Multi Cores : Mixed Mode of Programming :Using MPI & OpenMP

Example 1.1 Write an MPI-OpenMP program to print *Hello World"* . You have to use MPI Basic Library Calls and OpenMP PARALLEL For Directive.

Example 1.2 Write an MPI-OpenMP program to compute the value of PI *pie* function by numerical integration of a function $f(x) = 4/(1+x^2)$ between the limits 0 and 1. You have to use MPI Collective Communication and Computation Library Calls and OpenMP PARALLEL For Directive and CRITICAL section.

Example 1.3 Write an MPI-OpenMP program to calculate Infinity norm of a matrix using block striped partitioning with row wise data distribution. You have to use MPI Collective Communication and OpenMP Parallel For Directive and PRIVATE, SHARED Clauses.

Example 1.4 Write a MPI-OpenMP program to compute the matrix-vector multiplication using self scheduling algorithm. You have to use MPI Collective Communication and OpenMP *Parallel For Directive and PRIVATE, SHARED Clauses.*

Example 1.5 Write a MPI-OpenMP program to compute the matrix into matrix Multiplication using Checker-Board Partitoning of input Matrices (Assignment).

Example 1.6 Write a MPI-openmp program to solve a system of linear equations Ax=b using Conjugate Gradient Method. (Assignment).

Description of MPI-OpenMP Programs

Example 1.1 : Write an MPI-OpenMP program to print *Hello World"* . You have to use MPI basic Library Calls and OpenMP PARALLEL Directive

(Download source code :
mpi-Omp-hello-world.c) / ***mpi-Omp-hello-world.f)***

- **Objective**

  Write an MPI-OpenMP program to print *Hello World"* . You have to use MPI basic Library Calls and OpenMP PARALLEL Directives using *p* processes and t threads

- **Description**

  This is a very simple program to get the feel of threads and to get a feel of how to use threads with MPI. The implementation is as follows: The MPI application starts and each process creates two child threads. The main thread on each process passes an identifier to the thread and the address of the function to execute. Each thread will print the message "Hello World from thread: thread ID on process: Process No.

- **Input**

  None

- **Output**

  Hello World from Thread: thread No. on Process: process No.

Example 1.2 : Write an MPI-OpenMP program to compute the value of PI *pie* function by numerical integration of a function $f(x) = 4/(1+x^2)$ between the limits 0 and 1. You have to use MPI Collective Communication and Computation Library Calls and OpenMP PARALLEL FOR Directive and CRITICAL section.

(Download source code :
mpi-omp-pie-calculation.c) / ***mpi-omp-pie-calculation.f)***

- **Objective**

Write an MPI-OpenMP program to compute the value of PI *pie* function by numerical integration of a function $f(x) = 4/(1+x^2)$ between the limits 0 and 1. You have to use MPI Collective Communication and Computation Library Calls and OpenMP PARALLEL FOR Directive and CRITICAL section on *p* processes and t threads
.

- **Description**

  This program computes the value of PI over a given interval using Numerical integration. All the process determine the number of intervals to be calculated by it. Each process then creates threads as the number of intervals it is to calculate. The main thread assigns an interval to each thread. Each thread calculates thr value in its interval and adds it to a common variable. This variable is protected by a Critical section to ensure the atomicity of the operations. When all the child threads on a process finish, the variable holds the value in the interval assigned to it. Using a collective call, MPI_Reduce, the root process accumulates the calculated value of PI.

- **Input**

  Number of intervals.

- **Output**

  Calculated Value of PI.

Example 1.3 : Write an MPI-OpenMP program to calculate Infinity norm of a matrix using block striped partitioning with row wise data distribution. You have to use MPI Collective Communication and OpenMP Parallel For Directive and PRIVATE, SHARED Clauses.

(Download source code :
mpi-omp-mat-infnorm-blkstp.c) / ***mpi-omp-mat-infnorm-blkstp.f*** ; ***infndata.inp***)

- **Objective**

  Write an MPI-OpenMP program to calculate Infinity norm of a matrix using block striped partitioning with row wise data distribution.You have to use MPI Collective Communication and OpenMP Parallel For Directive and PRIVATE, SHARED Clauses on *p* processe and t threads.

- **Description**

  Infinity Norm of a Matrix: The Row-Wise infinity norm of a matrix is defined to be the maximum of sums of absolute values of elements in a row, over all rows. After the initial validity checks, each process reads the input matrix and determines the number of rows to be operated by it. Using its rank, each process determines the specific rows to be operated by it. After the distribution of rows, the main thread on each process creates the child threads as the number of rows it is to operate. Each thread operates on the specified row and calculates the sum of the absolute values of the elements and updates a common variable. After all the threads on a process complete their share of work, the common variable holds the maximum value of the rows assigned to it. Finally, the Root process determines the infinity norm using a Collective MPI call, MPI_Reduce and prints the value.

- **Input**

  The input file holding the square matrix

- **Output**

  Infinity Norm of the Matrix Value

Example 1.4 : Write an MPI-OpenMP program to calculate Infinity norm of a matrix using block striped partitioning with row wise data distribution. You have to use MPI Collective Communication and OpenMP Parallel For Directive and PRIVATE, SHARED Clauses.

(Download source code :
mpi-omp-mat-vect-mult-blkstp.c) / ***mpi-omp-mat-vect-mult-blkstp.f***; ***mdata.inp***; ***vdata.inp***)

- **Objective**

  Write a MPI-OpenMP program to compute the matrix-vector multiplication using self scheduling algorithm. You have to use MPI Collective Communication and OpenMP *Parallel For Directive and PRIVATE, SHARED Clauses on p processes and* t threads.

- **Description**

  In self scheduling algorithm, a master distributes the rows of the matrix to worker threads. The worker threads calculate the dot product of the given row and the vector. When all the workers finish their part of the work, the resultant vector is the product of the given matrix and the vector. In this implementation, all the processes after the initial checks and populating the matrix and the vector, determine the rows of

the matrix it needs to operate upon. The main thread on each process creates child threads to operate upon the rows it is assigned to. Each thread calculates the dot product of the row assigned to it and the vector and stores it in a local array on the process. After all the child threads on each process are done, the array holds the dot product of the rows of the matrix assigned to it and the vector. Finally, using a collective MPI call, MPI_Gatherv to accumulate the result vector.

- **Input**

  The input file holding the square matrix (Number of Rows, Columns of the matrix)

- **Output**

  Output Array : Product of Matrix Vector Multiplication.

Example 1.5 : Write a MPI-OpenMP program to compute the matrix into matrix Multiplication using Checker-Board Partitoning of input Matrices.

- **Objective**

  Write a MPI-OpenMP program to compute the matrix into matrix Multiplication using Checker-Board Partitoning of input Matrices. You have to use MPI Collective Communication and OpenMP *Parallel ForDirective and PRIVATE, SHARED Clauses on p processes and* t threads.

- **Description**

  In checkerboard partitioning, the matrix is divided into smaller square or rectangular blocks (submatrices) that are distributed among processes. A checkerboard partitioning splits both the rows and the columns of the matrix, so no process is assigned any complete row or column . Like striping partitioning, checkerboard partitioning can be block or cyclic.

- **Input**

  The input file holding the two square matrices (Number of Rows, Columns of the matrix)

- **Output**

  Output Array : Product of Matrix matrix Multiplication.

Example 1.6 : Write a MPI-OpenMP program to compute the solution for matrix system of linear equations **[A] {x} = {b}** by Conjugate Graident Method method using *p* processes and t threads.

- **Objective**

  Write an MPI-Pthreads program to compute the soluiton for matrix system of linear equations **[A] {x} = {b}b** by Conjugate Graident Method method using *p* processes and t threads.You have to use MPI Collective Communication and OpenMP *Parallel ForDirective and PRIVATE, SHARED Clauses on p processes and* t threads.

- **Description**

  Iterative methods are techniques to solve systems of equations of the form **[A]{x}={b}** that generate a sequence of approximations to the solution vector x. In each iteration, the coefficient matrix **A** is used to perform a matrix-vector multiplication. The number of iterations required to solve a system of equations with a desired precision is usually data dependent; hence, the number of iterations is not known prior to executing the algorithm. The Jacobi method starts with an initial guess x_0 for the solution vector x. This initial vector x_0 is used in the approriate equation sof Jacobi Algorithm to arrive at the next approximation x_1 to the solution vector. The vector x_1 is then used in the approriate equation sof Jacobi algorithm, and the process continues until a close enough approximation to the actual solution is found.

- **Input**

  The input file holding square matrix (Number of Rows, Columns of the matrix); and the Vector

- **Output**

  Output Array : Solution Array of Linear System of Equations

---