Hybrid Distributed Programming

Mohamad Ibrahim Cheikh

November 30, 2020

Introduction

Parallel computing refers to simulations in which multiple computational resources are implemented to solve a problem at the same time. Parallelism is usually achieved by dividing the problem into many smaller ones, in which each one is solved separately and also at the same time. Different forms of parallel programing has been developed like bit-level, instruction-level, data, and task parallelism. Some widely used parallel programming paradigms are based on message passing, such as MPI [3], which is suitable both on distributed memory (DM) and distributed-shared memory (DSM) systems architectures. Other popular models are based over the parallel compiler directives, such as OpenMP [4] and HPF [5], where the OpenMP takes advantage of the shared memory parallelism, and the HPF exploits the data parallelism.

The current report will implement a hybrid MPI/Openmp to solve a problem on a high performance machine like Beocat.

Experimental Configuration

The supercomputer on which we are going to conduct our performance analysis is Beocat, which is a high performance computer cluster at the Kansas State University [1]. The performance analysis will be performed on the Elves nodes of Beocat [2]. Table 1 gives a brief description of the nodes configuration.

Elve Nodes				
Nodes	1-56	57-72,77	73-76,78, 79	80-85
Processors	2x 8-Core Xeon E5-2690	2x 10-Core Xeon E5-2690 v2	2x 10-Core Xeon E5-2690 v2	2x 10-Core Xeon E5-2690v2
Ram	64GB	96GB	384GB	64GB
Hard Drive	1x 250GB 7,200 RPM SATA	1x 250GB 7,200 RPM SATA	1x 250GB 7,200 RPM SATA	1x 250GB 7,200 RPM SATA
NICs	4x Intel I350	4x Intel I350	4x Intel I350	4x Intel I350
10GbE and QDR Infiniband	MT27500 Family (ConnectX-3)	MT27500 Family (ConnectX-3)	MT27500 Family (ConnectX-3)	MT27500 Family (ConnectX-3)

Table 1: Elves node configuration taken from [2]

Part 1: Numerical Integration of Easom's function

The first problem that requires the implementation of the MPI/Openmp algorithm is finding the numerical integration of Easom's function defined below:

$$f(x,y) = -cos(x)sin(y)exp(-[(x-\pi)^2 + (y-\pi)^2]) \tag{1}$$

To integrate this function one has first to discretize the domain into smaller sub domains. Since this is a 2D problem then the smaller domains will be rectangles of size $dx.dy$. After dividing the domain, the program has to loop over all the small rectangles, multiply them by $f(x,y)$ and sum them up to get the the final integral value. So the code should do something like:

```
for(i = 0; i < endXdir ; i++)
  for(j = 0; j < endYdir; j++)
    volume += h*h*f(x + i*h, y + j*h);
```

where it loops over the points in x and y, calculates the volume using h (which is a small number $\approx$ 0.00001), and then add them up to get the final integral value.

MPI/Openmp

The hybrid MPI/Openmp implementation for the current problem will try to divide the domain over all the cores. The first step it will do is divide the domain along the x direction according to the number of MPI processes. This is achieved by controlling the $starting$ and $ending$ of i-th loop or first loop. After the domain is a divided into multiple smaller domains along the x-axis the code will enter the Openmp environment and discretize the mesh into even smaller domains, but this time along the y direction according to the number of Openmp threads.

So as a summary, we started with a large domain, and divided it first into smaller domains along the x-axis for the MPI enviroment, and then divided into smaller domains along the y-axis for the OpenMp domain. After the program finishes the with summing up the volumes in the parallelized OpenMp environment it will add them up together to get the final OpenMp version, by utilizing the critical command in Openmp. When this is over each MPI pocess will have part of the whole volume, to get the final one we will use MPI-reduce to get the final value of the integration. The figure gives a better picture of how the code works.

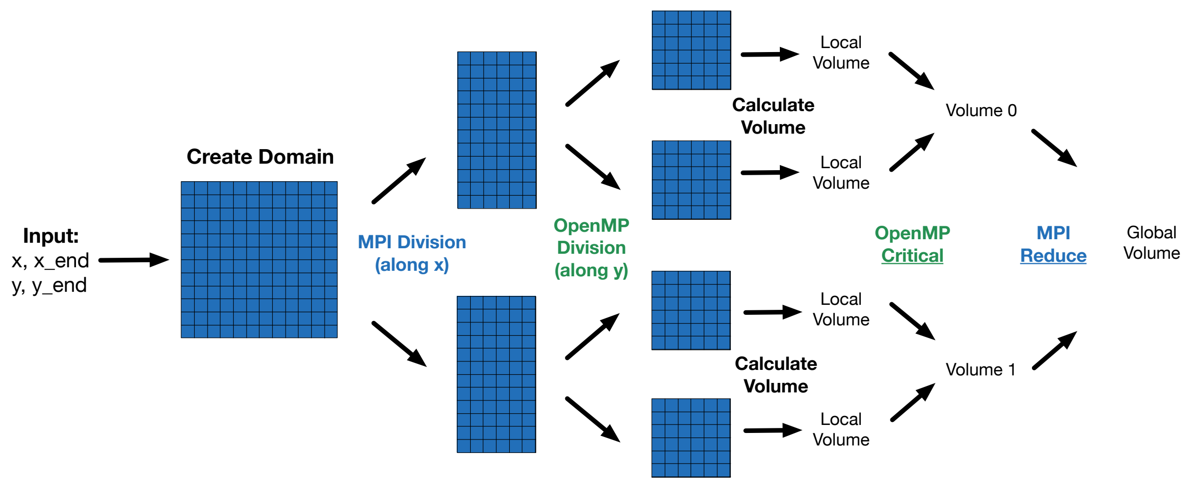

Figure 1: Hybrid MPI/Openmp implementation for Part 1, with MPI=2 and OpenMP=2

Part 2: Monte Carlo Method

The second problem that requires the implementation of the hybrid MPI/Openmp algorithm is optimization using the Monte-Carlo Method. The function that will be analyzed is defined below:

$$f(x) = cos(x) + |7.0 - x|^{2/15} + 2|5.0 - x|^{4/35} \tag{2}$$

The program will start by generate a set of random numbers between the first and last x values. After which the program will loop over the set of x values and calculate for each one the corresponding $f(x)$, and save the lowest one

MPI/Openmp

The hybrid MPI/Openmp implementation for this problem will try to divide the number of points over all the cores. The first thing that the program will do is divide the total number of points over the MPI-Process. So if we requested that the program use 1000 points and we have 4 MPI process then each process will generate 250 random point. After generating the set of random points the program will enter the Openmp enviroment and start the Monte-Carlo Method. Inside the Monte-Carlo method the code will enter the Openmp environment and will divide the set into an even smaller set according to the number of Openmp threads, and find the value for each point. At the end of the Openmp environment the program will save the minimum value. After exiting the Openmp enviroment, the cores will send their minimum value to the MPI-Process of rank 0 so it can calculate the global minimum.

So as a summary, we started with a large set of points, and divid it first into smaller sets for each MPI process, and then divided more into smaller set for each OpenMp thread. The figure gives a better picture of how the code works.

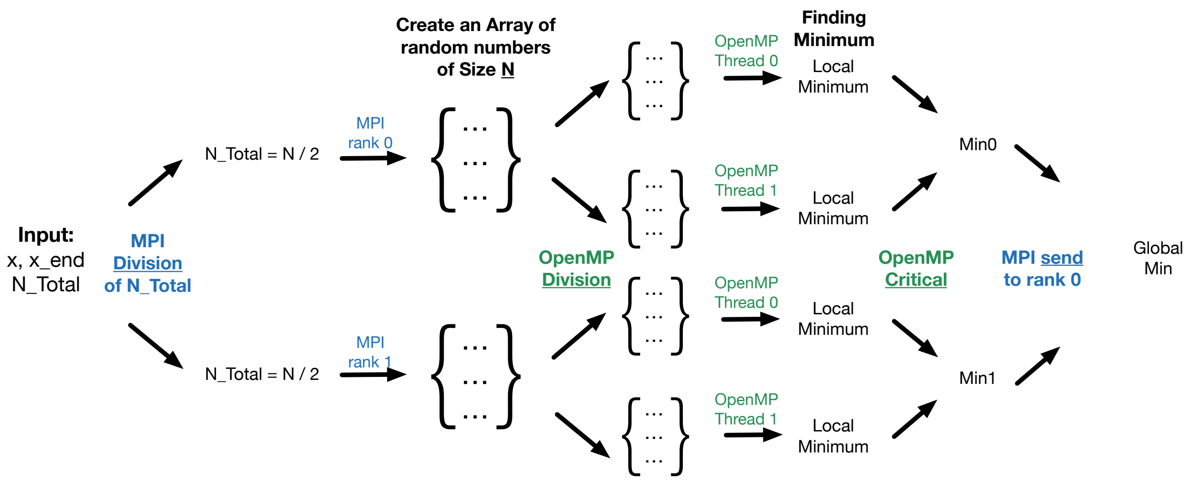

Figure 2: Hybrid MPI/Openmp implementation for Part 2, with MPI=2 and OpenMP=2

Part 3: Compute the standard deviation

The third problem that requires the implementation of the MPI/Openmp algorithm is a program that computes the standard deviation of a set of randomly generated numbers. The equation that governs the standard deviation σ of a probability distribution is defined as the square root of the variance σ^2,

$$\sigma = \sqrt{\langle x^2 \rangle + \langle x \rangle^2} \tag{3}$$

where $\langle x^2 \rangle$ sums of squares and $\langle x \rangle$ is the sum of all the numbers.

MPI/Openmp

The hybrid MPI/Openmp implementation for this problem is similar to the previous problem (Part 2). The program will first divide the total number of points over the MPI-Process. So if we requested that the program use 1000 points and we have 4 MPI process then each process will generate 250 random point. After generating the set of random points the program will enter the Openmp enviroment and start the *Standard Deviation* function. Inside the function the code will enter the Openmp environment and will divide the set into an even smaller set according to the number of Openmp threads. Each Openmp thread will calculate its local value $\sum x^2$ and $\sum x$, and at the end the Openmp will add all the local values together. When the calculation ends, the cores will send their value of $\sum x^2$ and $\sum x$ to rank 0 so it can add them up and find their mean, as follows

$$\langle x \rangle = \frac{\sum_{i=0}^{n=cores} (\sum x)_i}{\text{Total Num of Points}} \quad \text{and} \quad \langle x^2 \rangle = \frac{\sum_{i=0}^{n=cores} (\sum x^2)_i}{\text{Total Num of Points}} \tag{4}$$

finally rank 0 will calculate the standard deviation as follows

$$\sigma = \sqrt{\langle x^2 \rangle + \langle x \rangle^2} \tag{5}$$

Figure gives a better picture of how the code works.

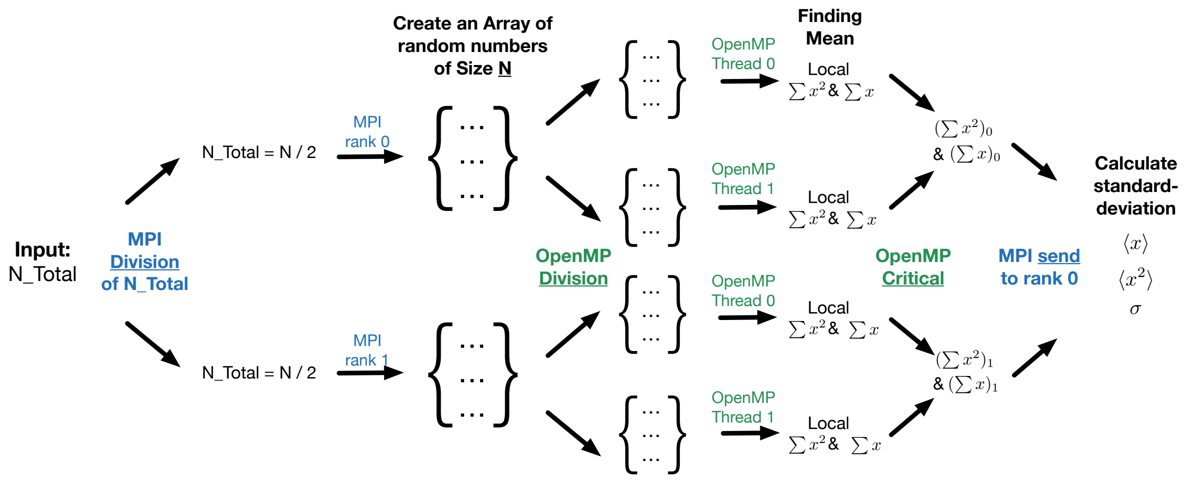

Figure 3: Hybrid MPI/Openmp implementation for Part 3, with MPI=2 and OpenMP=2

Compiling the three programs

The three programs where compiled on beocat using the command:

```
mpicc -O2 -fopenmp -o Part1-MPI-OpenMP Part1-MPI-OpenMP.c -lm
```

where a level 2 optimization was implemented.

Running the Hybrid MPI/OpenMP Programs on Beocat

The implementation of the Hybrid MPI/OpenMP was proved to be difficult. For the case of OpenMP=1 and MPI=N, the implementation was easy, all we had to do is write the following line in the bash script

```
#$ -pe mpi-spread N
mpirun -np N ./program 1 ## 1 for the OpenMP threads
```

An example of how the output should be for Part 1 with MPI=16, and OpenMP=1 is shown below

```
Thread 0 out of 1 from process 0 out of 16 on elf86
Thread 0 out of 1 from process 1 out of 16 on elf93
Thread 0 out of 1 from process 15 out of 16 on elf79
Thread 0 out of 1 from process 2 out of 16 on elf92
Thread 0 out of 1 from process 11 out of 16 on elf90
Thread 0 out of 1 from process 10 out of 16 on elf91
Thread 0 out of 1 from process 5 out of 16 on elf96
Thread 0 out of 1 from process 12 out of 16 on elf89
Thread 0 out of 1 from process 7 out of 16 on elf94
Thread 0 out of 1 from process 9 out of 16 on elf99
Thread 0 out of 1 from process 14 out of 16 on elf87
Thread 0 out of 1 from process 13 out of 16 on elf88
Thread 0 out of 1 from process 3 out of 16 on elf03
Thread 0 out of 1 from process 8 out of 16 on elf05
Thread 0 out of 1 from process 6 out of 16 on elf95
Thread 0 out of 1 from process 4 out of 16 on elf98
```

However for the case with OpenMP $>$ 1 it is mush more difficult. For example let us assume a case with MPI=4 and OpenMP = 4. The first thing we have to do is assign the correct number of cores, this is done by writting the command

```
#$ -pe mpi-4 16
```

This will assign 4 machines, with 4 cores on each machine. The next step is we have to update the host file and make it so that the number of slots on each machine is 1. To do that we added the lines below into the bash file that make use of the command *sed*:

```
sed 's/ 4/ slots=1/g' $PE_HOSTFILE > NEW_HOSTFILE3
sed -i 's/gen-reserved.q@elf...beocat.ksu.edu UNDEFINED//g'
    NEW_HOSTFILE3
```

The first line will search for the number "4" and replace it with "slots=1", and save it in a **new hostfile** called "NEW_HOSTFILE3". As for the second line it will clean the data after the "slots=1" command. So the **new hostfile** will look like

```
elf93.beocat.ksu.edu slots=1
elf92.beocat.ksu.edu slots=1
elf03.beocat.ksu.edu slots=1
elf98.beocat.ksu.edu slots=1
```

while the **old hostfile** looked like

```
elf93.beocat.ksu.edu 4 gen-reserved.q@elf93.beocat.ksu.edu UNDEFINED
elf92.beocat.ksu.edu 4 gen-reserved.q@elf92.beocat.ksu.edu UNDEFINED
elf03.beocat.ksu.edu 4 gen-reserved.q@elf...beocat.ksu.edu UNDEFINED
elf98.beocat.ksu.edu 4 gen-reserved.q@elf...beocat.ksu.edu UNDEFINED
```

After creating the **new hostfile** we will give it as input to the mpi command by typing

```
mpirun --hostfile NEW_HOSTFILE3 -np 4 ./program 4 ## 4 = num OpenMP
```

An example of how the output should be for Part 1 with MPI=4, and OpenMP=4 is shown below

```
Thread 2 out of 4 from process 0 out of 4 on elf93
Thread 3 out of 4 from process 0 out of 4 on elf93
Thread 0 out of 4 from process 0 out of 4 on elf93
Thread 1 out of 4 from process 0 out of 4 on elf93
Thread 0 out of 4 from process 1 out of 4 on elf92
Thread 3 out of 4 from process 1 out of 4 on elf92
Thread 1 out of 4 from process 1 out of 4 on elf92
Thread 2 out of 4 from process 1 out of 4 on elf92
Thread 0 out of 4 from process 3 out of 4 on elf98
Thread 1 out of 4 from process 3 out of 4 on elf98
Thread 2 out of 4 from process 3 out of 4 on elf98
Thread 3 out of 4 from process 3 out of 4 on elf98
Thread 2 out of 4 from process 2 out of 4 on elf03
Thread 3 out of 4 from process 2 out of 4 on elf03
Thread 1 out of 4 from process 2 out of 4 on elf03
Thread 0 out of 4 from process 2 out of 4 on elf03
```

For a look at the full bash script please refer to the appendix

Results

The three problems discussed in the previous section were tested for different numbers of inputs and with different number of cores (1, 2, 4, 8, and 16). The table below summarizes the number of cores used: The maximum OpenMP thread was 8.

Summary of the Cores Used		
Total Nodes	MPI Threads	OpenMP Threads
16	16	1
16	8	2
16	4	4
16	2	8
8	8	1
8	4	2
8	2	4
8	1	8
4	4	1
4	2	2
4	1	4
2	2	1
2	1	2
1	1	1

The results of part 1, 2, and 3 are shown in figures 4, 5 and 6.

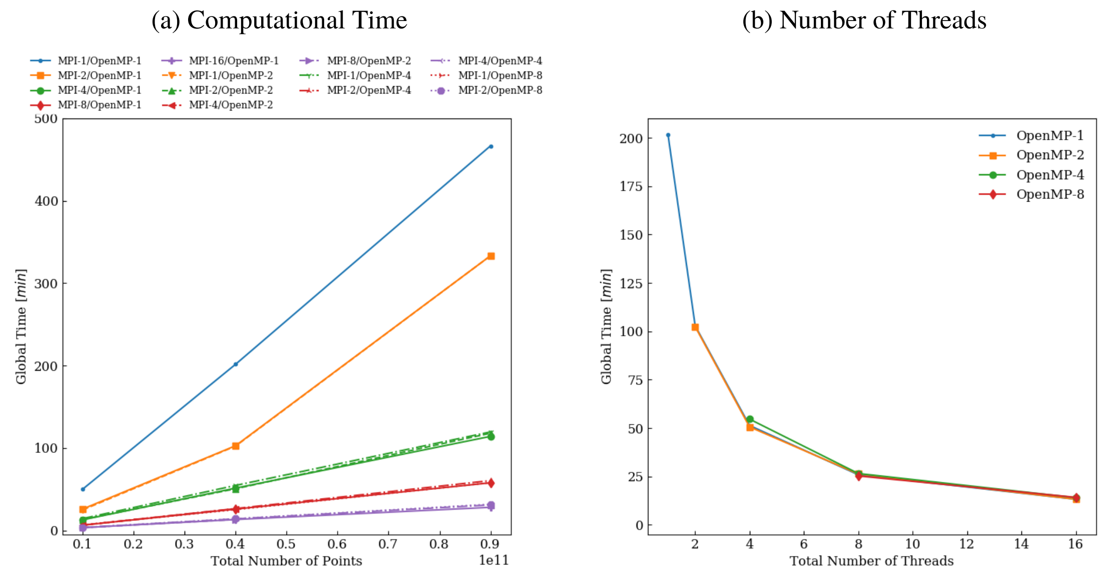

Figure 4: Results of Part 1; (a) Computational time versus number of input, (b) Computational Time versus total number of threads for a single input

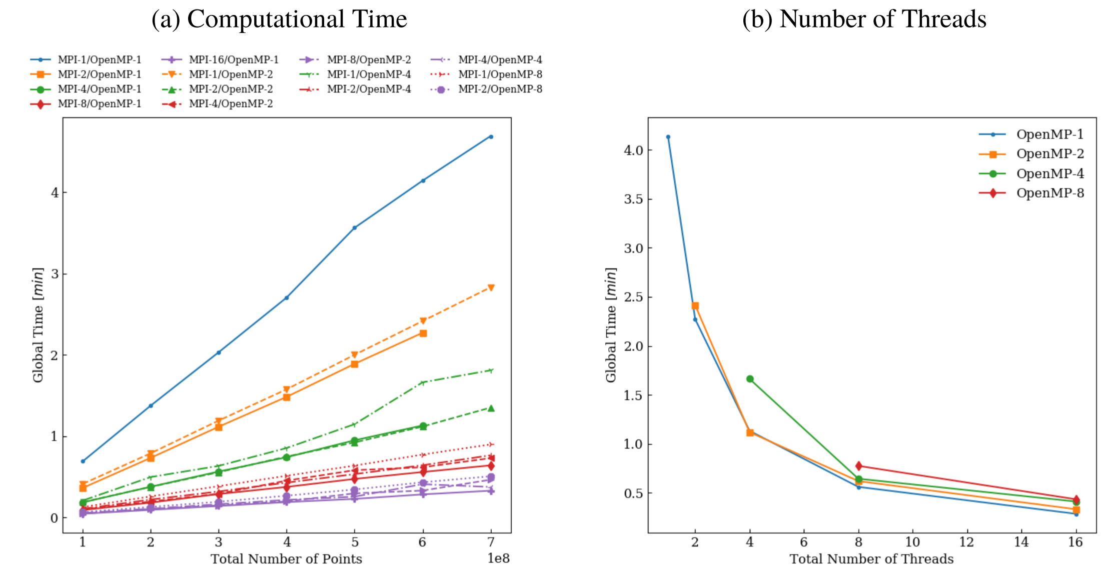

Figure 5: Results of Part 2; (a) Computational time versus number of input, (b) Computational Time versus total number of threads for a single input

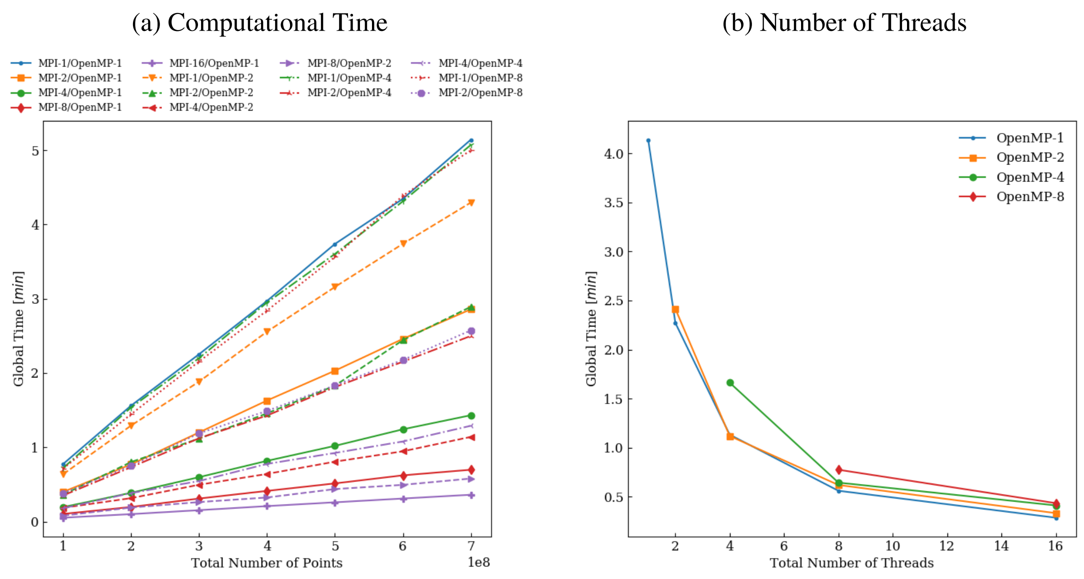

Figure 6: Results of Part 3; (a) Computational time versus number of input, (b) Computational Time versus total number of threads for a single input

From all the results it is clear that the pure MPI was faster than the hybrid OpenMP. At first this was surprising, since OpenMP utilizes the same *Shared memory* which is usually more efficient than the MPI *message passing* , as the latter usually requires increased data movement (moving data from the source to its destination) which is costly both performance-wise and energy-wise.

So to analyze this problem we looked at all the problems, and all of them have one thing in common, they are **extremely parallel**. What I mean is that the program can be easily parallelized without the need to create a difficult network between the threads. Thus the programs can easily divide the task according to the number of cores it has, and solve it without requiring the assistant of any of the other cores. The only time it requires communication is at the beginning, when the input are given, and at the end to sum up the data.

Thus by using the OpenMP enviroment we are increasing the communication time, since the program has to create private variables, solve them, and then combine them together, using a massaging interface. While the pure MPI will solve the problem with less communication. Another drawback of OpenMP is the ”Load balance”, where a barrier is used to force the fastest cores to wait for the slowest cores in the OpenMP region, especially at the critical location.

The only disadvantage of the pure MPI environment is that it the needs a larger memory requirement, then a hybrid MPI/Openmp.

1 References:

1.$https : //support.beocat.ksu.edu/BeocatDocs/index.php/Main_page$
2.$https : //support.beocat.ksu.edu/BeocatDocs/index.php/Compute\_Nodes$
3.W. D. Gropp. Parallel computing and domain decomposition. In Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations, Philadelphia, PA, 1992.
4. B. Chapman, G. Jost, and R. Van Der Pas. Using OpenMP: portable shared memory parallel programming, volume 10. MIT press, 2008.
5. C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. L. Steele Jr, M. E. Zosel, D. E. Ulberg, A. J. Mallinckrodt, S. McKay, et al. The high performance fortran handbook. Computers in Physics, 8(4):428−428, 1994.

2 Appendix:

Listing 1: 'Part1-MPI-OpenMP.c'

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/resource.h>
#include <mpi.h>
#include <time.h>
#include <omp.h>

#define PI 3.14159265

double tstart, ttotal;
double  volume ,global_volume;
int MPI_nthreads = 1, omp_nthreads;
double x,y,x_end,y_end;
char processor_name[20];

/* myclock: (Calculates the time)
--------------------------------------------*/
double myclock()
{
        static time_t t_start = 0;  // Save and subtract off each time

        struct timespec ts;
        clock_gettime(CLOCK_REALTIME, &ts);
        if( t_start == 0 ) t_start = ts.tv_sec;

        return (double) (ts.tv_sec - t_start) + ts.tv_nsec * 1.0e-9;
}


/* Easom's Function
--------------------------------------------*/
double f (double x, double y)
{
  return -cos(x)*sin(y)*exp( -((x - PI)*(x - PI) + (y - PI)*(y - PI)) )
     ;
}


/* Calculare the Integral of the list
--------------------------------------------*/

void * Integrate (void *rank_MPI, void* rank_OpenMP)
{

  double local_volume;
  double h = 0.00001;

  // MPI and OpenMP ID's
  int myMPI_ID =  *((int*) rank_MPI);
  int myOp_ID;

  // Total number of points
  long N_i = (x_end - x) / h;
  long N_j = (y_end - y) / h;

  // Start i and End i controlled by MPI
  int i_start = (myMPI_ID) * (N_i / MPI_nthreads);
  int i_end = i_start + (N_i / MPI_nthreads);

  // Start j and End j controlled by OpenMp
  int j_start, j_end;

  int i,j;

  #pragma omp private(rank_OpenMP,j_start,j_end,local_volume,i,j)
    {

      printf("Thread %d out of %d from process %d out of %d on %s\n",
        omp_get_thread_num(), omp_get_num_threads(), myMPI_ID,
           MPI_nthreads, processor_name);

    myOp_ID =  ((int) rank_OpenMP);

    local_volume = 0.0;

    // Start i and End i controlled by MPI
    j_start = ( myOp_ID) * (N_j / omp_nthreads);
    j_end = j_start + (N_j / omp_nthreads);

        // Initialize Local Integration
    for(i = i_start; i < i_end; i++)
    {
      for(j = j_start; j < j_end; j++)
        {
        local_volume += h*h*f(x + i*h + h/2.0, y + j*h + h/2.0);
        }
    }

    #pragma omp critical
    volume += local_volume;
  }

}

/*  Main
--------------------------------------------*/
int main(int argc, char *argv[])
{
  volume ,global_volume = 0.0, 0.0;
  struct rusage ru;
  // Default Value
  omp_nthreads = 1;
  x , y , x_end ,y_end = 1.0 ,1.0,5.0,5.0;
  if (argc >= 2){
    omp_nthreads = atol(argv[1]);
    sscanf(argv[2],"%lf",&x);
```

```c
    sscanf(argv[3],"%lf",&y);
    sscanf(argv[4],"%lf",&x_end);
    sscanf(argv[5],"%lf",&y_end);
  }

  // Initalize the MPI Enviroment
  int i, rc, rank, namelen;

  MPI_Status Status;
    MPI_Request Request;
  rc = MPI_Init(&argc,&argv);
  if (rc != MPI_SUCCESS){
    printf ("Error starting MPI program. Terminating.\n");
    MPI_Abort(MPI_COMM_WORLD, rc);
  }
    MPI_Comm_size(MPI_COMM_WORLD,&MPI_nthreads); // Number of cores
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);   // rank of each core
  MPI_Get_processor_name(processor_name, &namelen); // Processors name

  // Initalize the Open Mp Enviroment
  omp_set_num_threads(omp_nthreads);

  printf("rank:%d,x0:%lf,x_end:%lf,y0:%lf,y_end:%lf\n",rank,x,x_end,y,
      y_end);

    if (rank ==0)
    {
    printf("Start MPI/OpenMP Process:\n");
      tstart = myclock(); // Global Clock
  }

  // Calculate Integral
  #pragma omp parallel
  {
    Integrate ( &rank, omp_get_thread_num() );
  }


  getrusage(RUSAGE_SELF, &ru);
    long MEMORY_USAGE = ru.ru_maxrss;   // Memory usage in Kb

  MPI_Reduce(&volume, &global_volume, 1, MPI_DOUBLE, MPI_SUM, 0,
      MPI_COMM_WORLD);

  if(rank==0)
  {
    ttotal = myclock() - tstart;
    long N_j = (y_end - y) / 0.00001;
    printf("Part1-MPI-%d/OpenMP-%d:%ld,%lf,%lf,%lf,%lf,%.12lf,%lf,%ld\n
        ",MPI_nthreads,omp_nthreads,N_j,x,x_end,y,y_end,global_volume,
        ttotal,MEMORY_USAGE);
    printf("------------\n");
  }
  MPI_Finalize();

  return 0;
}
```

Listing 2: 'Part2-MPI-OpenMP.c'

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/resource.h>
#include <mpi.h>
#include <time.h>
#include <omp.h>

#define PI 3.14159265

#define MIN(a,b) (((a)<(b))?(a):(b))
#define MAX(a,b) (((a)>(b))?(a):(b))

double tstart, ttotal;
int MPI_nthreads = 1, omp_nthreads;
double x_0,x_end;
long N_Total, N; // Number of points
double min_f;

double *number_array;
/* myclock: (Calculates the time)
---------------------------------------------*/
double myclock()
{
        static time_t t_start = 0;  // Save and subtract off each time

        struct timespec ts;
        clock_gettime(CLOCK_REALTIME, &ts);
        if( t_start == 0 ) t_start = ts.tv_sec;

        return (double) (ts.tv_sec - t_start) + ts.tv_nsec * 1.0e-9;
}


/* Creates a list of Random numbers from x_0 to x_end
---------------------------------------------------------*/
void * Create_Number_Arrays(long n, int myID)
{
  long i;
  srand(time(NULL)+ (int) myID); // randomize seed
  int Z = 0;
  // allocate an array of size n pointers to chars
  number_array = malloc(sizeof(double *)*n);
  for ( i=0; i < n; i++)
  {
    Z = rand()% 1000000;
    number_array[i] = x_0 + (Z/1000000.0)*(x_end - x_0); // random
       number from x_0 to X_end
  }
}

```

```

/* Function that we wish to optimize
---------------------------------------------*/
double f (double x)
{
  return cos(x)+(pow(fabs(7.0-x), 2.0/15.0))+2*(pow(fabs(5.0-x), 4.0/35
      .0));
}


/* Monte Carlo Method
-----------------------*/

void * Monte_Carlo_Method (void* rank_OpenMP)
{

  min_f = f(x_0);
  double local_min_f;

  // OpenMP ID's
  int myOp_ID;

  // Start j and End j controlled by OpenMp
  int i, i_start, i_end;
  double x;

  #pragma omp private(myOp_ID,i_start,i_end,i,local_min_f,x)
    {

    myOp_ID =  ((int) rank_OpenMP);
      local_min_f = f(x_0);

    // Start i and End i controlled by MPI
    i_start = ( myOp_ID) * (N / omp_nthreads);
    i_end = i_start + (N / omp_nthreads);

        // Initialize Local Integration
    for(i = i_start; i < i_end; i++)
    {
      x = number_array[i];
      local_min_f = MIN(f(x),local_min_f);
    }

    #pragma omp critical
    {
      if (local_min_f < min_f)
            min_f = local_min_f;
    }
  }

}

/*  Main
---------------------------------------------*/
int main(int argc, char *argv[])
```

```
{
  struct rusage ru;
  // Default Value
  omp_nthreads = 1;
  x_0 ,x_end = 0.0,10.0;
  N_Total = 10;
  if (argc >= 2){
    omp_nthreads = atol(argv[1]);
    sscanf(argv[2],"%lf",&x_0);
    sscanf(argv[3],"%lf",&x_end);
    sscanf(argv[4],"%ld",&N_Total); // Number of points
  }

  // Initalize the MPI Enviroment
  int i, rc, rank;

  MPI_Status Status;
    MPI_Request Request;
  rc = MPI_Init(&argc,&argv);
  if (rc != MPI_SUCCESS){
    printf ("Error starting MPI program. Terminating.\n");
    MPI_Abort(MPI_COMM_WORLD, rc);
  }
    MPI_Comm_size(MPI_COMM_WORLD,&MPI_nthreads); // Number of cores
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);   // rank of each core

  // Initalize the Open Mp Enviroment
  omp_set_num_threads(omp_nthreads);

    N = N_Total / MPI_nthreads; // number of points for each MPI
        Process

    if (rank ==0)
    {
    printf("Start MPI/OpenMP Process:\n");
      tstart = myclock(); // Global Clock
  }

  // Create the array of random numbers from x_0 to x_end
  Create_Number_Arrays(N,rank);

  // Find the Min using monte carlo for each MPI Process

  #pragma omp parallel
  {
    Monte_Carlo_Method ( omp_get_thread_num() );
  }

  getrusage(RUSAGE_SELF, &ru);
    long MEMORY_USAGE = ru.ru_maxrss;   // Memory usage in Kb

    // Send the data to MPI
  if (rank != 0)
  {
    MPI_Isend(&min_f , 1, MPI_DOUBLE, 0, 1234, MPI_COMM_WORLD, &Request
        );
    printf("Done rank %d\n",rank);
  }

  MPI_Barrier( MPI_COMM_WORLD ) ;

  if (rank == 0)
  {
    double local_min_f=0.0;

    for (i = 1; i< MPI_nthreads; i++)
    {
      MPI_Irecv(&local_min_f, 1, MPI_DOUBLE, i, 1234, MPI_COMM_WORLD, &
         Request);
      min_f = MIN(local_min_f,min_f);
    }

    ttotal = myclock() - tstart;
    printf("Part2-MPI-%d/OpenMP-%d:%ld,%lf,%lf,%ld\n",MPI_nthreads,
       omp_nthreads,N_Total,min_f,ttotal,MEMORY_USAGE);
  }

  MPI_Finalize();
  return 0;
}
```

Listing 3: 'Part3-MPI-OpenMP.c'

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/resource.h>
#include <mpi.h>
#include <time.h>
#include <omp.h>

double tstart, ttotal;
long N, N_total,  sums, sums_squares, sums_recieved,
   sums_squares_recieved;
double stdev;
int MPI_nthreads = 1, omp_nthreads;

int *number_array; // A 1D array of char arrays (a pointer to pointers
   to chars)


/* myclock: (Calculates the time)
---------------------------------------------*/
double myclock()
{
        static time_t t_start = 0;  // Save and subtract off each time

        struct timespec ts;
        clock_gettime(CLOCK_REALTIME, &ts);
        if( t_start == 0 ) t_start = ts.tv_sec;

        return (double) (ts.tv_sec - t_start) + ts.tv_nsec * 1.0e-9;
}


/* Creates a list of Random numbers
--------------------------------------------*/
void * Create_Number_Arrays(int myID)
{
  time_t t;
  long i;
  srand(time(NULL)+ (int) myID); // randomize seed
  // allocate an array of size N pointers to chars
  number_array = malloc(sizeof(int *)*N);

  for ( i=0; i < N; i++)
  {
    number_array[i] = rand() % 1000; // random number from 0 to 1000
  }
  sums = 0;
  sums_squares = 0;
}


// Print the list
void print_arrays()
{
  long i;
  for ( i=0; i < N; i++)
  {
    printf("%d\n",number_array[i] ); // random number from 0 to 1
        million
  }
}


/* Calculare the Sum of the list
--------------------------------------------*/
void * Calculate_Sums(void *myID)
{
  long i;
  long local_sums;
  long local_sums_squares;
  int startPos, endPos;

  #pragma omp private(myID,startPos,endPos,local_sums,
      local_sums_squares,i)
    {
      startPos = ((int) myID) * (N / omp_nthreads);
        endPos = startPos + (N / omp_nthreads);

        // Initialize Local sums
        local_sums = 0;
        local_sums_squares = 0;

    for ( i= startPos; i < endPos; i++)
    {
      local_sums+= number_array[i] ; // random number from 0 to 1
         million
      local_sums_squares += number_array[i]*number_array[i];
    }

    //printf("%d-%ld-%d-%d\n",omp_get_thread_num(),local_sums,startPos,
       endPos);

    // sum up the partial sum into the global sum
    #pragma omp critical
    sums += local_sums;
    sums_squares += local_sums_squares;
  }
}

/* Calculate the Standard Deviation
--------------------------------------------*/
void * Calculate_Standard_Deviation()
{
  stdev =  sqrt( ((double)sums_squares/N_total) - ((double)sums/
     N_total)*((double)sums/ N_total) );
}

/*  Main
--------------------------------------------*/
int main(int argc, char *argv[])
{

  struct rusage ru;
  // Default Value
  N_total    = 10;
  omp_nthreads = 1;
  if (argc >= 2){
    omp_nthreads = atol(argv[1]);
    N_total = atol(argv[2]);
  }

  // Initalize the MPI Enviroment
  int i, rc, rank;

  MPI_Status Status;
    MPI_Request Request;
  rc = MPI_Init(&argc,&argv);
  if (rc != MPI_SUCCESS){
    printf ("Error starting MPI program. Terminating.\n");
    MPI_Abort(MPI_COMM_WORLD, rc);
  }
    MPI_Comm_size(MPI_COMM_WORLD,&MPI_nthreads); // Number of cores
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);   // rank of each core

  // Initalize the Open Mp Enviroment
  omp_set_num_threads(omp_nthreads);

    if (rank ==0)
    {
    printf("Start MPI/OpenMP Process:\n");
      tstart = myclock(); // Global Clock
  }
    // Create Number Arrays
    // -------------------------

  MPI_Bcast(&N_total, 1, MPI_LONG, 0, MPI_COMM_WORLD);

  N = N_total / MPI_nthreads;

  Create_Number_Arrays(rank); // All mpi process will create a random
     list
  //print_arrays();
  // Calculate Sums
  #pragma omp parallel
  {
    Calculate_Sums(omp_get_thread_num());
  }

  getrusage(RUSAGE_SELF, &ru);
    long MEMORY_USAGE = ru.ru_maxrss;   // Memory usage in Kb


  if (rank != 0)
  {
    MPI_Isend(&sums, 1, MPI_LONG, 0, 1234, MPI_COMM_WORLD, &Request);
    MPI_Isend(&sums_squares, 1, MPI_LONG, 0, 5678, MPI_COMM_WORLD, &
       Request);
    printf("Done rank %d\n",rank);
  }
  else
  {

    for (i = 1; i< MPI_nthreads; i++)
    {
      MPI_Irecv(&sums_recieved, 1, MPI_LONG, i, 1234, MPI_COMM_WORLD, &
         Request);
      sums += sums_recieved ;

      MPI_Irecv(&sums_squares_recieved, 1, MPI_LONG, i, 5678,
         MPI_COMM_WORLD, &Request);
      sums_squares += sums_squares_recieved ;
    }

    Calculate_Standard_Deviation();
    ttotal = myclock() - tstart;
    printf("Part3-MPI-%d/OpenMP-%d:%ld,%lf,%lf,%ld\n",MPI_nthreads,
       omp_nthreads,N_total,stdev,ttotal,MEMORY_USAGE);
  }

  MPI_Finalize();
  return 0;
}
```

Listing 4: Part 1 - Bash script for MPI equal 1

```bash
#!/bin/bash
#$ -l mem=5G
#$ -l h_rt=24:00:00
#$ -l killable
#$ -cwd
#$ -P KSU-GEN-RESERVED
#$ -q \*@@elves
#$ -pe single 8

# MPI = 1
for i in 2 3 4
do
mpirun -np 1  /homes/mcheikh/CIS_625/hw5/part-1/Part1-MPI-OpenMP 8 1 1
    $i $i
mpirun -np 1  /homes/mcheikh/CIS_625/hw5/part-1/Part1-MPI-OpenMP 1 1 1
    $i $i

echo -e "-----------DONE-----------\n"
done
```

Listing 5: Part 1 - Bash script for MPI equal 2

```bash
#!/bin/bash
#$ -l mem=5G
#$ -l h_rt=24:00:00
#$ -l killable
#$ -cwd
#$ -P KSU-GEN-RESERVED
#$ -q \*@@elves
#$ -pe mpi-8 16

echo "PE_HOSTFILE:"
echo $PE_HOSTFILE
echo
echo "cat PE_HOSTFILE:"
cat $PE_HOSTFILE
echo '-----'
sed 's/ 8/ slots=1/g' $PE_HOSTFILE > NEW_HOSTFILE0
sed -i 's/gen-reserved.q@elf...beocat.ksu.edu UNDEFINED//g'
    NEW_HOSTFILE0

# OpenMp = 2
for i in 2 3 4
do
mpirun --hostfile NEW_HOSTFILE0 -np 2  /homes/mcheikh/CIS_625/hw5/part
    -1/Part1-MPI-OpenMP 8 1 1 $i $i
mpirun --hostfile NEW_HOSTFILE0 -np 2  /homes/mcheikh/CIS_625/hw5/part
    -1/Part1-MPI-OpenMP 4 1 1 $i $i
mpirun --hostfile NEW_HOSTFILE0 -np 2  /homes/mcheikh/CIS_625/hw5/part
    -1/Part1-MPI-OpenMP 2 1 1 $i $i

echo -e "-----------DONE-----------\n"
done
```

Listing 6: Part 1 - Bash script for OpenMP equal 1

```bash
#!/bin/bash
#$ -l mem=5G
#$ -l h_rt=24:00:00
#$ -l killable
#$ -cwd
#$ -P KSU-GEN-RESERVED
#$ -q \*@@elves
#$ -pe mpi-spread 16

# Openmp 1
for i in 2 3 4
do
mpirun -np 16  /homes/mcheikh/CIS_625/hw5/part-1/Part1-MPI-OpenMP 1 1 1
     $i $i
mpirun -np 8  /homes/mcheikh/CIS_625/hw5/part-1/Part1-MPI-OpenMP 1 1 1
    $i $i
mpirun -np 4  /homes/mcheikh/CIS_625/hw5/part-1/Part1-MPI-OpenMP 1 1 1
    $i $i
mpirun -np 2  /homes/mcheikh/CIS_625/hw5/part-1/Part1-MPI-OpenMP 1 1 1
    $i $i

echo -e "-----------DONE-----------\n"
done
```

Listing 7: Part 1 - Bash script for OpenMP equal 2

```bash
#!/bin/bash
#$ -l mem=5G
#$ -l h_rt=24:00:00
#$ -l killable
#$ -cwd
#$ -P KSU-GEN-RESERVED
#$ -q \*@@elves
#$ -pe mpi-2 16

echo "PE_HOSTFILE:"
echo $PE_HOSTFILE
echo
echo "cat PE_HOSTFILE:"
cat $PE_HOSTFILE
echo '-----'

sed 's/ 2/ slots=1/g' $PE_HOSTFILE > NEW_HOSTFILE2
sed -i 's/gen-reserved.q@elf...beocat.ksu.edu UNDEFINED//g'
    NEW_HOSTFILE2

# OpenMp = 2
for i in 2 3 4
do
mpirun --hostfile NEW_HOSTFILE2 -np 8  /homes/mcheikh/CIS_625/hw5/part
    -1/Part1-MPI-OpenMP 2 1 1 $i $i
mpirun --hostfile NEW_HOSTFILE2 -np 4  /homes/mcheikh/CIS_625/hw5/part
    -1/Part1-MPI-OpenMP 2 1 1 $i $i
mpirun --hostfile NEW_HOSTFILE2 -np 1  /homes/mcheikh/CIS_625/hw5/part
    -1/Part1-MPI-OpenMP 2 1 1 $i $i
```

```

echo -e "-----------DONE-----------\n"
done
```

Listing 8: Part 1 - Bash script for OpenMP equal 4

```
#!/bin/bash
#$ -l mem=5G
#$ -l h_rt=24:00:00
#$ -l killable
#$ -cwd
#$ -P KSU-GEN-RESERVED
#$ -q \*@@elves
#$ -pe mpi-4 16

echo "PE_HOSTFILE:"
echo $PE_HOSTFILE
echo
echo "cat PE_HOSTFILE:"
cat $PE_HOSTFILE
echo '-----'

sed 's/ 4/ slots=1/g' $PE_HOSTFILE > NEW_HOSTFILE3
sed -i 's/gen-reserved.q@elf...beocat.ksu.edu UNDEFINED//g'
    NEW_HOSTFILE3

# OpenMp = 2
for i in 2 3 4
do
mpirun --hostfile NEW_HOSTFILE3 -np 4  /homes/mcheikh/CIS_625/hw5/part
    -1/Part1-MPI-OpenMP 4 1 1 $i $i
mpirun --hostfile NEW_HOSTFILE3 -np 1  /homes/mcheikh/CIS_625/hw5/part
    -1/Part1-MPI-OpenMP 4 1 1 $i $i

echo -e "-----------DONE-----------\n"
done
```