# MPI Demo

---

**hello_mpi.c:**

```
#include <stdio.h>
#include "mpi.h"

int main( int argc, char *argv[])
{
    int rank, size;
    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    printf( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

---

**pi_mpi.c:**

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double );
double f( double a )
{
    return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stderr,"Process %d on %s\n",
           myid, processor_name);

    n = 0;
    while (!done)
    {
        if (myid == 0)
        {
/*
            printf("Enter the number of intervals: (0 quits) ");
            scanf("%d",&n);
*/
           if (n==0) n=100; else n=0;

           startwtime = MPI_Wtime();
        }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (n == 0)
```

```
            done = 1;
        else
        {
            h   = 1.0 / (double) n;
            sum = 0.0;
            for (i = myid + 1; i <= n; i += numprocs)
            {
                x = h * ((double)i - 0.5);
                sum += f(x);
            }
            mypi = h * sum;

            MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

            if (myid == 0)
           {
                printf("pi is approximately %.16f, Error is %.16f\n",
                       pi, fabs(pi - PI25DT));
             endwtime = MPI_Wtime();
             printf("wall clock time = %f\n",
                    endwtime-startwtime);
           }
        }
    }
    MPI_Finalize();

    return 0;
}
```

# OpenMP Demo

---

**hello_openmp.c:**

```
#include <omp.h>
#include <stdio.h>

int main()
{
#pragma omp parallel
  {
    int ID = omp_get_thread_num();
    printf(" hello(%d) \n", ID);
    printf(" world(%d) \n", ID);
  }
}
```

---

**pi_openmp.c:**

```
#include <omp.h>
#include <stdio.h>

static long num_steps = 100000;
double step;

#define NUM_THREADS 2

int main()
{
  int i;
  double x, pi, sum = 0.0;
  step = 1.0/(double) num_steps;
  omp_set_num_threads(NUM_THREADS);

#pragma omp parallel for reduction(+:sum) private(x)
  for (i=1;i<= num_steps; i++) {
    x = (i-0.5)*step;
    sum = sum + 4.0/(1.0+x*x);
  }
  pi = step * sum;

  printf("pi = %.16f\n",pi);
}
```

# STAR-P Demo

```
np % Number of processes

% Big matrix multiplications
n=500*p
a=randn(n)
b=randn(n)
tic, c=a*b; toc

n=n*2, a=randn(n); b=randn(n);
tic, c=a*b; toc

n=n*2, a=randn(n); b=randn(n);
tic, c=a*b; toc

n=n*2, a=randn(n); b=randn(n);
tic, c=a*b; toc

% Big matrix inverse
n=500*p
a=randn(n); tic, b=inv(a); toc
n=n*2, a=randn(n); tic, b=inv(a); toc
n=n*2, a=randn(n); tic, b=inv(a); toc

e=eye(n)
c=a*b-e
norm(c,inf)

% Hilbert matrix
type hilb
hilb(3)
hilb(7)
c=hilb(6*p)
c(:,:)
c(:,:)-hilb(6)
c3 = c*c*c;
c3(:,:)-hilb(6)^3

% Big dense linear system
n=2000*p;
b=randn(n,1)
a=randn(n)
c=a\b;
residual = a*c-b
norm(residual)

% MM-Mode - Compute pi

% Method: atan(1)*4
format long
atan(1)*4

% Use MATLAB quadrature
quadl('4./(1+x.^2)',0,1,1e-14)

a = (0:(np-1)*p)/np
a(:)
b = a + (1/np)
b(:)
mypi = mm('quadl','4./(1+x.^2)',a,b,1e-14);
mypi(:)

mypi=sum(mypi)
```