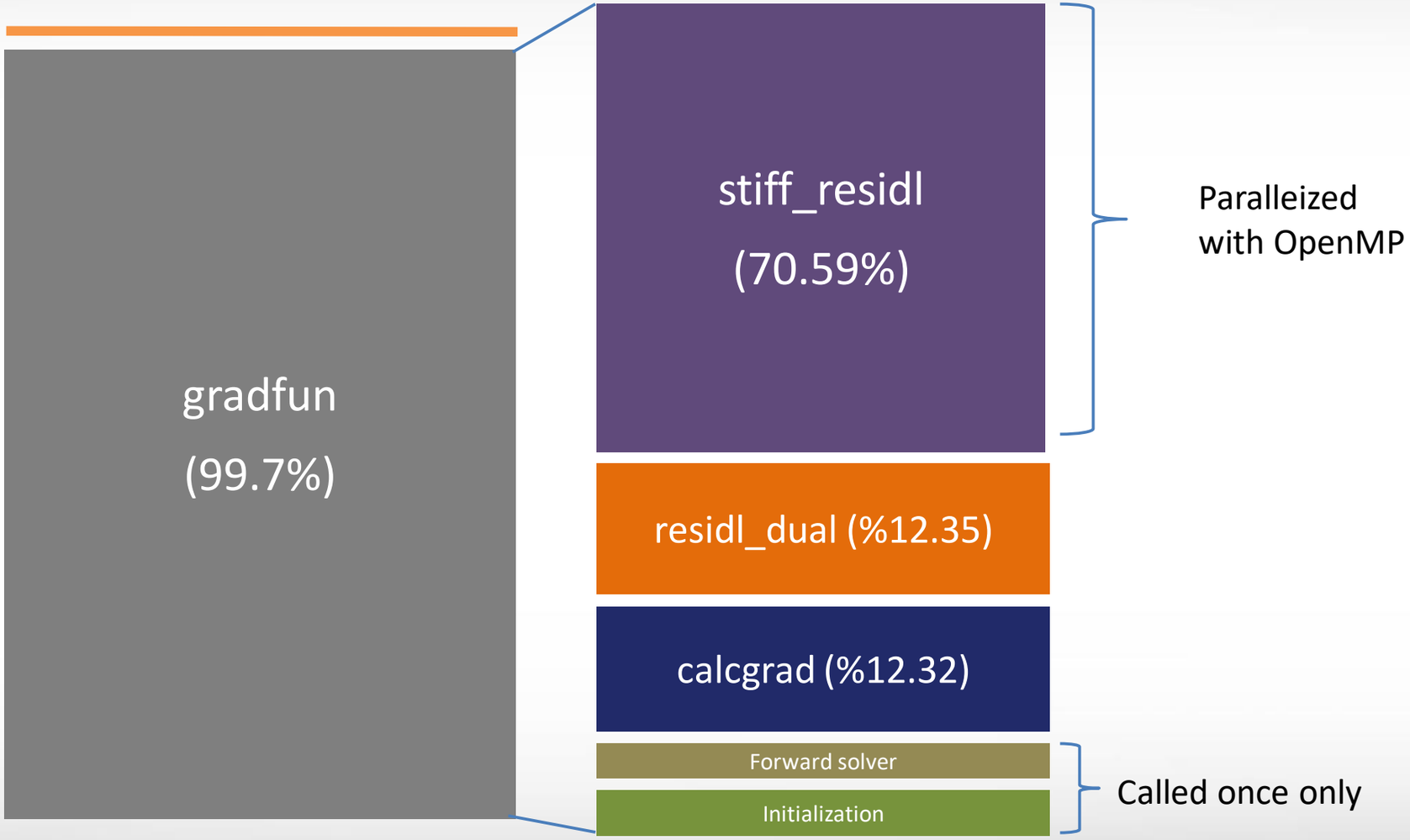
example9/poisson_1d.c (.f90)

example10-hybrid/poisson_1d_hybrid.c

MPI+OMP: A Case Study

- The nlace code developed by Prof. Sevan Goenezen's group from the TAMU ME department is used to estimate the non-homogeneous elastic material properties using force and surface displacement data from multiple measurements.
- Was partially parallelized with OpenMP
  - The inverse solver has a good speedup with up to 3 CPU cores. Increasing number of cores won't help to speedup the code further
- Cannot process 3D cases due to extremely slow running time.

Profiling


gradfun
(99.7%)
stiff_residl
(70.59%)
residl_dual (%12.35)
calcgrad (%12.32)
Forward solver
Initialization
Paralleized
with OpenMP
Called once only

Amdahl's Law

$$S_n = \frac{1}{\frac{p}{n} + (1 - p)}$$

S_n – speedup of the parallel code on n core vs the serial code

p – percentage of the code that can be parallelized

n – number of CPU cores used to run the code

According to Amdahl's law, S_n is bounded by the serial part of the code that cannot benefit from increasing the number of CPU cores.

The speedup of the original code cannot exceed 4 due to the 25% of serial code.

Results

- Parallelized the serial portion of the code with OpenMP
- Parallelized the entire code with MPI which has successfully explored the data parallelism among different measurements.

	Num of Cores	Wall Clock Time	Speedup
Original OpenMP	3	108m	1
Improved OpenMP	10	38m47s	2.8
OpenMP+MPI	100	3m20s	32.7

References

- A significant amount of content is adapted from former Supercomputing Facility staff Mr. Spiros Vellas' introductory MPI short course
- Two books: <<Using MPI>> and <<Using MPI II>>
- MPI standard: http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
- https://www.cac.cornell.edu/education/Training/parallelMay2011/Hybrid_Talk-110524.pdf