

Matrix Multiplication using MPI

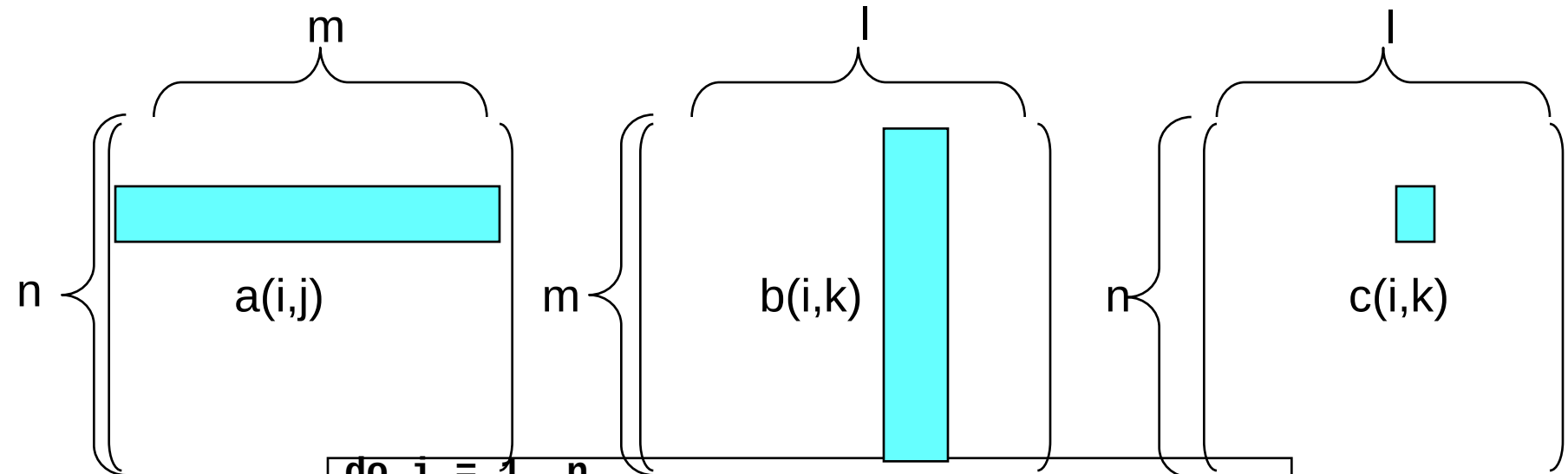
Dieter an Mey

*Center for Computing and Communication
Aachen University of Technology*

anmey@rz.rwth-aachen.de

Matrix Multiplication Serial Algorithm

$$A * B = C$$



```

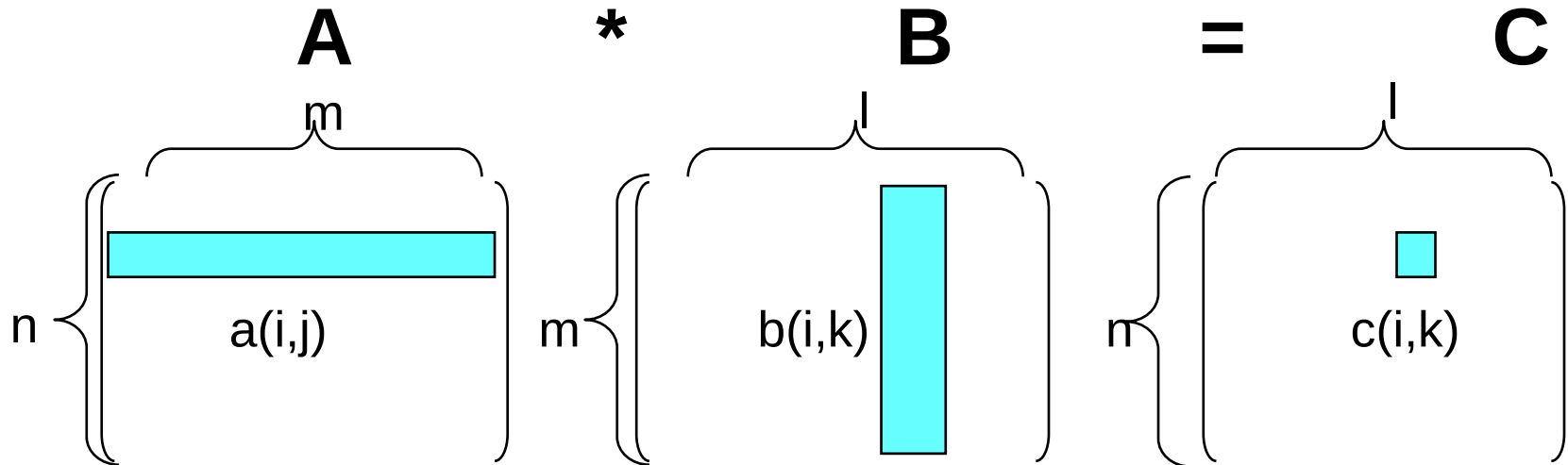
do i = 1, n
  do k = 1, l
    do j = 1, m
      c(i,k) = c(i,k) + a(i,j)*b(j,k)
    end do
  end do
end do

```

1. Serial Version
2. MPI Version 0.1, redundant calculation
3. MPI Version 0.2, redundant calculation, master only IO
4. MPI Version 1.0, redundant data, worksharing => Speed-up, still wasting memory
5. MPI Version 2.0, storing A redundantly, splitting matrices B and C => reduction of memory consumption
6. MPI Version 3.0, splitting matrix A, stepwise cyclically shifting of A.
7. MPI Version 4.0, overlapping communication (cyclically shifting of A) and computation.
8. MPI Version 5.0 like 4.0 with One-Sided Communication
9. Master-Slave Version

Matrix Multiplikation

Library Function (Example: sunperf)



gemm - perform one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$

SUBROUTINE GEMM([TRANSA],[TRANSB],[M],[N],[K],ALPHA,A,[LDA],B,[LDB], BETA, C,[LDC])

Fortran90: call gemm (alpha=1.0d0, a=a, b=b, beta=0.0d0, c=c)

C: dgemm('T', 'T', m, n, k, 1.0, matrixA, k, matrixB, n, 0.0, matrixC, m);

```

module mxmdat
  implicit none
  ! a(n,m) x b(m,l) = c(n,l)
  integer :: n,m,l
  real*8, allocatable, dimension(:,:) :: a, b, c
end module mxmdat

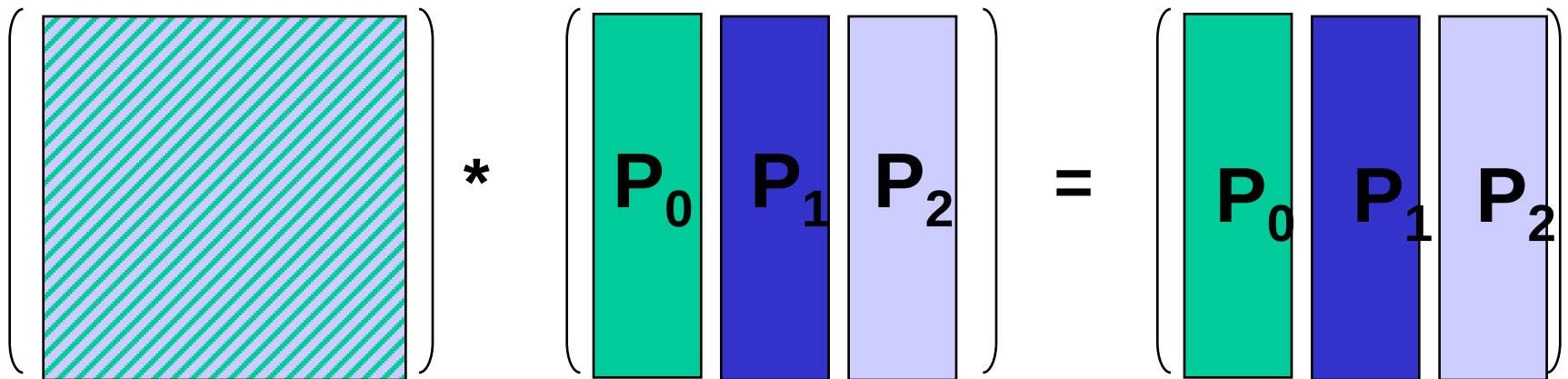
program mxm
  use mxmdat
  use sunperf
  call mat_dimensions      ! Read matrix dimensions
  call mat_alloc           ! Allocate matrices A, B, C
  call mat_input           ! Generate/read/initialize A, B, C
  call gemm ( alpha=1.0d0, a=a, b=b, beta=0.0d0, c=c )
                           ! Matrix multiplication by
  sunperf
  call mat_output          ! Print matrices A, B, C
end program mxm

```

Matrix Multiplication

First Parallel Algorithm

$$A * (B_0 | B_1 | B_2) = (A*B_0 | A*B_1 | A*B_2) \\ = (C_0 | C_1 | C_2)$$



A copied to all processors P_x

```

module mpicontrol
  include 'mpif.h'
  integer*4 :: myrank, nprocs, ierror, length
  integer*4 :: status(MPI_STATUS_SIZE)
  character*255 :: hostname
end module mpicontrol

...

program mxm
  use mpicontrol
  ...
  call initialize_mpi          ! Initialize MPI
  ...
  call MPI_Finalize (ierror)  ! Terminate MPI
end program mxm

```

```

subroutine initialize_mpi
  use mpicontrol
  character*80 :: string
  call MPI_Init (ierror)
  call MPI_Comm_size (MPI_Comm_world, nprocs, ierror)
  call MPI_Comm_rank (MPI_Comm_world, myrank, ierror)
  call MPI_Get_processor_name ( hostname, length, ierror )
  length = index(hostname, " ") - 1
  write(string,*) myrank, nprocs, trim(hostname(1:length))
  call print_ordered ( string )
end subroutine initialize_mpi

subroutine print_ordered ( string )

  call MPI_Barrier (MPI_COMM_WORLD, ierror)
  do i = 0, nprocs-1
    if ( i == myrank ) write (*,*) string
    call MPI_Barrier (MPI_COMM_WORLD, ierror)
  end do
end subroutine print_ordered

```

```

subroutine print_ordered ( string )      ! alternate version
  use mpicontrol
  character*(*) :: string
  integer*4 :: token

  call MPI_Barrier (MPI_COMM_WORLD, ierror)
  if ( myrank == 0 ) then
    token = 0
  else
    call MPI_Recv ( token, 1, MPI_INTEGER, myrank-1, ... )
  end if
  if ( token /= myrank ) write(*,*) "Error"
  write (*,*) trim(string)                ! ordered print
  ierror = flush(6)
  token = token + 1
  if ( token < nprocs ) then
    call MPI_Send ( token, 1, MPI_INTEGER, token, ... )
  end if
  call MPI_Barrier (MPI_COMM_WORLD, ierror)
end subroutine print_ordered

```

```

subroutine mat_input
  use mxmdat
  implicit none
  integer*4 :: i
  if ( myrank == 0 ) then
    do i = 1, n
      a(i,1:m) = i
    end do
    do i = 1, 1
      b(1:m,i) = i
    end do
  end if
  call MPI_Bcast (a(1,1),size(a),MPI_DOUBLE_PRECISION,0,...)
  call MPI_Bcast (b(1,1),size(b),MPI_DOUBLE_PRECISION,0,...)
  c = 0.0d0
end subroutine mat_input

```

```

subroutine mat_output
  use mxmdat
  implicit none
  integer*4 :: istat,i,j
  if ( myrank == 0 ) then
    do i = 0, nprocs-1
      if (i>0) then
        call MPI_Recv (c(1,1),size(c),MPI_DOUBLE_PRECISION,i,...)
      end if
      write (*,*) "Matrix C: (rank=", i, ")"
      do j = 1, n
        write (*,*) c(j,1:1)
      end do
    end do
  else
    call MPI_Send (c(1,1),size(c),MPI_DOUBLE_PRECISION,0,...)
  end if
end subroutine mat_output

```

```

program mxm
  ..
  character*132 :: string
  integer*4 :: ilo, ihi

  call initialize_mpi           ! Initialize MPI
  call mat_dimensions          ! Read matrix dimensions
  call mat_alloc               ! Allocate matrices A, B, and C
  call mat_input               ! Generate/read/Initialize ...

  call workshare ( myrank, nprocs, 1, ilo, ihi )
  write (string,*) myrank, "work:",ilo,ihl,"chunk:",ihl-ilo+1
  call print_ordered(string)

  call gemm ( alpha=1.0d0, a=a(1:n,1:m), b=b(1:m,ilo:ihl), &
             beta=0.0d0, c=c(1:n,ilo:ihl) ) ! Matrix multiplication

  call mat_output              ! Print matrices A, B, and C
  call MPI_Finalize (ierror)
end program mxm

```

```
subroutine workshare ( rank, nprocs, n, ilo, ihi )  
  ! (1:n) is partitioned into nprocs pieces.  
  ! task <rank> gets (ilo:ihi)  
  integer*4, intent(in) :: rank, nprocs, n  
  integer*4, intent(out) :: ilo, ihi  
  integer*4 :: nrem, nchunk, token  
  
  nrem = mod ( n, nprocs )  
  nchunk = ( n - nrem ) / nprocs  
  if ( rank < nrem ) then  
    ilo = 1 + rank * ( nchunk + 1 )  
    ihi = ilo + nchunk  
  else  
    ilo = 1 + rank * nchunk + nrem  
    ihi = ilo + nchunk - 1  
  end if  
end subroutine workshare
```

```

subroutine mat_output
  use mpicontrol
  use mxmdat
  integer*4 :: istat,i,j
  if ( myrank == 0 ) then
    write (*,*) "Matrix C:"
    do i = 1, nprocs-1
      call workshare ( i, nprocs, 1, ilo, ihi )
      call MPI_Recv (c(1:n,ilo:ihi),n*(ihi-ilo+1),MPI_D...,i,...)
    end do
    do j = 1, n
      write (*,*) c(j,1:1)
    end do
  else ! slaves
    call workshare ( myrank, nprocs, 1, ilo, ihi )
    call MPI_Send (c(1:n,ilo:ihi),n*(ihi-ilo+1),MPI_D...,0,...)
  end if
end subroutine mat_output

```

```

program mxm
  use mxmdat
  use sunperf
  use mpicontrol
  character*132 :: string
  integer*4 :: ilo, ihi

  call initialize_mpi           ! Initialize MPI
  call mat_dimensions           ! Read matrix dimensions

  call workshare ( myrank, nprocs, 1, ilo, ihi )
  call mat_alloc ( ilo, ihi ) ! Allocate matrices A, B, C
  call mat_input ( ilo, ihi ) ! Generate/read/initialize A,B,C

  call gemm ( alpha=1.0d0, a=a, b=b, beta=0.0d0, c=c )

  call mat_output ( ilo, ihi ) ! Print matrices A, B, and C
  call MPI_Finalize (ierror)
end program mxm

```

```
subroutine mat_alloc ( ilo, ihi )
  use mxmdat
  integer*4, intent(in) :: ilo, ihi
  integer*4 :: istat
  allocate ( a(n,m), b(m,ilo:ihi), c(n,ilo:ihi), stat=istat )
end subroutine mat_alloc
```

! Each MPI process initializes the full matrix A redundantly and its chunk of B and C

```
subroutine mat_input ( ilo, ihi )
  do i = 1, n
    a(i,1:m) = i
  end do
  do i = ilo, ihi
    b(1:m,i) = i
  end do
  c(1:n,ilo:ihi) = 0.0d0
end subroutine mat_input
```

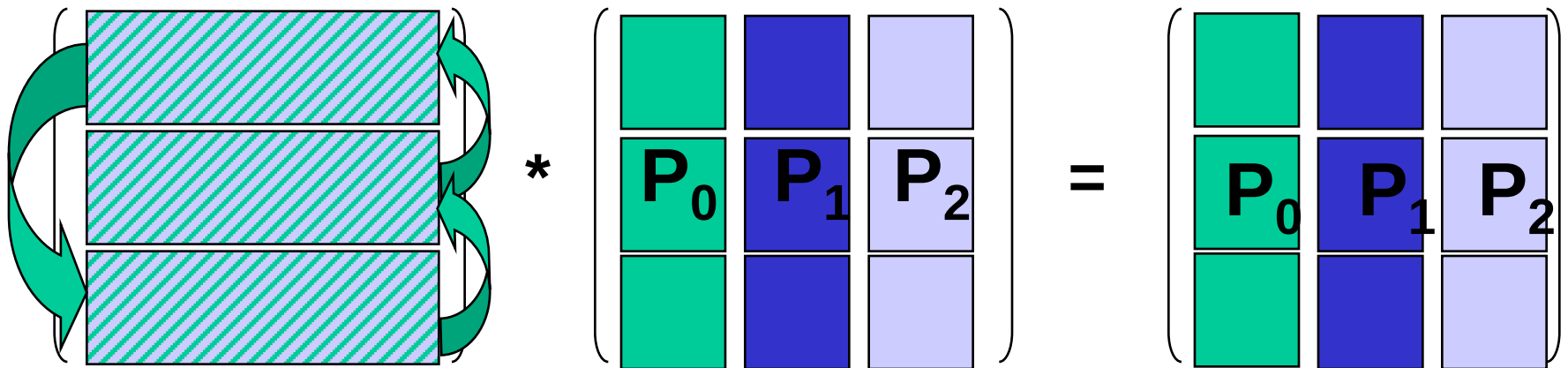
! *Each MPI prints its chunk of B and C*

```
subroutine mat_output ( ilo, ihi )  
  use mpicontrol  
  use mxmdat  
  implicit none  
  integer*4, intent(in) :: ilo, ihi  
  integer*4 :: istat,i,j  
  
  call MPI_Barrier (MPI_COMM_WORLD, ierror)  
  do i = 0, nprocs-1  
    if ( i == myrank ) then  
      write (*,*) "Matrix C: (chunk ", myrank, ")"  
      do j = 1, n  
        write (*,*) c(j,ilo:ihi)  
      end do  
    end if  
    call MPI_Barrier (MPI_COMM_WORLD, ierror)  
  end do  
  
end subroutine mat_output
```

Matrix Multiplication

Second parallel Algorithm

$$A * (B_0 \mid B_1 \mid B_2) = (A*B_0 \mid A*B_1 \mid A*B_2) \\ = (C_0 \mid C_1 \mid C_2)$$



Stripes are shifted cyclically at each step

```
program mxm
```

```
  call initialize_mpi
  call mat_dimensions
```

```
  call workshare ( myrank, nprocs, 1, ilo, ihi )
```

```
  call mat_alloc ( ilo, ihi )
  call mat_input ( ilo, ihi )
```

```
  do istep = 0, nprocs-1
```

```
    ! calculate the chunk of rows of A for MPI process myrank in step istep
```

```
    call workshare ( mod(myrank+istep, nprocs), nprocs, n, jlo, jhi )
```

```
    ! Work on slice 0
```

```
    call gemm (alpha=1.0d0, a=a(:, :, 0), b=b, beta=0.0d0, &
                                     c=c(jlo:jhi, ilo:ihi))
```

```
    if(istep < nprocs-1) call mat_shift
  end do
```

```
  call mat_output ( ilo, ihi )
  call MPI_Finalize (ierror)
```

```
end program mxm
```

```
module mxmdat
```

```
  real*8, allocatable, dimension(:, :, :) :: a
  real*8, allocatable, dimension(:, :) :: b, c
end module mxmdat
```

For simplicity, n is a divisor of the processors count.

```
subroutine mat_alloc ( ilo, ihi )
  allocate (a(1:n/nprocs,m,2),b(m,ilo:ihi), &
            c(n,ilo:ihi),stat=istat )
end subroutine mat_alloc
```

```

subroutine mat_shift
  use mxmdat
  use mpicontrol
  integer*4 :: sendreq, recvreq, req(2)

  ! Send slice 0
  call MPI_Isend ( a(1,1,0), size(a(:, :, 0)), &
    MPI_DOUBLE_PRECISION, mod(myrank-1+nprocs,nprocs), &
    13, MPI_COMM_WORLD, sendreq, ierror )

  ! Receive slice 0
  call MPI_Irecv ( a(1,1,1), size(a(:, :, 0)), &
    MPI_DOUBLE_PRECISION, mod(myrank+1,nprocs), &
    13, MPI_COMM_WORLD, recvreq, ierror )
  req(1) = sendreq
  req(2) = recvreq

  call MPI_Waitall ( 2, req, MPI_STATUS_IGNORE, ierror )
  call MPI_Barrier (MPI_COMM_WORLD, ierror)

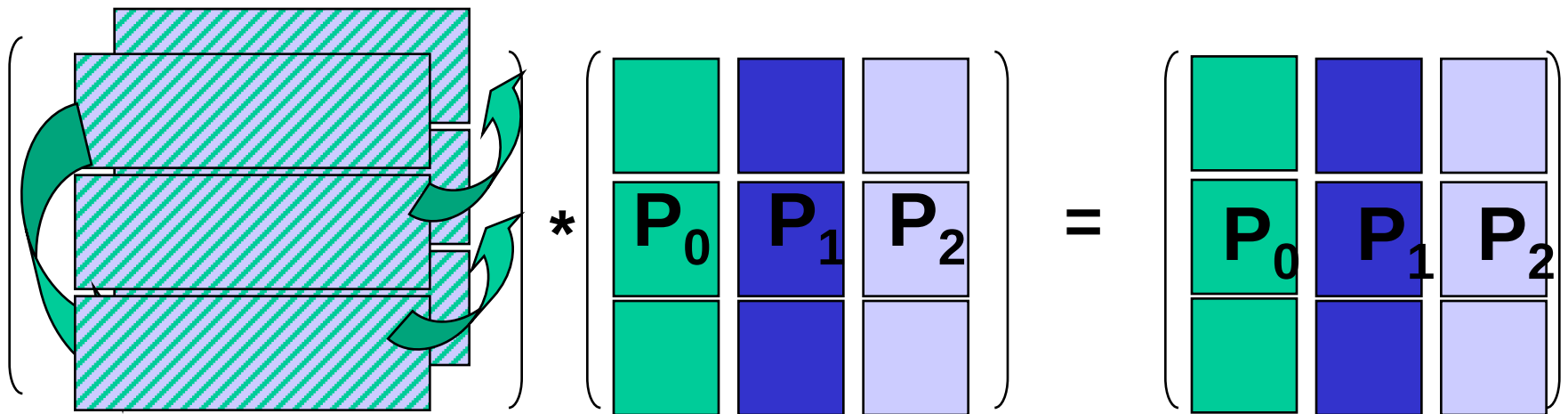
  ! Copy slice 1 to slice 0
  a(:, :, 0) = a(:, :, 1)
end subroutine mat_shift

```

Matrix Multiplication

Third parallel Algorithm

$$A * (B_0 | B_1 | B_2) = (A*B_0 | A*B_1 | A*B_2) \\ = (C_0 | C_1 | C_2)$$



Stripes are shifted cyclically at each step

By alternatingly employing two stripes of A
communication and computation can be overlapped.

```

program mxm
  call initialize_mpi
  call mat_dimensions
  call workshare ( myrank, nprocs, 1, ilo, ihi )
  call mat_alloc ( ilo, ihi )
  call mat_input ( ilo, ihi )

  do istep = 0, nprocs-1
    toggle = mod(istep,2)
    call workshare(mod(myrank+istep, nprocs),nprocs,n,jlo,jhi)
    if ( istep<nprocs-1 ) call mat_shift_start ( toggle, req )
! Work on slice toggle
    call gemm ( alpha=1.0d0, a=a(:, :, toggle), &
               b=b, beta=0.0d0, c=c(jlo:jhi, ilo:ihi) )
    if ( istep < nprocs-1 ) call mat_shift_end ( toggle, req )
  end do

  call mat_output ( ilo, ihi, istep )
  call MPI_Finalize (ierror)
end program mxm

```

```

subroutine mat_shift_start ( toggle, req )

  integer*4 , intent(in) :: toggle
  integer*4 :: sendreq, recvreq, req(2)
! Work and send slice toggle
  call MPI_Isend ( a(1,1,toggle), size(a(:, :, 0)), ... )
! Receive slice (1-toggle)
  call MPI_Irecv ( a(1,1,1-toggle), size(a(:, :, 0)), ... )
  req(1) = sendreq
  req(2) = recvreq
end subroutine mat_shift_start

subroutine mat_shift_end ( toggle , req )

  integer*4 , intent(in) :: toggle
  integer*4 :: sendreq, recvreq , req(2)

  call MPI_Waitall ( 2, req, MPI_STATUS_IGNORE, ierror )
end subroutine mat_shift_end

```

```

subroutine mat_alloc ( ilo, ihi, win )
  allocate (a(1:n/nprocs,m,0:1),b(m,ilo:ihi),c(n,ilo:ihi),...)
  call MPI_Win_create ( a, 2*8*m*n/nprocs, 8, info, &
                        MPI_COMM_WORLD, win, ierror )
end subroutine mat_alloc

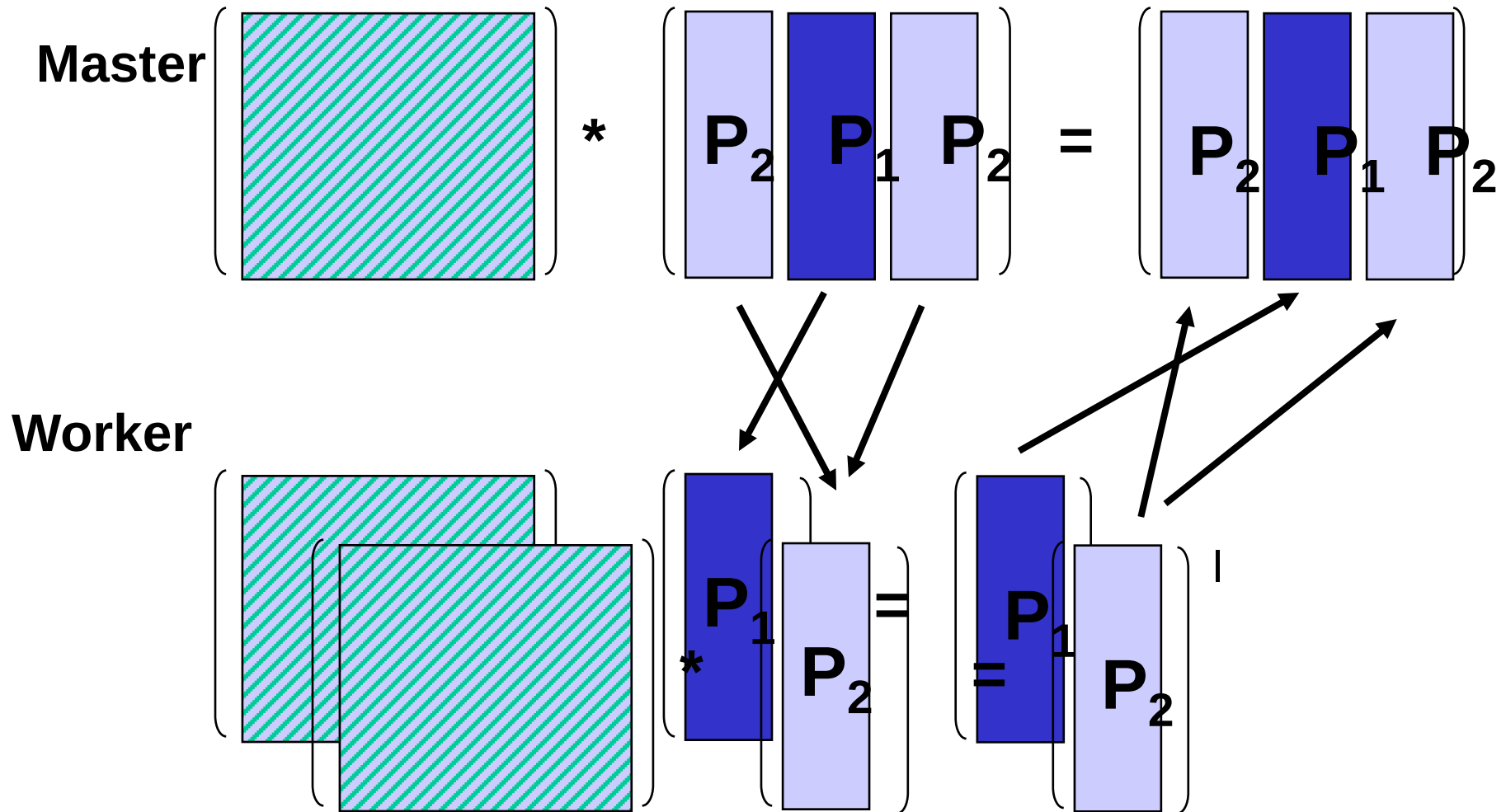
subroutine mat_shift_start ( toggle, win )
  call MPI_Win_fence ( 0, win, ierror ) ! Synchronisation
  call MPI_Put ( a(1,1,toggle), size(a(:, :, toggle)), &
                MPI_DOUBLE_PRECISION, mod(myrank-1+nprocs,nprocs), &
                (1-toggle)*m*n/nprocs, size(a(:, :, toggle)), &
                MPI_DOUBLE_PRECISION, win, ierror )
end subroutine mat_shift_start

subroutine mat_shift_end ( toggle, win )
  call MPI_Win_fence ( 0, win, ierror )
end subroutine mat_shift_end

```

Matrix Multiplication

Master-Worker Concept



```

program mxm

  use mxmdat
  use sunperf

  implicit none
  integer*4 :: i

  call mat_dimensions      ! Read matrix dimensions
  call mat_alloc           ! Allocate matrices A, B, C
  call mat_input           ! Generate/read/initialize A,B,C

  do i = 1, 1
    call gemm ( alpha=1.0d0, a=a, b=b(:,i:i), beta=0.0d0, &
               c=c(:,i:i) ) ! Matrix-Vector Multiplication
  end do

  call mat_output          ! Print matrices A, B, and C
end program mxm

```

```

subroutine mat_dimensions
  implicit none
  integer*4 :: eins
  if ( myrank == 0 ) then                                ! master only
    write (*,*) "Multiplication of 2 Matrices: &
                  a(n,m) x b(m,l) = c(n,l)"
    write (*,"(a)") "Input of matrix dimensions n,m,l :"
    read (*,*) n,m,l
  else
    l = 1
  end if
  call MPI_Bcast (n,1,MPI_INTEGER,0,MPI_COMM_WORLD, ierror )
  call MPI_Bcast (m,1,MPI_INTEGER,0,MPI_COMM_WORLD, ierror )

end subroutine mat_dimensions

```

```
program mxm
```

```

  ..
  call initialize_mpi           ! Initialize MPI
  call mat_dimensions          ! Read matrix dimensions
  call mat_alloc               ! Allocate matrices A, B, C
  call mat_input               ! Generate/read/initialize A,B,C

  if ( myrank == 0 ) then
    if ( nprocs == 1 ) then
      call gemm ( alpha=1.0d0, a=a, b=b, beta=0.0d0, c=c )
    else
      call master
    end if
    call mat_output            ! Print matrices A, B, and C
  else
    call worker
  end if

  call MPI_Finalize ( ierror )
end program mxm

```

```

subroutine worker
  use mxmdat
  use mpicontrol
  implicit none
  do
    ! wait_for_work
    call MPI_Recv ( b(1,1), n, MPI_DOUBLE_PRECISION, &
                   0, MPI_ANY_TAG, MPI_COMM_WORLD, status, ierror )
    if ( status(MPI_TAG) == READY_TAG ) then
      exit
    end if
    ! do_work
    call gemm ( alpha=1.0d0, a=a, b=b, beta=0.0d0, c=c )
    ! send_results
    call MPI_Send ( c(1,1), n, MPI_DOUBLE_PRECISION, &
                   0, RESULT_TAG, MPI_COMM_WORLD, ierror )
  end do
end subroutine worker

```

```
subroutine master
  use mxmdat
  use mpicontrol
  implicit none
  integer*4 :: i, worker
  do i = 1, 1  ! First and stupid solution, just to test slave code first
    worker = mod(i,nprocs-1) + 1
    call MPI_Send ( b(1,i), n, MPI_DOUBLE_PRECISION, &
                   worker, WORK_TAG, MPI_COMM_WORLD, ierror )
    call MPI_Recv ( c(1,i), n, MPI_DOUBLE_PRECISION, &
                   worker, RESULT_TAG, MPI_COMM_WORLD, status,ierror)
  end do
  do i = 1, nprocs-1
    call MPI_Send ( READY_TAG, 0, MPI_DOUBLE_PRECISION, &
                   i, READY_TAG, MPI_COMM_WORLD, ierror )
  end do
end subroutine master
```

```
subroutine master
```

```

integer*4, allocatable, dimension(:) :: active_jobs
allocate ( active_jobs(nprocs-1), stat=istat )
job = 0
active_jobs_counter = 0
do worker = 1, nprocs-1  ! Initial phase
    job = job + 1
    if ( job > 1 ) then ! More workers than matrix columns
        active_jobs(worker) = -1
        call MPI_Send ( READY_TAG, 0, MPI_DOUBLE_PRECISION, &
            worker, READY_TAG, MPI_COMM_WORLD, ierror )
    else
        call MPI_Send ( b(1,job), n, MPI_DOUBLE_PRECISION, &
            worker, WORK_TAG, MPI_COMM_WORLD, ierror )
        active_jobs(worker) = job ! Additional book-keeping
        active_jobs_counter = active_jobs_counter + 1
    end if
end do
...
```

```

do
    ! Final phase
    call MPI_Probe ( MPI_ANY_SOURCE, RESULT_TAG, &
                     MPI_COMM_WORLD, status, ierror )
    worker = status(MPI_SOURCE)
    call MPI_Recv ( c(1:n,active_jobs(worker)),n, &
                     MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE, RESULT_TAG,
&
                     MPI_COMM_WORLD,status,ierror)
    active_jobs_counter = active_jobs_counter - 1
    job = job + 1
    if ( job > 1 ) then ! No more work
        active_jobs(worker) = -1
        call MPI_Send ( READY_TAG, 0, MPI_DOUBLE_PRECISION, &
                        worker, READY_TAG, MPI_COMM_WORLD, ierror )
        if ( active_jobs_counter == 0 ) exit
    else ! Still more work to do
        call MPI_Send ( b(1,job), n, MPI_DOUBLE_PRECISION, &
                        worker, WORK_TAG, MPI_COMM_WORLD, ierror )
        active_jobs(worker) = job
        active_jobs_counter = active_jobs_counter + 1
    end if
end do
end subroutine master

```