COMP/CS 605: Introduction to Parallel Computing
Topic : MPI: Vector Operations

Mary Thomas

Department of Computer Science
Computational Science Research Center (CSRC)
San Diego State University (SDSU)

Posted: 02/20/17
Updated: 02/21/17

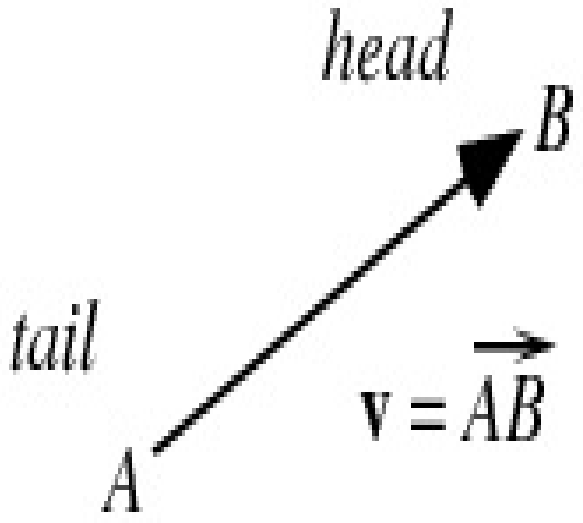
Table of Contents

Pacheco Source code: parallel_dot.c (2/3)

```
/*********************************************************************/
void Read_vector(
         char*  prompt     /* in  */,
         float  local_v[]  /* out */,
         int    n_bar      /* in  */,
         int    p          /* in  */,
         int    my_rank    /* in  */) {
    int i, q;
    float temp[MAX_LOCAL_ORDER];
    MPI_Status status;
    if (my_rank == 0) {
        printf("Enter %s\n", prompt);
        for (i = 0; i < n_bar; i++)
            scanf("%f", &local_v[i]);
        for (q = 1; q < p; q++) {
            for (i = 0; i < n_bar; i++)
                scanf("%f", &temp[i]);
            MPI_Send(temp, n_bar, MPI_FLOAT, q, 0, MPI_COMM_WORLD);
        }
    } else {
        MPI_Recv(local_v, n_bar, MPI_FLOAT, 0, 0, MPI_COMM_WORLD,
            &status);
    }
}  /* Read_vector */
```

In this case, the master core/node does not need to allocate all of *a* only a temp vector; the destination core store the value of *a* into local variable *local_n*

Definition of a Vector



$\vec{A}$ is the tail, $\vec{B}$ is the head

Where $\vec{A} = [a_1, a_2, \ldots, a_n]$, and $\vec{B} = [b_1, b_2, \ldots, b_n]$

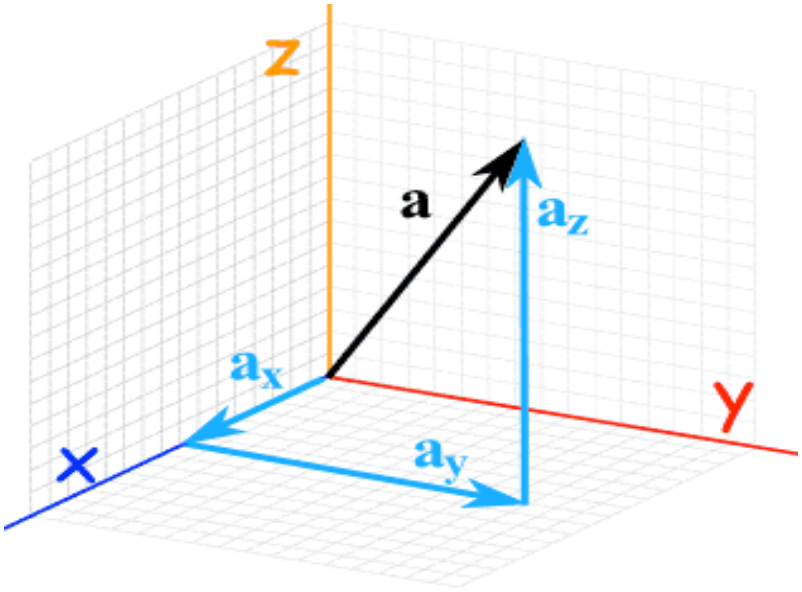
Norm of a vector (length): $|A| = \sqrt{a_1^2 + a_2^2 + \ldots a_n^2}$

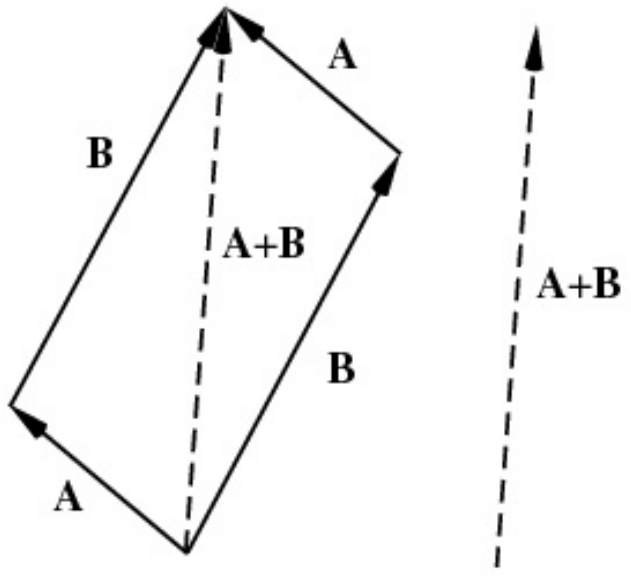
Common Vector Operations

- Addition
- Subtraction (difference)
- Multiplication:
  - multiply by scalars
  - Dot product
  - Cross product
- Different coordinate systems:
  - 3D Cartesian: $\vec{x} = [x_i, x_j, x_k] = [x_1, x_2, x_3]$
  - Spherical:

$$r = \sqrt{x_1^2 + x_2^2 + \ldots x_3^2}$$
$$x = r cos\theta \, sin\phi$$
$$y = r sin\theta \, sin\phi$$
$$z = r cos\phi$$

Vector Addition





For $\vec{A} = [a_1, a_2, \ldots, a_n]$, and $\vec{B} = [b_1, b_2, \ldots, b_n]$:

$$\vec{A} + \vec{B} = \sum_{i=1}^{n} a_i + b_i = (a_1 + b_1,\ a_2 + b_2,\ \cdots,\ a_n + b_n)$$

Image Source: `https://www.mathsisfun.com/algebra/images/vector-3d.gif`

Data distributions

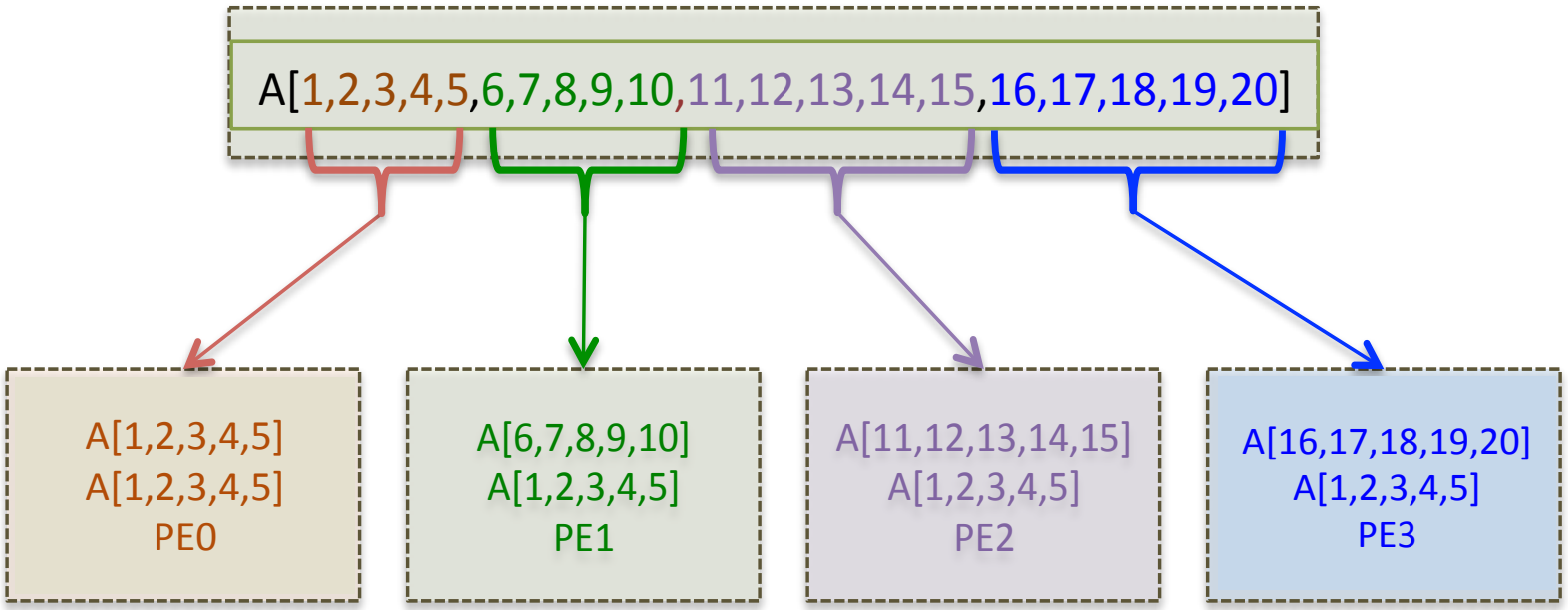
$$
\begin{aligned}
\mathbf{x}+\mathbf{y} &= (x_0, x_1, \ldots, x_{n-1}) + (y_0, y_1, \ldots, y_{n-1}) \\
&= (x_0 + y_0, x_1 + y_1, \ldots, x_{n-1} + y_{n-1}) \\
&= (z_0, z_1, \ldots, z_{n-1}) \\
&= \mathbf{z}
\end{aligned}
$$

Compute a vector sum.

Serial implementation of vector addition

```
void Vector_sum(double x[], double y[], double z[], int n) {
   int i;

   for (i = 0; i < n; i++)
      z[i] = x[i] + y[i];
} /* Vector_sum */
```

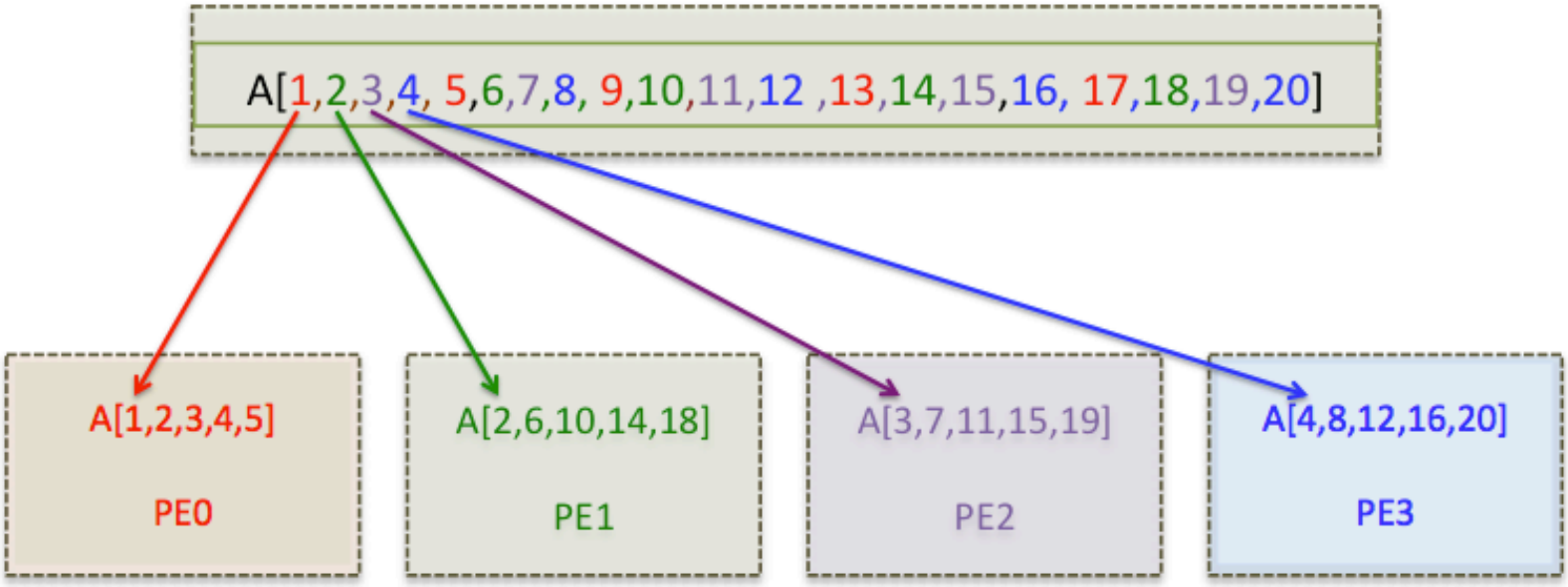
1D Vector Distribution: Block



1D Vector Data Distribution

(Using Fortran indexing)

1D Vector Distribution: Cyclic



1D Vector Data Distribution: Cyclic (Note: using Fortran indexing)

Parallel Vector Sum (Pacheco IPP-2011 Ch3)

```
int main(void) {
   int n, local_n;
   int comm_sz, my_rank;
   double *local_x, *local_y, *local_z;
   MPI_Comm comm;

   MPI_Init(NULL, NULL);
   comm = MPI_COMM_WORLD;
   MPI_Comm_size(comm, &comm_sz);
   MPI_Comm_rank(comm, &my_rank);

   Read_n(&n, &local_n, my_rank, comm_sz, comm);
#  ifdef DEBUG
   printf("Proc %d > n = %d, local_n = %d\n", my_rank, n, local_n);
#  endif
   Allocate_vectors(&local_x, &local_y, &local_z, local_n, comm);

   Read_vector(local_x, local_n, n, "x", my_rank, comm);
   Print_vector(local_x, local_n, n, "x is", my_rank, comm);
   Read_vector(local_y, local_n, n, "y", my_rank, comm);
   Print_vector(local_y, local_n, n, "y is", my_rank, comm);

   Parallel_vector_sum(local_x, local_y, local_z, local_n);
   Print_vector(local_z, local_n, n, "The sum is", my_rank, comm);

   free(local_x);
   free(local_y);
   free(local_z);

   MPI_Finalize();

   return 0;
}  /* main */
```

Parallel Vector Sum (Pacheco IPP-2011 Ch3)

```
/*---------------------------------------------------------------------
 * Function:  Read_n
 * Purpose:   Get the order of the vectors from stdin on proc 0 and
 *            broadcast to other processes.
 * In args:   my_rank:    process rank in communicator
 *            comm_sz:    number of processes in communicator
 *            comm:       communicator containing all the processes
 *                        calling Read_n
 * Out args:  n_p:        global value of n
 *            local_n_p:  local value of n = n/comm_sz
 *
 * Errors:    n should be positive and evenly divisible by comm_sz
 */
void Read_n(
      int*      n_p        /* out */,
      int*      local_n_p  /* out */,
      int       my_rank    /* in  */,
      int       comm_sz    /* in  */,
      MPI_Comm  comm       /* in  */) {
   int local_ok = 1;
   char *fname = "Read_n";

   if (my_rank == 0) {
      printf("What's the order of the vectors?\n");
      scanf("%d", n_p);
   }
   MPI_Bcast(n_p, 1, MPI_INT, 0, comm);
   if ( (*n_p <= 0) || (*n_p % comm_sz != 0) ) local_ok = 0;
   Check_for_error(local_ok, fname,
         "n should be > 0 and evenly divisible by comm_sz", comm);
   *local_n_p = *n_p/comm_sz;
}  /* Read_n */
```

Parallel Vector Sum (Pacheco IPP-2011 Ch3)

```
/*-------------------------------------------------------------------
 * Function:  Allocate_vectors
 * Purpose:   Allocate storage for x, y, and z
 * In args:   local_n:  the size of the local vectors
 *            comm:     the communicator containing the calling processes
 * Out args:  local_x_pp, local_y_pp, local_z_pp:  pointers to memory
 *               blocks to be allocated for local vectors
 *
 * Errors:    One or more of the calls to malloc fails
 */
void Allocate_vectors(
      double**   local_x_pp  /* out */,
      double**   local_y_pp  /* out */,
      double**   local_z_pp  /* out */,
      int        local_n     /* in  */,
      MPI_Comm   comm        /* in  */) {
   int local_ok = 1;
   char* fname = "Allocate_vectors";

   *local_x_pp = malloc(local_n*sizeof(double));
   *local_y_pp = malloc(local_n*sizeof(double));
   *local_z_pp = malloc(local_n*sizeof(double));

   if (*local_x_pp == NULL || *local_y_pp == NULL ||
       *local_z_pp == NULL) local_ok = 0;
   Check_for_error(local_ok, fname, "Can't allocate local vector(s)",
         comm);
}  /* Allocate_vectors */
```

Parallel Vector Sum (Pacheco IPP-2011 Ch3

```
 * Function:   Read_vector
 * Purpose:    Read a vector from stdin on process 0 and distribute among the processes using a block distribution.
 * Errors:     if the malloc on process 0 for temporary storage fails the program terminates
 * Note:
 *    This function assumes a block distribution and the order
 *   of the vector evenly divisible by comm_sz.
 */
void Read_vector(
      double    local_a[]   /* out */,
      int       local_n     /* in  */,
      int       n           /* in  */,
      char      vec_name[]  /* in  */,
      int       my_rank     /* in  */,
      MPI_Comm  comm        /* in  */) {

   double* a = NULL;
   int i;
   int local_ok = 1;
   char* fname = "Read_vector";

     if (my_rank == 0) {
      a = malloc(n*sizeof(double));
      if (a == NULL) local_ok = 0;
      Check_for_error(local_ok, fname, "Can't allocate temporary vector",
            comm);
      printf("Enter the vector %s\n", vec_name);
      for (i = 0; i < n; i++)
         scanf("%lf", &a[i]);
      MPI_Scatter(a, local_n, MPI_DOUBLE, local_a, local_n, MPI_DOUBLE, 0,
         comm);
      free(a);
   } else {
      Check_for_error(local_ok, fname, "Can't allocate temporary vector",
            comm);
      MPI_Scatter(a, local_n, MPI_DOUBLE, local_a, local_n, MPI_DOUBLE, 0,
         comm);
   }
}  /* Read_vector */
```

Parallel Vector Sum (Pacheco IPP-2011 Ch3

```
/* Function:  Print_vector
* Purpose:   Print a vector that has a block distribution to stdout
*    Assumes order of vector is evenly divisible by the number of processes  */
void Print_vector(
      double    local_b[]  /* in */,
      int       local_n    /* in */,
      int       n          /* in */,
      char      title[]    /* in */,
      int       my_rank    /* in */,
      MPI_Comm  comm       /* in */) {

   double* b = NULL;
   int i;
   int local_ok = 1;
   char* fname = "Print_vector";

   if (my_rank == 0) {
      b = malloc(n*sizeof(double));
      if (b == NULL) local_ok = 0;
      Check_for_error(local_ok, fname, "Can't allocate temporary vector",
            comm);
      MPI_Gather(local_b, local_n, MPI_DOUBLE, b, local_n, MPI_DOUBLE,
            0, comm);
      printf("%s\n", title);
      for (i = 0; i < n; i++)
         printf("%f ", b[i]);
      printf("\n");
      free(b);
   } else {
      Check_for_error(local_ok, fname, "Can't allocate temporary vector",
            comm);
      MPI_Gather(local_b, local_n, MPI_DOUBLE, b, local_n, MPI_DOUBLE, 0,
         comm);
   }
}  /* Print_vector */
```

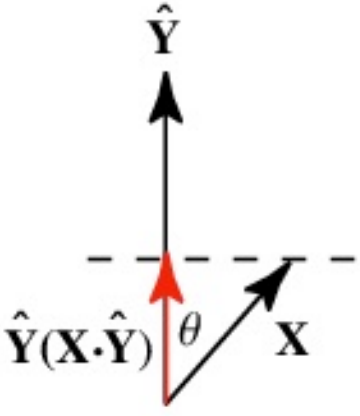
Parallel Vector Sum (Pacheco IPP-2011 Ch3

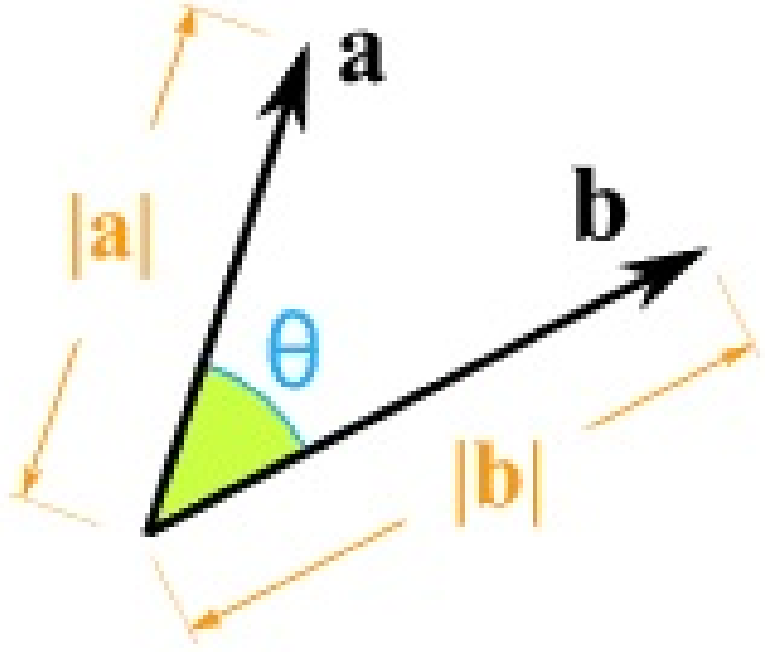
```
/*--------------------------------------------------------------------
 * Function:  Parallel_vector_sum
 * Purpose:   Add a vector that's been distributed among the processes
 * In args:   local_x:  local storage of one of the vectors being added
 *            local_y:  local storage for the second vector being added
 *            local_n:  the number of components in local_x, local_y,
 *                      and local_z
 * Out arg:   local_z:  local storage for the sum of the two vectors
 */
void Parallel_vector_sum(
      double  local_x[]  /* in  */,
      double  local_y[]  /* in  */,
      double  local_z[]  /* out */,
      int     local_n    /* in  */) {
   int local_i;

   for (local_i = 0; local_i < local_n; local_i++)
      local_z[local_i] = local_x[local_i] + local_y[local_i];
}  /* Parallel_vector_sum */
```

Pacheco Vector Add: Output

```
[gidget] mthomas% mpicc -o vector_add vector_add.c
[gidget] mthomas% ./vector_add
What's the order of the vectors?
3
Enter the vector x
1 2 3 4 5 6 7
Enter the vector y
The sum is
5.000000 7.000000 9.000000
[gidget] mthomas% ./vector_add
What's the order of the vectors?
4
Enter the vector x
1 2 3 4
Enter the vector y
5 6 7 8
The sum is
6.000000 8.000000 10.000000 12.000000
[gidget:intro-par-pgming-pacheco/ipp-source/ch3] mthomas%
```

Vector Dot Product

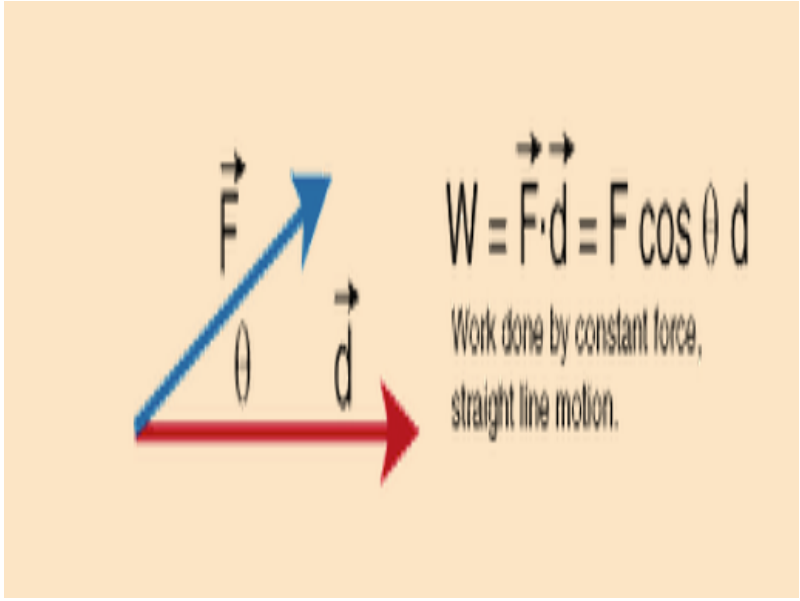


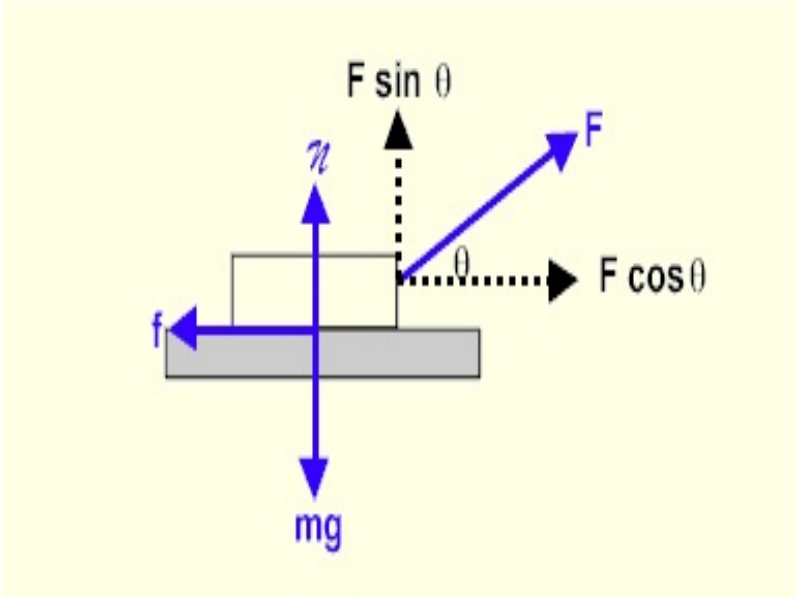


Dot Product is: $\vec{X} \bullet \vec{Y} = |X|\,|Y|\,cos\theta$

Geometric interpretation: length of the projection of $\vec{X}$ onto $\vec{Y}$

$$\vec{X} \bullet \vec{Y} = \sum_{i=1}^{n} A_i B_i = A_1 B_1 + A_2 B_2 + \cdots + A_n B_n$$





Source: http://hyperphysics.phy-astr.gsu.edu/hbase/vsca.html#vsc3 Source: http://dev.physicslab.org/Document.aspx?doctype=3&filename=Dynamics_ForcesActingAngle.xml

Pacheco Source code: parallel_dot.c (1/3)

```
/* parallel_dot.c -- compute a dot product of a vector distributed among
 *     the processes.  Uses a block distribution of the vectors.
 *
 * Note:  Arrays containing vectors are statically allocated.  Assumes  n, the global order
 *     of the vectors, is divisible by p, the number of processes.
 * See Chap 5, pp. 75 & ff in PPMPI.
 */
#include <stdio.h>
#include "mpi.h"
#define MAX_LOCAL_ORDER 100
main(int argc, char* argv[]) {
    float  local_x[MAX_LOCAL_ORDER];
    float  local_y[MAX_LOCAL_ORDER];
    int    n,n_bar;  /* = n/p */
    float  dot;
    int    p,my_rank;
    void Read_vector(char* prompt, float local_v[], int n_bar, int p,int my_rank);
    float Parallel_dot(float local_x[], float local_y[], int n_bar);

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    if (my_rank == 0) {
        printf("Enter the order of the vectors\n");
        scanf("%d", &n);
    }
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    n_bar = n/p;

    Read_vector("the first vector", local_x, n_bar, p, my_rank);
    Read_vector("the second vector", local_y, n_bar, p, my_rank);

    dot = Parallel_dot(local_x, local_y, n_bar);
    if (my_rank == 0)  printf("The dot product is %f\n", dot);

    MPI_Finalize();
}  /* main */
```

Pacheco Source code: parallel_dot.c (2/3)

```
/*********************************************************************/
void Read_vector(
         char*  prompt     /* in  */,
         float  local_v[]  /* out */,
         int    n_bar      /* in  */,
         int    p          /* in  */,
         int    my_rank    /* in  */) {
    int i, q;
    float temp[MAX_LOCAL_ORDER];
    MPI_Status status;
    if (my_rank == 0) {
        printf("Enter %s\n", prompt);
        for (i = 0; i < n_bar; i++)
            scanf("%f", &local_v[i]);
        for (q = 1; q < p; q++) {
            for (i = 0; i < n_bar; i++)
                scanf("%f", &temp[i]);
            MPI_Send(temp, n_bar, MPI_FLOAT, q, 0, MPI_COMM_WORLD);
        }
    } else {
        MPI_Recv(local_v, n_bar, MPI_FLOAT, 0, 0, MPI_COMM_WORLD,
            &status);
    }
}  /* Read_vector */
```

Pacheco Source code: parallel_dot.c (3/3)

```
/*********************************************************************/
float Serial_dot(
          float  x[]  /* in */,
          float  y[]  /* in */,
          int    n    /* in */) {

    int    i;
    float  sum = 0.0;

    for (i = 0; i < n; i++)
        sum = sum + x[i]*y[i];
    return sum;
} /* Serial_dot */


/*********************************************************************/
float Parallel_dot(
          float  local_x[]  /* in */,
          float  local_y[]  /* in */,
          int  n_bar        /* in */) {

    float  local_dot;
    float  dot = 0.0;
    float  Serial_dot(float x[], float y[], int m);

    local_dot = Serial_dot(local_x, local_y, n_bar);
    MPI_Reduce(&local_dot, &dot, 1, MPI_FLOAT,
        MPI_SUM, 0, MPI_COMM_WORLD);
    return dot;
} /* Parallel_dot */
```

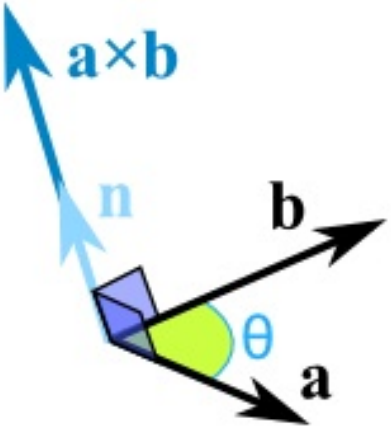
Parallel Vector Dot Product: Output

```
[mthomas@tuckoo chap05]$
[mthomas@tuckoo chap05]$ mpicc -o parallel_dot parallel_dot.c
[mthomas@tuckoo chap05]$
[mthomas@tuckoo chap05]$ mpirun -np 4 ./parallel_dot
Enter the order of the vectors
12
Enter the first vector
1 2 3 4 5 6 7 8 9 10 11 12
Enter the second vector
2 4 6 8 10 12 14 16 18 20 22 24 26
The dot product is 1300.000000
```

Vector Cross Product

The Cross Product of two vectors is another vector that is at right angles to both. Cross Product is: $\vec{a} \times \vec{b} = |a|\,|b|\,sin\,(\theta)\,\vec{n}$

Examples: unit normal vector



- $|a|$ is the magnitude of $\vec{a}$
- $|b|$ is the magnitude of $\vec{b}$
- θ is the angle between $\vec{a}$ and $\vec{b}$
- $\vec{n}$ is the unit vector at right angles to both $\vec{a}$ and $\vec{b}$

`https://en.wikipedia.org/wiki/Cross_product`