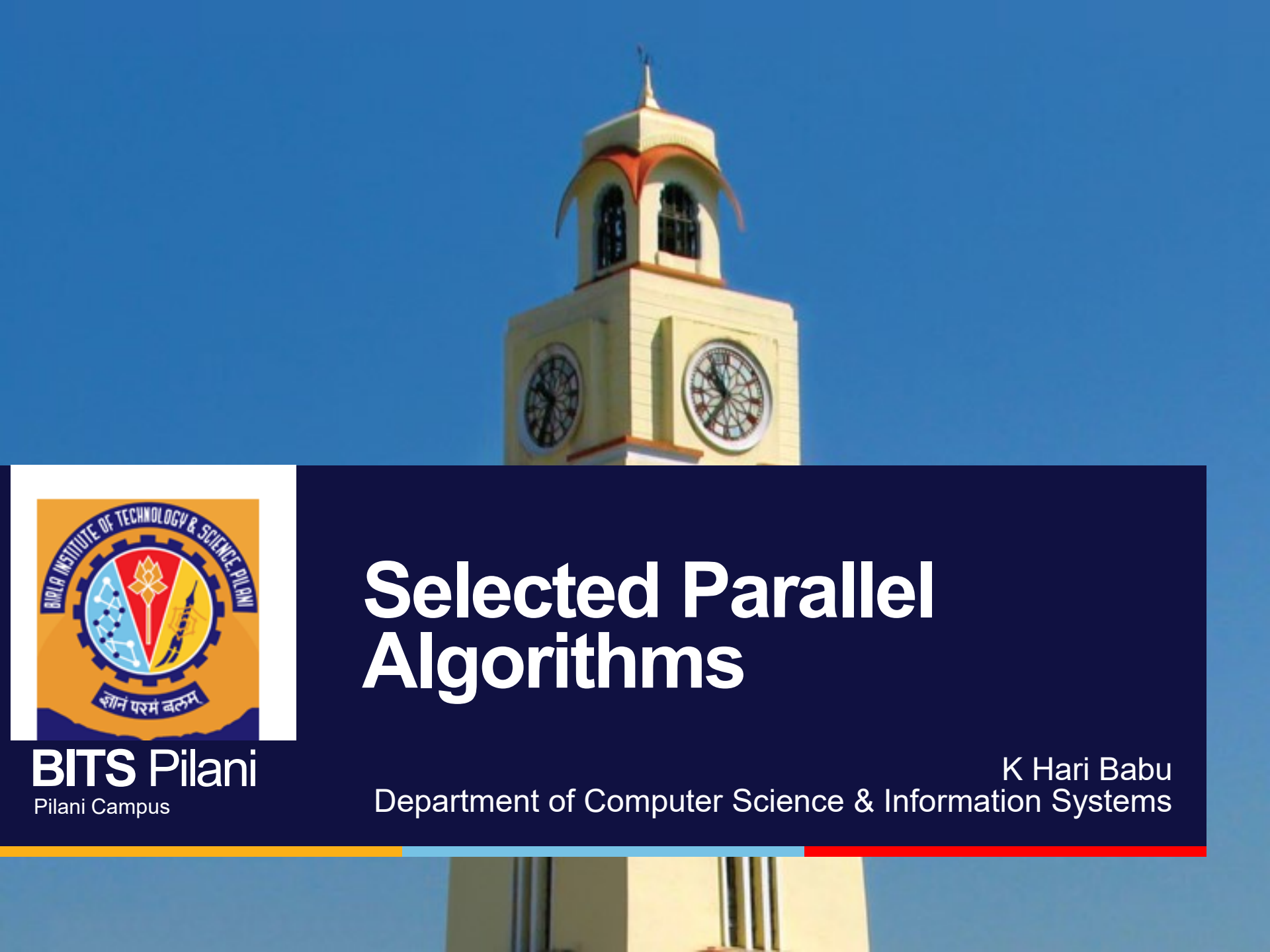


Selected Parallel Algorithms

BITS Pilani
Pilani Campus

K Hari Babu
Department of Computer Science & Information Systems

innovate achieve lead

BITS Pilani
Pilani Campus

Matrix-vector Multiplication

Matrix-vector Multiplication

- Usage
  - Many iterative algorithms for solving systems of linear equations rely upon matrix -vector multiplication
    - E.g. The conjugate gradient method
  - in implementation of neural networks
- Sequential Algorithm

Input: $a[0..m-1, 0..n-1]$ — matrix with dimensions $m \times n$
$b[0..n-1]$ — vector with dimensions $n \times 1$

Output: $c[0..m-1]$ — vector with dimensions $m \times 1$

for $i \leftarrow 0$ to $m-1$

$$\begin{bmatrix} 3 & 1 & 0 & 4 & 2 & -1 \\ 0 & 1 & -1 & 5 & -2 & 3 \\ 1 & 0 & 2 & 3 & 1 & 0 \\ 4 & 2 & -1 & -1 & 0 & -3 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 2 \\ 4 \\ 1 \\ -2 \end{bmatrix} = \begin{bmatrix} 23 \\ 10 \\ 18 \\ 4 \end{bmatrix}$$

$$(3 \cdot 1) + (1 \cdot 0) + (0 \cdot 2) + (4 \cdot 4) + (2 \cdot 1) + ((-1) \cdot (-2)) = 23$$

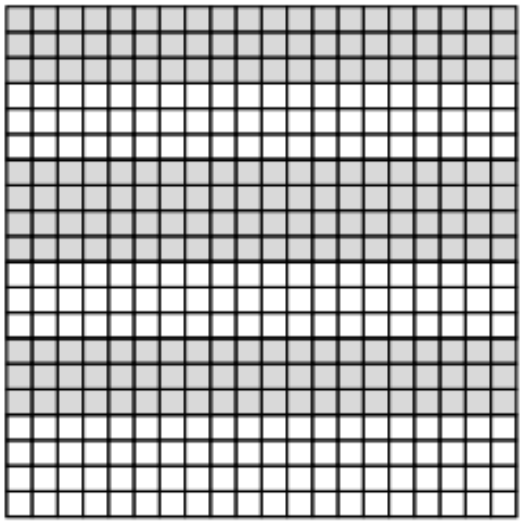
$$(0 \cdot 1) + (1 \cdot 0) + (-1 \cdot 2) + (5 \cdot 4) + (-2 \cdot 1) + (3 \cdot (-2)) = 10$$

$$(1 \cdot 1) + (0 \cdot 0) + (2 \cdot 2) + (3 \cdot 4) + (1 \cdot 1) + (0 \cdot (-2)) = 18$$

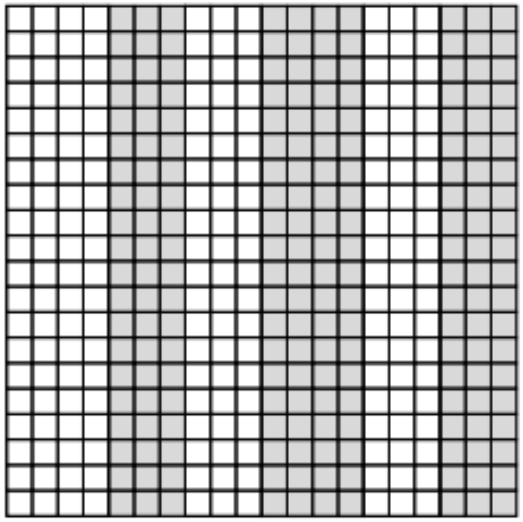
$$(4 \cdot 1) + (2 \cdot 0) + (-1 \cdot 2) + (-1 \cdot 4) + (0 \cdot 1) + (3 \cdot (-2)) = 4$$

Decomposition

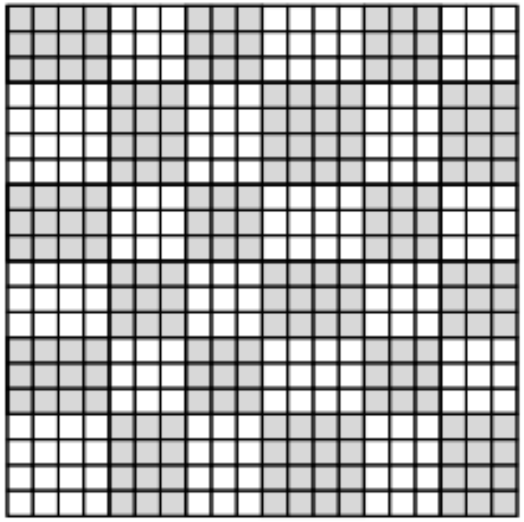
- Use the domain decomposition strategy to develop parallel algorithms. There are a variety of ways to partition, agglomerate and map the matrix and vector elements
- Each data decomposition results in a different parallel algorithm
- Three ways to decompose a two-dimensional matrix



(a) Row-wise block-striped matrix decomposition.



(b) Column-wise block-striped matrix decomposition.

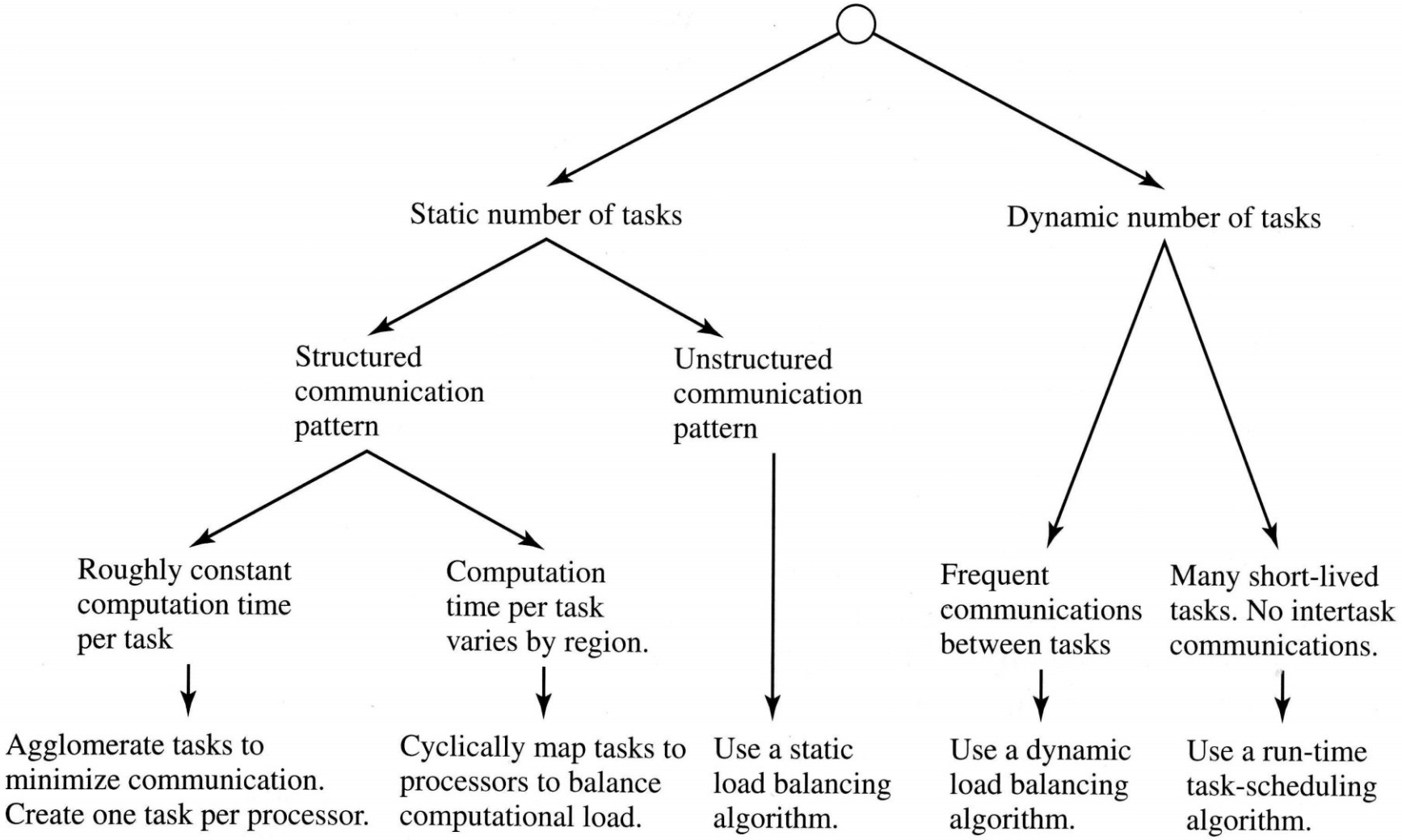


(c) Checkerboard block decomposition.

Decomposition

- There are two natural ways to distribute vectors b and c
  - The vector elements may be replicated, meaning all the vector elements are copied on all of the tasks, or the vector elements may be divided among some or all of the tasks.
  - In a block decomposition of an n-element vector, each of the p processes is responsible for a contiguous group of n/p vector elements
- With three ways to decompose the matrix and two ways to distribute the vector, six parallel algorithms are possible

Guidelines for Agglomeration & Mapping



Decompositions

	Row distribution	Vector distribution	Communication	Agglomoration	Mapping
Rowwise block-striped decomposition	Task i has row i	Replicated among all tasks	Allgather	Constant computation per task. p tasks. n/p rows for a task	One task for each processor.
Columnwise block striped decomposition	Task i has column i	Element i of vectors b and c	All to all exchange or all to all personalized communication	Constant computation per task. p tasks. n/p columns for a task	One task for each processor.
Checkerboard block decomposition	Task i has one elemnt	distributed	Send, Broadcast	each task is responsible tor a block of matrix A	One task for each processor.

Rowwise block-striped decomposition

- Communication complexity
  - Allgather communication (all to all broadcast)
  - n/p values in each message
  - Considering hypercube topology, $t_s(logp) + t_w(n/p)(p-1)$. When p is large, $t_s(logp) + t_w(n)$
- Computation complexity
  - Each row has n values. Vector has n values. *n* multiplications and additions.
  - There are n/p rows. Total = n/p * n = n^2/p
- Parallel run time
  - $T_p = \frac{n^2}{p} + t_s logp + t_w n$

Algo	Ring	2D Mesh	Hypercube
All-to-all broadcast	$(t_s + t_w m)(p-1)$	$2t_s(\sqrt{p} - 1) + t_w m(p-1)$	$t_s(logp) + t_w m(p-1)$.

Rowwise block-striped decomposition

- Scalability analysis
  - $W=n^2$
  - Overhead $T_0=pT_p-W$ → $T_0=n^2+t_s p\log p+t_w np - n^2$
  - $T_0=t_s p\log p + t_w np$
  - When n is reasonably large, communication is dominated by message transmission time. $T_0=t_w np$
  - $W=Kt_w np$
  - ➔$n^2=Kt_w np$
  - ➔$n=Kt_w p$
  - ➔$n^2=K^2 t_w^2 p^2$
  - ➔$W=K^2 t_w^2 p^2$
  - This rate of $\theta(p^2)$ is the asymptotic isoefficiency function of the rowwise block-striped decomposition of matrix vector multiplication.

$$W = \left(\frac{E}{1-E}\right) * T_0(W,p)$$

Columnwise block-striped decomposition

- Communication complexity
  - All to all personalized communication (all to all exchange)
  - n/p values in each message
  - Considering hypercube topology, $t_s+t_w(n/p)p/2(logp) \rightarrow (t_s+t_w n/2)(logp)$ OR $(t_s+t_w(n/p))(p-1) \rightarrow (p-1)t_s+nt_w$
- Computation complexity
  - Each column has n values. Vector has 1 values. *n* multiplications.
  - There are n/p rows. Total = n/p * n = n^2/p
- Parallel run time
  - $T_p=\frac{n^2}{p}+t_s logp+t_w nlogp$ OR $T_p=\frac{n^2}{p}+t_s(p-1)+t_w n$

Algo	Ring	2D Mesh	Hypercube
All-to-all personalized	$t_s+t_w mp/2(p-1)$	$(2t_s + t_w mp)(\sqrt{p} - 1)$	$t_s+t_w mp/2(logp)$ Or $t_s+t_w m(p-1)$

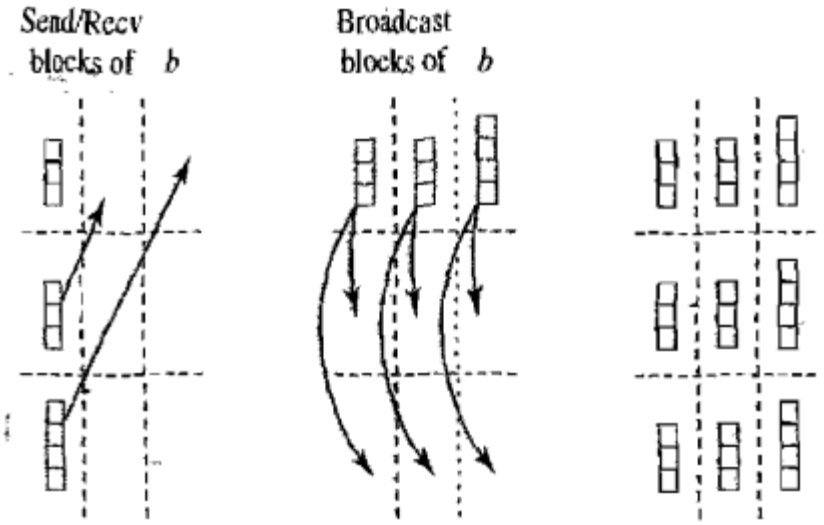
Columnwise block-striped decomposition

- Scalability analysis
  - $W=n^2$
  - Overhead $T_0=pT_p-W$ → $T_0=\frac{p*n^2}{p}+p*t_s(p-1)+p*t_w n-n^2$
  - $T_0=t_s p(p-1)+t_w pn$
  - When n is reasonably large, communication is dominated by message transmission time. $T_0=t_w np$
  - $W=Kt_w np$
  - →$n^2=Kt_w np$
  - →$n=Kt_w p$
  - →$n^2=K^2 t_w^2 p^2$
  - →$W=K^2 t_w^2 p^2$
  - This rate of $\theta(p^2)$ is the asymptotic isoefficiency function of the rowwise block-striped decomposition of matrix vector multiplication.

$$W=\left(\frac{E}{1-E}\right)*T_0(W,p)$$

Chekerboard Block Decomposition

- Communication complexity
  - Vector b is divided among the processes in the first column.
  - Each task in column 0 send to the first task in first row and then broadcasted
  - $t_s + tw\left(\frac{n}{\sqrt{p}}\right) + \left(t_s +$

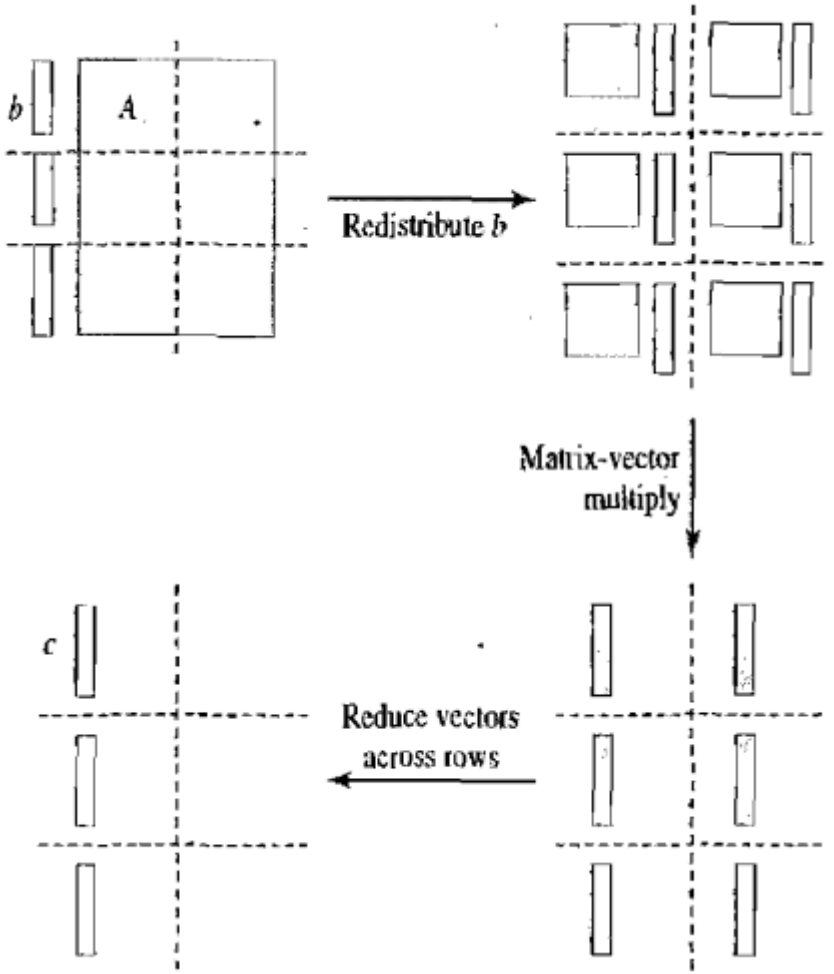


$$T_P = \overbrace{n^2/p}^{\text{computation}} + \overbrace{t_s + t_w n/\sqrt{p}}^{\text{aligning the vector}} + \overbrace{(t_s + t_w n/\sqrt{p})\log(\sqrt{p})}^{\text{columnwise one-to-all broadcast}} + \overbrace{(t_s + t_w n/\sqrt{p})\log(\sqrt{p})}^{\text{all-to-one reduction}}$$

$$\approx \frac{n^2}{p} + t_s \log p + t_w \frac{n}{\sqrt{p}} \log p$$

Chekerboard Block Decomposition

- Redistribute (through broadcast)
- Computation
- All-to-one reduction



- Parallel run time

$$T_P = \overbrace{n^2/p}^{\text{computation}} + \overbrace{t_s + t_w n/\sqrt{p}}^{\text{aligning the vector}} + \overbrace{(t_s + t_w n/\sqrt{p})\log(\sqrt{p})}^{\text{columnwise one-to-all broadcast}} + \overbrace{(t_s + t_w n/\sqrt{p})\log(\sqrt{p})}^{\text{all-to-one reduction}}$$

$$\approx \frac{n^2}{p} + t_s \log p + t_w \frac{n}{\sqrt{p}} \log p$$

Chekerboard Block Decomposition

- Scalability analysis
  - $T_o = pT_p - W$
  - $T_0 = p(\frac{n^2}{p} + tslogp + tw\frac{n}{\sqrt{p}}logp) - n^2$
  - $T_0 = t_s plogp + twn\sqrt{p}logp$
  - $W=n^2=Kt_w n\sqrt{p}logp$ (taking dominating term)
  - $\rightarrow n=Kt_w\sqrt{p}logp$
  - $\rightarrow n^2=K^2t_w{}^2plog^2p$
  - $\rightarrow W=K^2t_w{}^2plog^2p$

$$W = \left(\frac{E}{1-E}\right) * T_0(W,p)$$

Decompositions

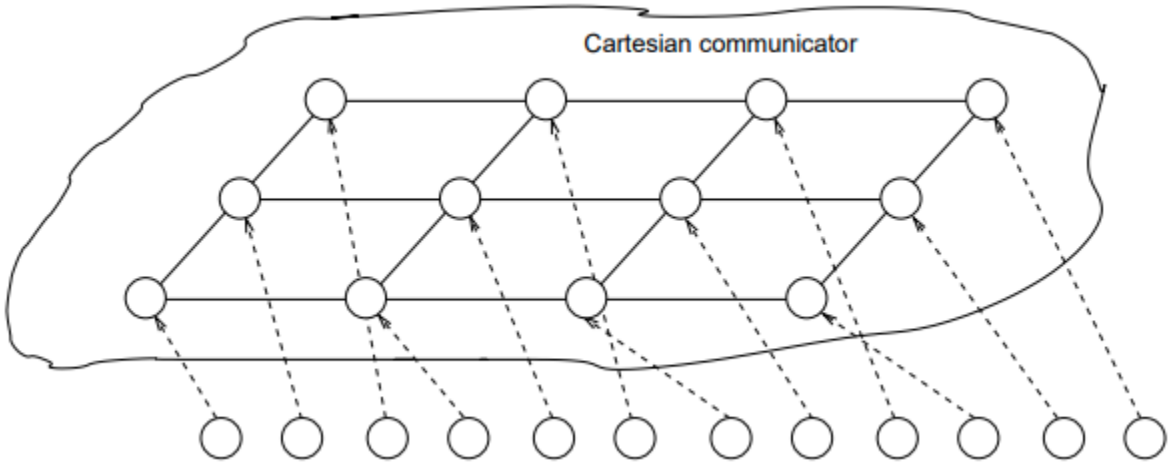
	Communication complexity	Parallel runtime	Isoefficiency function	p? when cost-optimal
Rowwise block-striped decomposition	$t_s(\log p) + t_w(n)$	$\frac{n^2}{p} + t_s \log p + t_w n$	p^2	$p^2=O(n^2)$ $p=O(n)$
Columnwise block striped decomposition	$(p-1)t_s+nt_w$	$\frac{n^2}{p} + t_s(p-1) + t_w n$	p^2	$p^2=O(n^2)$ $p=O(n)$
Checkerboard block decomposition	$t_s \log p + tw\frac{n}{\sqrt{p}}\log p$	$\frac{n^2}{p} + ts\log p + tw\frac{n}{\sqrt{p}}\log p$	$p\log^2 p$	(1) $p\log^2 p=O(n^2)$ $\log p + 2\log\log p=O(\log n)$ $\log p=O(\log n)$ (neglecting 2nd term) Substituting in (1),→ $p\log^2 n=O(n^2)$

Programming

- Cartesian Topologies
  - A Cartesian topology is a topology in which the processes are arranged in a grid of any number of dimensions
  - In order to create a Cartesian topology, MPI needs to know the number of dimensions, the size of each dimension, and whether or not, in each dimension, the coordinates wrap around (so that the first and last index values in that dimension are adjacent)
  - to populate the topology with processes, it needs a communicator containing a group or processes, and it needs to know whether it must preserve their ranks or is free to assign new ranks to them if it chooses. The function to create Cartesian topologies is called MPI_Cart_create.

Programming for Checkerboard (2D)

- Virtual topologies - Cartesian Topologies
  - This function creates a virtual Cartesian process topology. Any subsequent MPI functions that want to use this new Cartesian topology must use comm_cart as the communicator handle.



Mapping of processes into a 3 × 4 two-dimensional Cartesian topology by a call to MPI_Cart_create. It is important to remember that after the call the processes exist in the old communicator as well as the new one

```
#include <mpi.h>
int MPI_Cart_create(
MPI_Comm comm_old, /* handle to the communicator from which processes come */
int ndims, /* number of dimensions in grid */
int *dims, /* integer array of size ndims of grid dimensions */
int *periods, /* logical array of size ndims of flags (1 = periodic) */
int reorder, /* flag indicating is allowed to reorder processes */
MPI_Comm *comm_cart /* returned handle of new communicator */
)
```

Programming for Checkerboard (2D)

```
2   #include <mpi.h>
3   int MPI_Comm_split(
4       MPI_Comm comm, /* handle to the communicator */
5       int color, /* subgroup identifier */
6       int key, /* possibly new rank for calling process */
7       MPI_Comm *newcomm /* handle to new communicator */
8   )
```

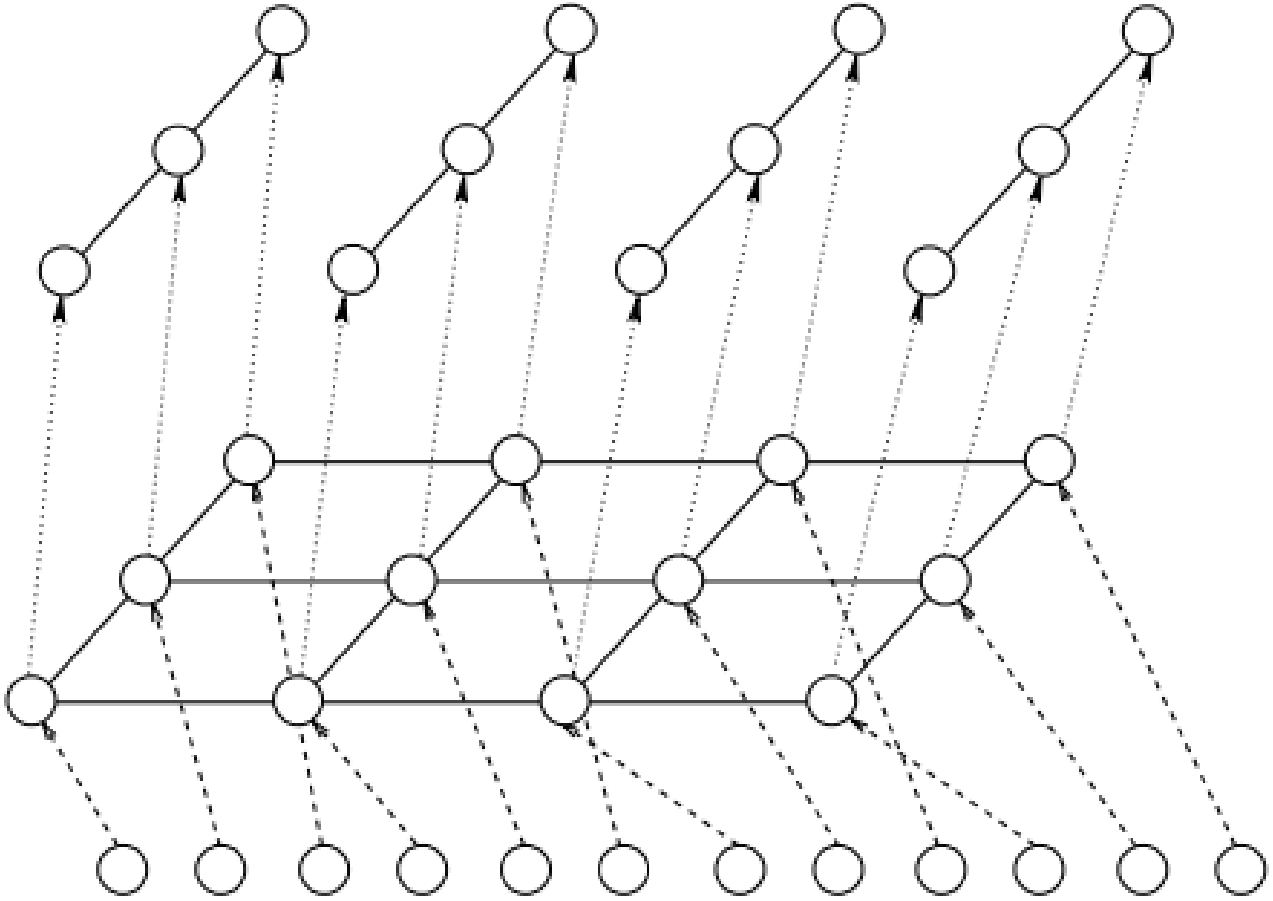
- This function partitions the group associated with original communicator comm into a collection of disjoint subgroups, one for each value of the parameter color
- Each process calls the function with a value for the color parameter. Those processes that supply the same value for color are put into the same subgroup.
  - All of the processes within a subgroup have a new rank in that subgroup. That rank is based on the value of the key parameter. If each process in a given subgroup supplies a unique value for key, then each will have its new rank equal to key. If two or more processes supply the same value for key, then MPI_Comm_split breaks the tie by assigning ranks to these processes according to their ranks in the old group

Programming for Checkerboard (2D)

- Suppose we want to partition Cartesian grid so that all processes in the same grid row are in the same subgroup, so that each row of processes can perform its own reduction operation. Then every process can pass the row index to the color parameter

```
int my_rank; /* rank of process in MPI_COMM_WORLD */
int my_cart_rank; /* rank of process in Cartesian topology */
int my_cart_coords[2]; /* coordinates of process in Cartesian topology */
int dimensions[2] = {0,0}; /* dimensions of Cartesian grid */
int periodic[2] = {0,0}; /* flags to turn off wrapping in grid */
MPI_Comm cartesian_comm; /* Cartesian communicator handle */
MPI_Comm row_comm; /* Communicator for row subgroup */
/* Get optimal dimensions for grid */
MPI_Dims_create(p, 2, dimensions);
/* Create the Cartesian communicator using these dimensions */
MPI_Cart_create(MPI_COMM_WORLD, 2, dimensions, periodic, 1, &cartesian_comm);
/* Compute for a while .... */
/* Get rank of process in the Cartesian communicator */
MPI_Comm_rank( cartesian_comm, &my_cart_rank );
/* Use this rank to get coordinates in the grid */
MPI_Cart_coords(cartesian_comm, my_cart_rank, 2, my_cart_coords );
/* Use my_cart_coords[0] as the subgroup id, and my_cart_coords[1] as new rank
and split off into a group of all processes in the same grid row */
MPI_Comm_split( cartesian_comm, my_cart_coords[0], my_cart_coords[1], &row_comm);
```

Programming for Checkerboard (2D)



- The result of splitting a cartesian communicator by columns. After the split, four more communicators exist

Using GPU

```
#define BLOCKSIZE 16
#define SIZE 1024

// Vector length , Matrix Row and Col sizes..............
     vlength = matColSize = SIZE;
     matRowSize = SIZE;
/*function to launch kernel*/
void launch_Kernel_MatVectMul()
{
/*    threads_per_block, blocks_per_grid  */
int max=BLOCKSIZE*BLOCKSIZE;
int BlocksPerGrid=matRowSize/max+1;
 dim3 dimBlock(BLOCKSIZE,BLOCKSIZE);
if(matRowSize%max==0)BlocksPerGrid--;
dim3 dimGrid(1,BlocksPerGrid);
 check_block_grid_dim(deviceProp,dimBlock,dimGrid);

MatVectMultiplication<<<dimGrid,dimBlock>>>(device_Mat,device_Vect,matRowSize,vlength,device_ResVect);

}
```

- Decomposition
  - One marix row to each thread

GPU

- Sequential vesion

```
1   void CPU_MatVect()
2   {
3       cpu_ResVect = (double *)malloc(matRowSize*sizeof(double));
4       if(cpu_ResVect==NULL)
5                   mem_error("cpu_ResVect","vectmatmul",size,"double");
6
7       int i,j;
8       for(i=0;i<matRowSize;i++)
9       {cpu_ResVect[i]=0;
10      for(j=0;j<matColSize;j++)
11      cpu_ResVect[i]+=host_Mat[i*vlength+j]*host_Vect[j];
12      }
13  }
```

- SIMT version

```
1   __global__ void MatVectMultiplication(double *device_Mat,
2   double *device_Vect,int matRowSize, int vlength,double *device_ResVect)
3     {
4           int tidx = blockIdx.x*blockDim.x + threadIdx.x;
5           int tidy = blockIdx.y*blockDim.y + threadIdx.y;
6           int tindex=tidx+gridDim.x*BLOCKSIZE*tidy;
7
8
9           if(tindex<matRowSize)
10      {
11          int i;int m=tindex*vlength;
12        device_ResVect[tindex]=0.00;
13        for(i=0;i<vlength;i++)
14        device_ResVect[tindex]+=device_Mat[m+i]*device_Vect[i];
15      }
16
17        __syncthreads();
18
19    }//end of MatVect device function
```

References

- Chapter 8.1text book.
- Chapter 8 of R2. M.J. Quinn, Parallel Programming in C with MPI and OpenMP, McGraw Hill Inc. 2nd Edition 2004.

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You