*VECTOR SUM AND PRODUCT*

SOURCE CODE:

#include "mpi.h" #include <stdio.h> #include <stdlib.h> #include <time.h> #define MASTER 0

#define ARRAY\_SIZE 20000

int main (int argc, char \*argv[])

{

int \* a; int \* b; int \* c;

int total\_proc; // total nuber of processes int rank; // rank of each process

int n\_per\_proc; // elements per process

int n = ARRAY\_SIZE; // number of array elements int i; // loop index

MPI\_Status status;

// initialization of MPI environment MPI\_Init (&argc, &argv);

MPI\_Comm\_size (MPI\_COMM\_WORLD, &total\_proc);\ MPI\_Comm\_rank (MPI\_COMM\_WORLD,&rank);

// Now you know the rank of the current process

// Smaller arrays that will be held on each separate process int \* ap;

int \* bp; int \* cp;

// Choose process rank 0 to be the root, or master,

// which will be used to initialize the full arrays. double tick;

if (rank == MASTER) { tick = clock();

a = (int \*) malloc(sizeof(int)\*n); b = (int \*) malloc(sizeof(int)\*n); c = (int \*) malloc(sizeof(int)\*n); for(i=0;i<n;i++)

a[i] = i+4\*i; for(i=0;i<n;i++)

b[i] = i\*i-1;

//printf("\nArray A: ");

//for(i=0;i<n;i++) printf("%d ",a[i]);

//printf("\nArray B: ");

//for(i=0;i<n;i++) printf("%d ",b[i]);

}

// determine how many elements each process will work on n\_per\_proc = n/total\_proc;

// 5. Initialize my smaller subsections of the larger array ap = (int \*) malloc(sizeof(int)\*n\_per\_proc);

bp = (int \*) malloc(sizeof(int)\*n\_per\_proc); cp = (int \*) malloc(sizeof(int)\*n\_per\_proc);

//scattering array a from MASTER node out to the other nodes MPI\_Scatter(a, n\_per\_proc, MPI\_INT, ap, n\_per\_proc, MPI\_INT, MASTER,

MPI\_COMM\_WORLD);

//scattering array b from MASTER node out to the other node MPI\_Scatter(b, n\_per\_proc, MPI\_INT, bp, n\_per\_proc, MPI\_INT, MASTER,

MPI\_COMM\_WORLD);

// Compute the addition of elements in my subsection of the array for(i=0;i<n\_per\_proc;i++)

cp[i] = ap[i]+bp[i]; // + FOR SUM \* FOR PRODUCT

// MASTER node gathering array c from the workers

MPI\_Gather(cp, n\_per\_proc, MPI\_INT, c, n\_per\_proc, MPI\_INT, MASTER, MPI\_COMM\_WORLD);

if (rank == MASTER) {

//printf("\nResultant: "); for(i=0;i<n;i++) {

//printf ("%d ", c[i]);

if (c[i] != a[i] + b[i]) { printf("error”); break;

}

}

printf("\n");

double tock = clock();

printf("Time taken: %lf\n", (tock-tick)/CLOCKS\_PER\_SEC);

}

if (rank == MASTER) {

free(a); free(b); free(c);

}

free(ap); free(bp); free(cp); MPI\_Finalize();

return 0;

}