PCAP : Demonstration of MPI & openmp programs

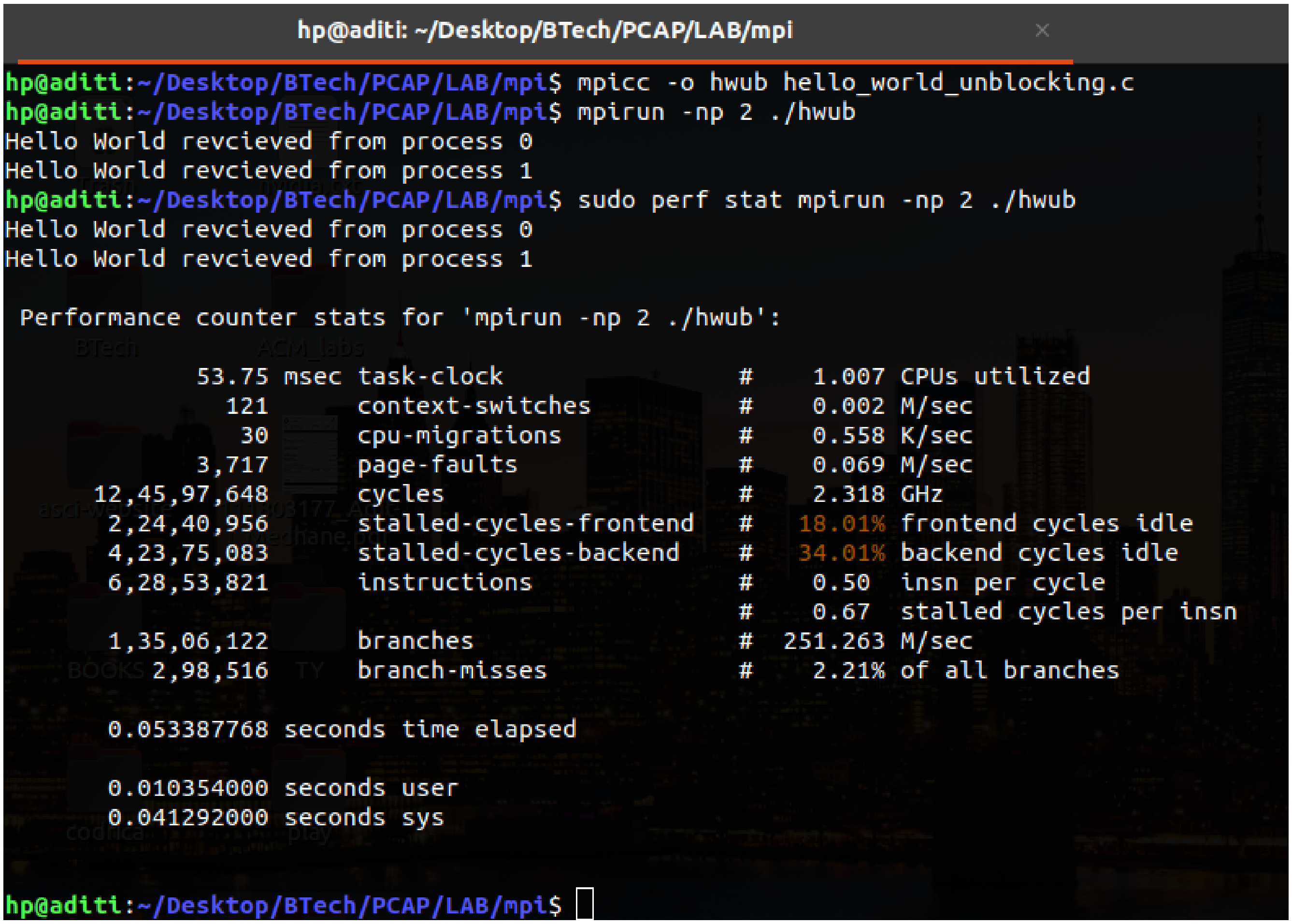
Aditi Rajendra Medhane 111803177

1 : Hello World Non-blocking

Code

```c
#include <stdio.h>
#include <mpi.h>
#include <math.h>
#include <string.h>

int main(int argc, char* argv) {
    MPI_Status status;
    int num;

    MPI_Request request;
    //Initialize MPI computation
    MPI_Init(NULL, NULL);
    //Determine a process's ID number
    MPI_Comm_rank(MPI_COMM_WORLD, &num);

    double d = 100.0;
    char arr[] = "Hello World";

    int tag = 1;

    if(num == 0) {
        MPI_Isend(arr, strlen(arr)+1, MPI_BYTE, 1, tag, MPI_COMM_WORLD, &request);
        MPI_Irecv(arr, strlen(arr)+1, MPI_BYTE, 1, tag, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, &status);
        printf("%s revcieved from process %d\n", arr, num);
    }
    else {
        MPI_Isend(arr, strlen(arr)+1, MPI_BYTE, 0, tag, MPI_COMM_WORLD, &request);
        MPI_Irecv(arr, strlen(arr)+1, MPI_BYTE, 0, tag, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, &status);
        printf("%s revcieved from process %d\n", arr, num);
    }

    MPI_Finalize();
    return 0;
}
```

Output



```
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ mpicc -o hwub hello_world_unblocking.c
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ mpirun -np 2 ./hwub
Hello World revcieved from process 0
Hello World revcieved from process 1
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ sudo perf stat mpirun -np 2 ./hwub
Hello World revcieved from process 0
Hello World revcieved from process 1

 Performance counter stats for 'mpirun -np 2 ./hwub':

             53.75 msec task-clock                #    1.007 CPUs utilized
               121      context-switches          #    0.002 M/sec
                30      cpu-migrations            #    0.558 K/sec
             3,717      page-faults               #    0.069 M/sec
      12,45,97,648      cycles                    #    2.318 GHz
       2,24,40,956      stalled-cycles-frontend   #   18.01% frontend cycles idle
       4,23,75,083      stalled-cycles-backend    #   34.01% backend cycles idle
       6,28,53,821      instructions              #    0.50  insn per cycle
                                                  #    0.67  stalled cycles per insn
       1,35,06,122      branches                  #  251.263 M/sec
          2,98,516      branch-misses             #    2.21% of all branches

       0.053387768 seconds time elapsed

       0.010354000 seconds user
       0.041292000 seconds sys


hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ 
```

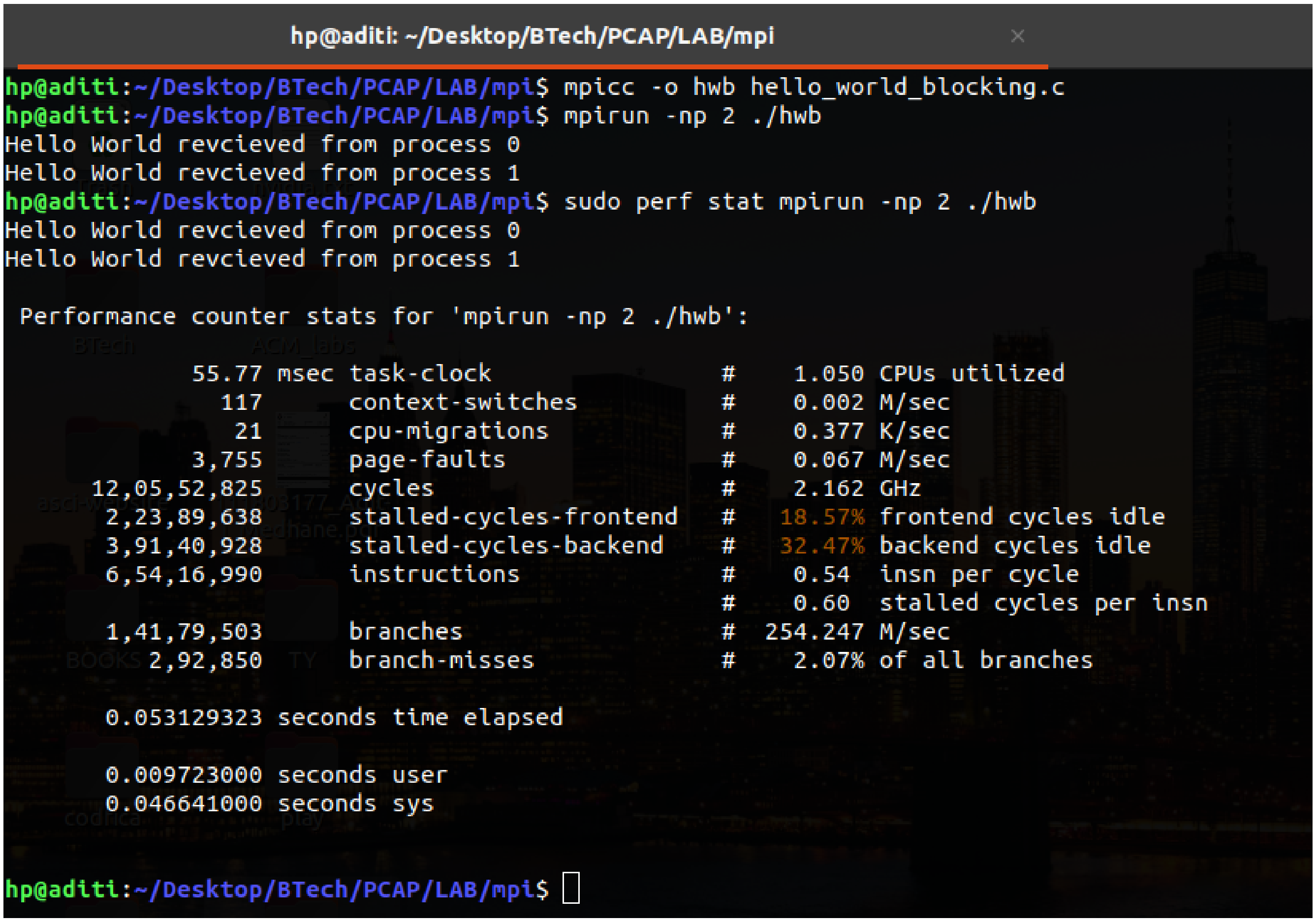
Figure 1: compilation, execution, perf stats of hello world non-blocking

2 : Hello World blocking

Code

```c
#include <stdio.h>
#include <mpi.h>
#include <math.h>
#include <string.h>

int main(int argc, char* argv) {
    MPI_Status status;
    int num;

    //Initialize MPI computation
    MPI_Init(NULL, NULL);
    //Determine a process's ID number
    MPI_Comm_rank(MPI_COMM_WORLD, &num);

    double d = 100.0;
    char arr[] = "Hello World";

    int tag = 1;

    if(num == 0) {
        MPI_Send(arr, strlen(arr)+1, MPI_BYTE, 1, tag, MPI_COMM_WORLD);
        MPI_Recv(arr, strlen(arr)+1, MPI_BYTE, 1, tag, MPI_COMM_WORLD, &status);
        printf("%s revcieved from process %d\n", arr, num);
    }
    else {
        MPI_Send(arr, strlen(arr)+1, MPI_BYTE, 0, tag, MPI_COMM_WORLD);
        MPI_Recv(arr, strlen(arr)+1, MPI_BYTE, 0, tag, MPI_COMM_WORLD, &status);
        printf("%s revcieved from process %d\n", arr, num);
    }

    MPI_Finalize();
    return 0;
}
```

Output



```
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ mpicc -o hwb hello_world_blocking.c
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ mpirun -np 2 ./hwb
Hello World revcieved from process 0
Hello World revcieved from process 1
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ sudo perf stat mpirun -np 2 ./hwb
Hello World revcieved from process 0
Hello World revcieved from process 1

 Performance counter stats for 'mpirun -np 2 ./hwb':

             55.77 msec task-clock                #    1.050 CPUs utilized
               117      context-switches          #    0.002 M/sec
                21      cpu-migrations            #    0.377 K/sec
             3,755      page-faults               #    0.067 M/sec
      12,05,52,825      cycles                    #    2.162 GHz
       2,23,89,638      stalled-cycles-frontend   #   18.57% frontend cycles idle
       3,91,40,928      stalled-cycles-backend    #   32.47% backend cycles idle
       6,54,16,990      instructions              #    0.54  insn per cycle
                                                  #    0.60  stalled cycles per insn
       1,41,79,503      branches                  #  254.247 M/sec
          2,92,850      branch-misses             #    2.07% of all branches

       0.053129323 seconds time elapsed

       0.009723000 seconds user
       0.046641000 seconds sys


hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ 
```

Figure 2: compilation, execution, perf stats of hello world blocking

3 : Trapezium rule Non-blocking

Implements the trapezoidal rule for numerical integration. To approximate the area between the graph of a function, y = f(x), two vertical lines, and the x-axis.

Explanation

Steps cosidered while parallelizing code : 1. Partition the problem solution into tasks. 2. Identify the communication channels between the tasks. 3. Aggregate the tasks into composite tasks. 4. Map the composite tasks to cores.

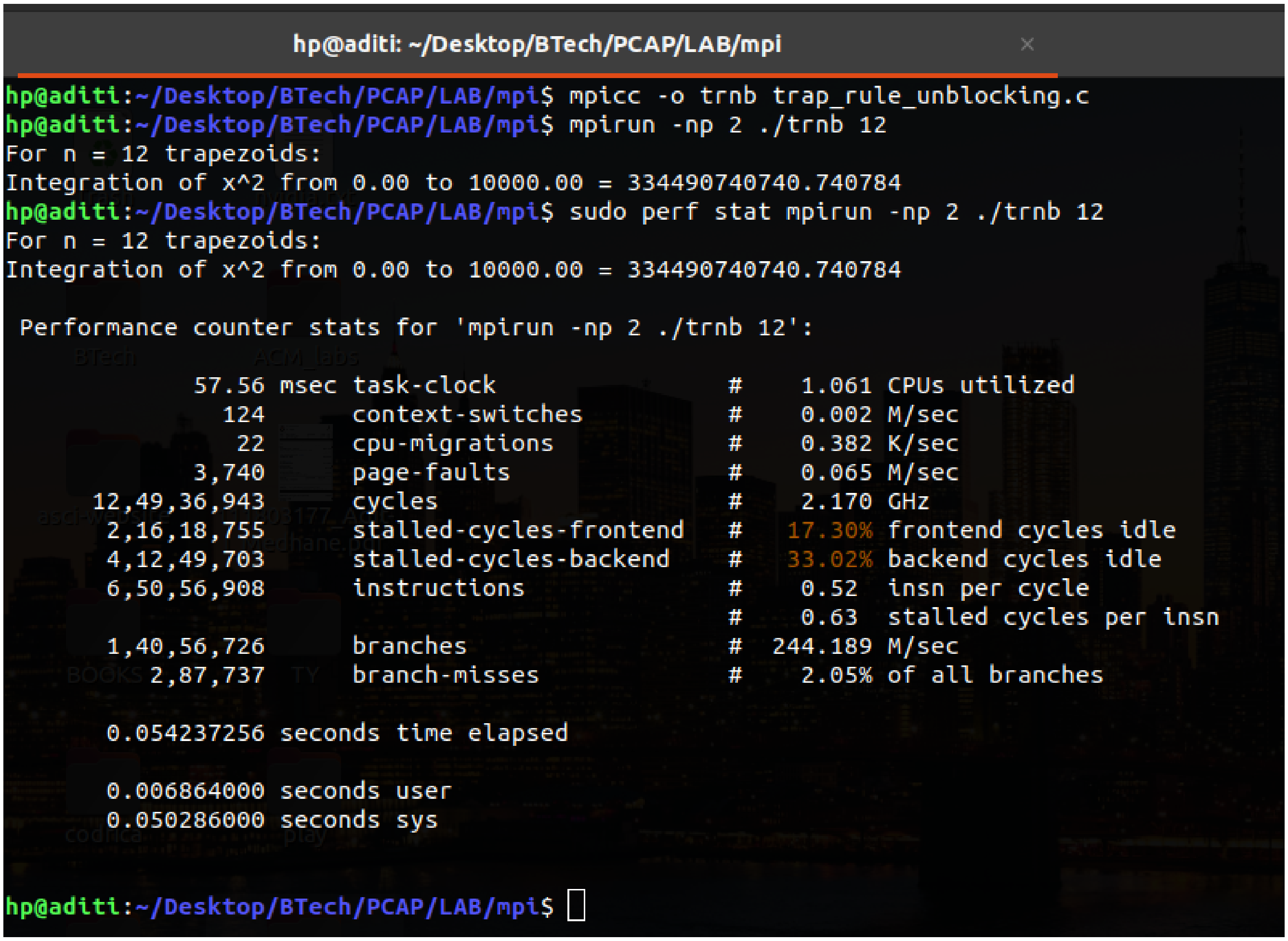
For the trapezoidal rule, two types of tasks will need to consider : 1. finding the area of a single trapezoid. 2. computing the sum of these areas. Then the communication channels will join each of the tasks of the first type to the single task of the second type.

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mpi.h>

const double a = 0;
const double b = 10000;

double trapezoid_area(double left_endpt, double right_endpt, int trap_count, double base_len);
double F(double x);


int main(int argc, char** argv) {
    int rank, size, n_trapezoids, n;
    double x0, x1, h, process_integral, final_integral;
    int source;

    MPI_Request request;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (argc!= 2){
        printf("Enter the command as : mpirun -np <N> %s <number of trapezoids> \n", argv[0]);
        n_trapezoids = -1;
        MPI_Finalize();
        exit(-1);
    }
    else {
        n_trapezoids = atoi(argv[1]);
    }
    MPI_Bcast(&n_trapezoids, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    //For every process, h and n will be same
    h = (b-a)/n_trapezoids;
    n = n_trapezoids/size;

    //For calculating the interval of integration for each process
    x0 = a + rank * n * h;
    x1 = x0 + n * h;

    MPI_Barrier(MPI_COMM_WORLD);

    //calculate integral of each process
    process_integral = trapezoid_area(x0, x1, n, h);

    if (rank != 0) {
        MPI_Isend(&process_integral, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &request);
    }
    else {
        final_integral = process_integral;
            for (source = 1; source < size; source++) {
```

```c
            MPI_Status status;
            MPI_Irecv(&process_integral, 1, MPI_DOUBLE, source, 0, MPI_COMM_WORLD, &request);
            MPI_Wait(&request, &status);
            final_integral += process_integral;
            }

            printf("For n = %d trapezoids:\n", n_trapezoids);
            printf("Integration of x^2 from %0.2f to %0.2f = %f\n", a, b, final_integral);
    }

    MPI_Finalize();

    return 0;

}

double F(double x) {
    return x * x;
}

double trapezoid_area(double left_endpt, double right_endpt, int trapezoid_count, double base_len) {
    double integral, x;
    int i;

    integral = (F(left_endpt) + F(right_endpt))/2.0;
    for (i = 1; i <= trapezoid_count-1; i++) {
        x = left_endpt + i * base_len;
        integral += F(x);
    }
    integral = integral * base_len;

    return integral;
}
```

Output



```
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ mpicc -o trnb trap_rule_unblocking.c
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ mpirun -np 2 ./trnb 12
For n = 12 trapezoids:
Integration of x^2 from 0.00 to 10000.00 = 334490740740.740784
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ sudo perf stat mpirun -np 2 ./trnb 12
For n = 12 trapezoids:
Integration of x^2 from 0.00 to 10000.00 = 334490740740.740784

 Performance counter stats for 'mpirun -np 2 ./trnb 12':

             57.56 msec task-clock                #    1.061 CPUs utilized
               124      context-switches          #    0.002 M/sec
                22      cpu-migrations            #    0.382 K/sec
             3,740      page-faults               #    0.065 M/sec
      12,49,36,943      cycles                    #    2.170 GHz
       2,16,18,755      stalled-cycles-frontend   #   17.30% frontend cycles idle
       4,12,49,703      stalled-cycles-backend    #   33.02% backend cycles idle
       6,50,56,908      instructions              #    0.52  insn per cycle
                                                  #    0.63  stalled cycles per insn
       1,40,56,726      branches                  #  244.189 M/sec
          2,87,737      branch-misses             #    2.05% of all branches

       0.054237256 seconds time elapsed

       0.006864000 seconds user
       0.050286000 seconds sys


hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ 
```

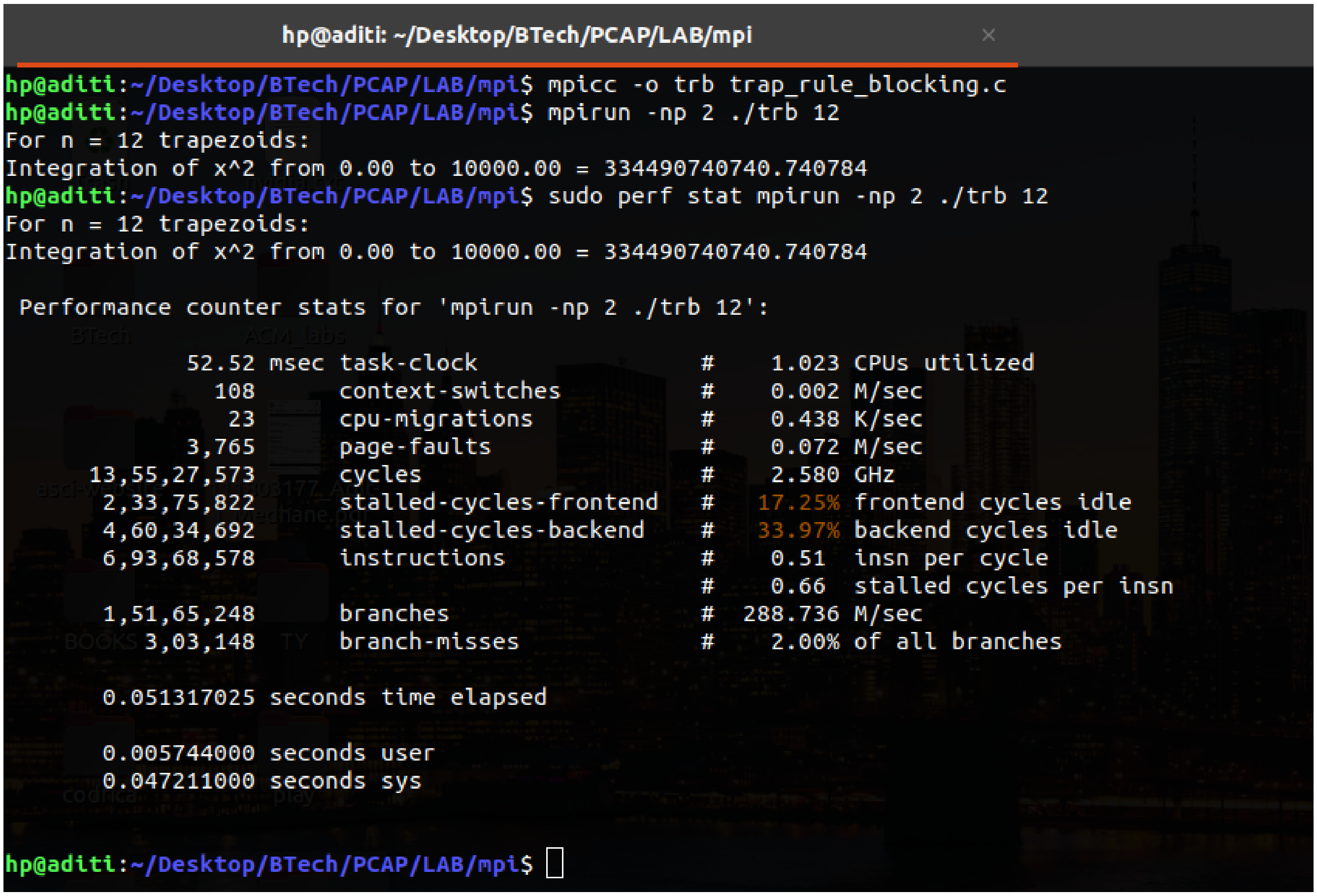
Figure 3: compilation, execution, perf stats of trapezium non-blocking

4 : Trapezium rule blocking

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mpi.h>

const double a = 0;
const double b = 10000;

double trapezoid_area(double left_endpt, double right_endpt, int trap_count, double base_len);
double F(double x);


int main(int argc, char** argv) {
    int rank, size, n_trapezoids, n;
    double x0, x1, h, process_integral, final_integral;
    int source;

    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (argc!= 2){
        printf("Enter the command as : mpirun -np <N> %s <number of trapezoids> \n", argv[0]);
        n_trapezoids = -1;
        MPI_Finalize();
        exit(-1);
    }
    else {
        n_trapezoids = atoi(argv[1]);
    }
    MPI_Bcast(&n_trapezoids, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    //For every process, h and n will be same
    h = (b-a)/n_trapezoids;
    n = n_trapezoids/size;

    //For calculating the interval of integration for each process
    x0 = a + rank * n * h;
    x1 = x0 + n * h;

    MPI_Barrier(MPI_COMM_WORLD);

    //calculate integral of each process
    process_integral = trapezoid_area(x0, x1, n, h);

    if (rank != 0) {
        MPI_Send(&process_integral, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    }
    else {
        final_integral = process_integral;
            for (source = 1; source < size; source++) {
            MPI_Status status;
            MPI_Recv(&process_integral, 1, MPI_DOUBLE, source, 0, MPI_COMM_WORLD, &status);
            final_integral += process_integral;
            }

            printf("For n = %d trapezoids:\n", n_trapezoids);
            printf("Integration of x^2 from %0.2f to %0.2f = %f\n", a, b, final_integral);
    }

    MPI_Finalize();

    return 0;

}

double F(double x) {
    return x * x;
}

double trapezoid_area(double left_endpt, double right_endpt, int trapezoid_count, double base_len) {
    double integral, x;
    int i;

    integral = (F(left_endpt) + F(right_endpt))/2.0;
    for (i = 1; i <= trapezoid_count-1; i++) {
        x = left_endpt + i * base_len;
        integral += F(x);
    }
    integral = integral * base_len;

    return integral;
}
```

Output



```
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ mpicc -o trb trap_rule_blocking.c
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ mpirun -np 2 ./trb 12
For n = 12 trapezoids:
Integration of x^2 from 0.00 to 10000.00 = 334490740740.740784
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ sudo perf stat mpirun -np 2 ./trb 12
For n = 12 trapezoids:
Integration of x^2 from 0.00 to 10000.00 = 334490740740.740784

 Performance counter stats for 'mpirun -np 2 ./trb 12':

             52.52 msec task-clock                #    1.023 CPUs utilized
               108      context-switches          #    0.002 M/sec
                23      cpu-migrations            #    0.438 K/sec
             3,765      page-faults               #    0.072 M/sec
      13,55,27,573      cycles                    #    2.580 GHz
       2,33,75,822      stalled-cycles-frontend   #   17.25% frontend cycles idle
       4,60,34,692      stalled-cycles-backend    #   33.97% backend cycles idle
       6,93,68,578      instructions              #    0.51  insn per cycle
                                                  #    0.66  stalled cycles per insn
       1,51,65,248      branches                  #  288.736 M/sec
          3,03,148      branch-misses             #    2.00% of all branches

       0.051317025 seconds time elapsed

       0.005744000 seconds user
       0.047211000 seconds sys


hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ 
```

Figure 4: compilation, execution, perf stats of trapezium blocking

5 : Matrix Multiplication using MPI

Code

```c
#include <stdio.h>
#include <mpi.h>
#include <math.h>
#include<unistd.h>
#include<stdlib.h>

MPI_Status status;


int main(int argc, char **argv)  {
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if(argc < 2) {
        printf("Enter the correct arguments\n");
        return 1;
    }

    int N = atoi(argv[1]);

    double a[N][N],b[N][N],c[N][N];


    int workers, rows, offset, dest, source, r1=N, c1=N,i,j,k;

    if(rank == 0)  {
        for (i = 0; i < r1; i++) {
            for (j = 0; j < c1; j++) {
                a[i][j] = rand() % 10;
                b[i][j] = rand() % 10;
            }
        }

        workers = size - 1;


        rows = N/workers;
        offset = 0;

        for(dest = 1; dest <= workers; dest++) {
            MPI_Send(&offset, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
            MPI_Send(&rows, 1, MPI_INT, dest, 1, MPI_COMM_WORLD);
            MPI_Send(&a[offset][0], rows*N, MPI_DOUBLE,dest,1, MPI_COMM_WORLD);
            MPI_Send(&b, N*N, MPI_DOUBLE, dest, 1, MPI_COMM_WORLD);
            offset = offset + rows;
        }

        for (i=1; i<=workers; i++) {
            source = i;
            MPI_Recv(&offset, 1, MPI_INT, source, 2, MPI_COMM_WORLD, &status);
            MPI_Recv(&rows, 1, MPI_INT, source, 2, MPI_COMM_WORLD, &status);
            MPI_Recv(&c[offset][0], rows*N, MPI_DOUBLE, source, 2, MPI_COMM_WORLD, &status);
        }


        printf("Matrix multiplication is done\n");
        /*
        for (i=0; i<N; i++) {
            for (j=0; j<N; j++)
                printf("%6.2f   ", c[i][j]);
            printf ("\n");
        }
        */
    }

    if (rank > 0) {
        source = 0;
        MPI_Recv(&offset, 1, MPI_INT, source, 1, MPI_COMM_WORLD, &status);
        MPI_Recv(&rows, 1, MPI_INT, source, 1, MPI_COMM_WORLD, &status);
        MPI_Recv(&a, rows*N, MPI_DOUBLE, source, 1, MPI_COMM_WORLD, &status);
        MPI_Recv(&b, N*N, MPI_DOUBLE, source, 1, MPI_COMM_WORLD, &status);

        for (k=0; k<N; k++) {
            for (i=0; i<rows; i++) {
                c[i][k] = 0.0;
                for (j=0; j<N; j++)
                    c[i][k] = c[i][k] + a[i][j] * b[j][k];
                }
        }

        MPI_Send(&offset, 1, MPI_INT, 0, 2, MPI_COMM_WORLD);
        MPI_Send(&rows, 1, MPI_INT, 0, 2, MPI_COMM_WORLD);
        MPI_Send(&c, rows*N, MPI_DOUBLE, 0, 2, MPI_COMM_WORLD);

    }

    MPI_Finalize();
    return 0;
}
```

Conclusion

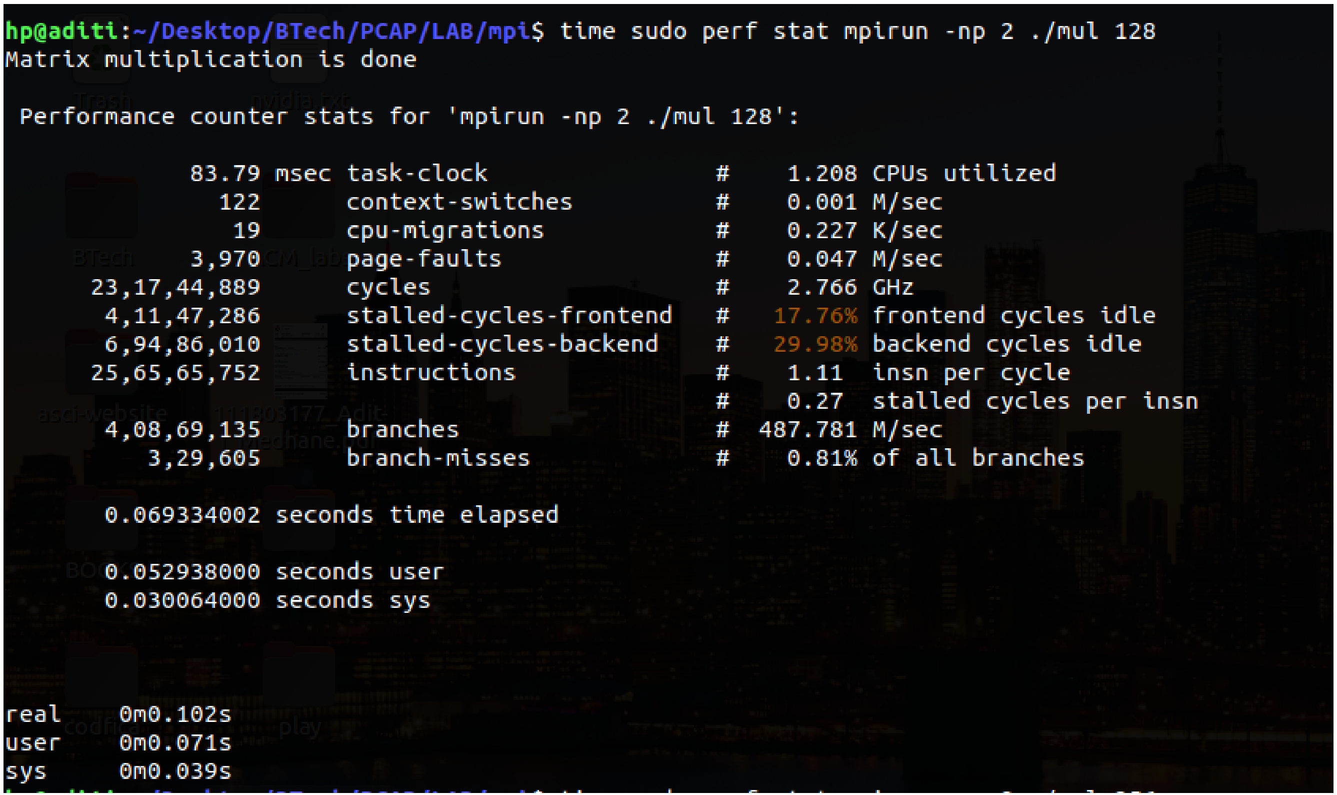
Size | Time Elapsed(MPI) | CPU Utilization(MPI) |

128 0.069979909 seconds 1.208 CPUs utilized

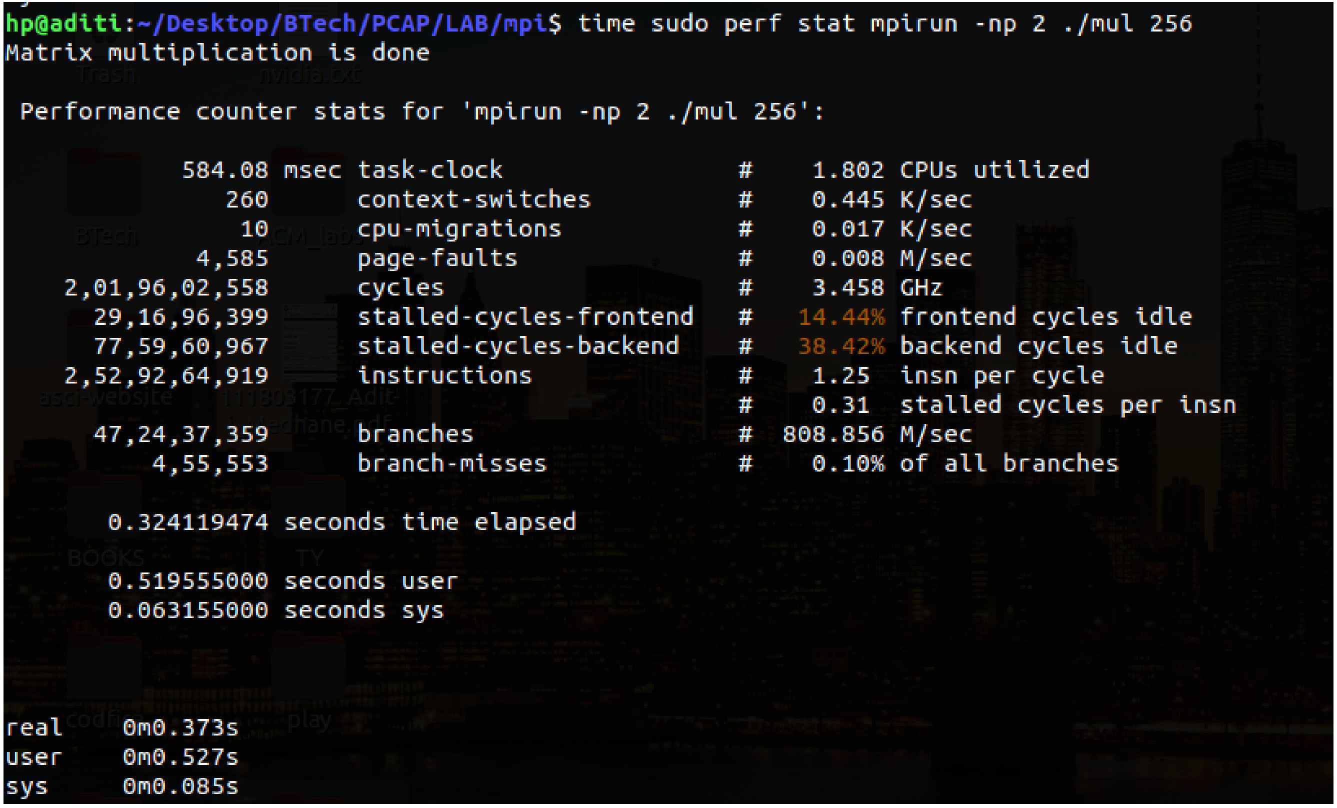
256 0.324119474 seconds 1.802 CPUs utilized

512 6.599078551 seconds 1.945 CPUs utilized

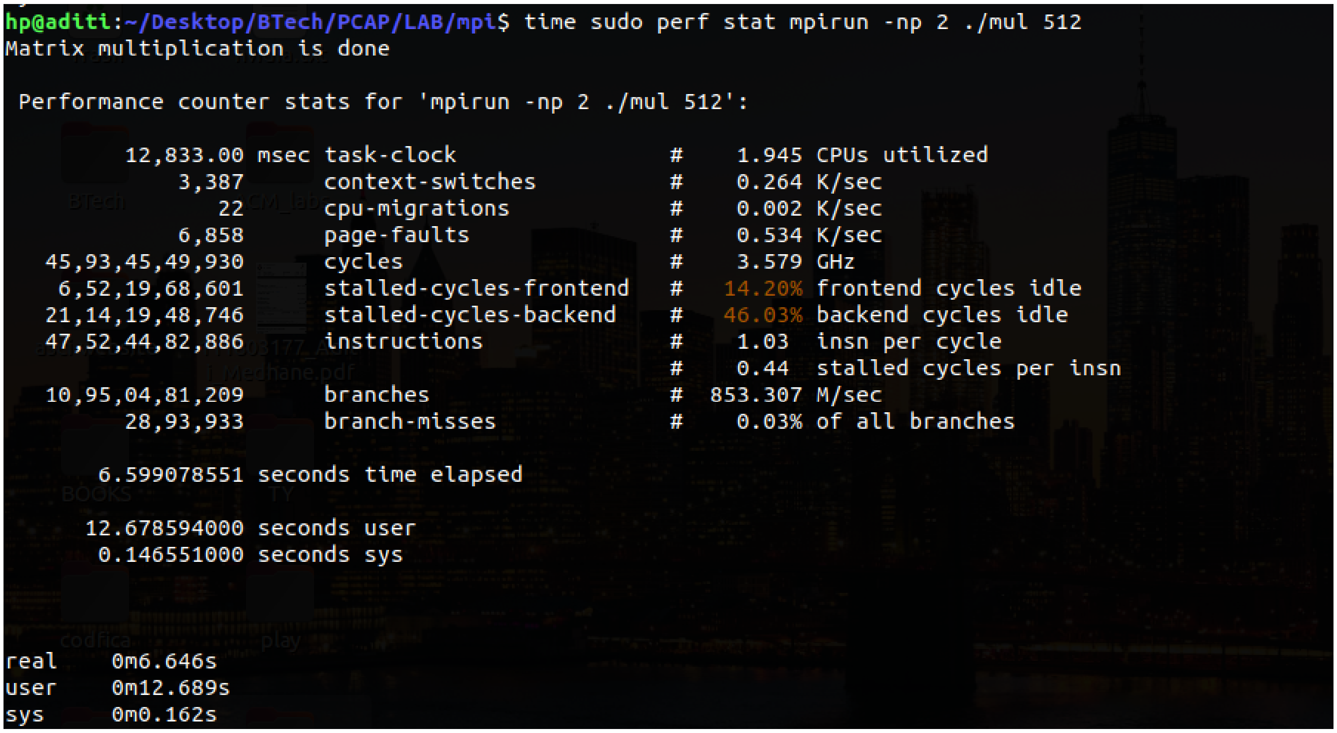
Output



```
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ time sudo perf stat mpirun -np 2 ./mul 128
Matrix multiplication is done

 Performance counter stats for 'mpirun -np 2 ./mul 128':

             83.79 msec task-clock                #    1.208 CPUs utilized
               122      context-switches          #    0.001 M/sec
                19      cpu-migrations            #    0.227 K/sec
             3,970      page-faults               #    0.047 M/sec
       23,17,44,889      cycles                   #    2.766 GHz
        4,11,47,286      stalled-cycles-frontend  #   17.76% frontend cycles idle
        6,94,86,010      stalled-cycles-backend   #   29.98% backend cycles idle
       25,65,65,752      instructions             #    1.11  insn per cycle
                                                  #    0.27  stalled cycles per insn
        4,08,69,135      branches                 #  487.781 M/sec
           3,29,605      branch-misses            #    0.81% of all branches

       0.069334002 seconds time elapsed

       0.052938000 seconds user
       0.030064000 seconds sys



real	0m0.102s
user	0m0.071s
sys	0m0.039s
```



```
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ time sudo perf stat mpirun -np 2 ./mul 256
Matrix multiplication is done

 Performance counter stats for 'mpirun -np 2 ./mul 256':

            584.08 msec task-clock                #    1.802 CPUs utilized
               260      context-switches          #    0.445 K/sec
                10      cpu-migrations            #    0.017 K/sec
             4,585      page-faults               #    0.008 M/sec
    2,01,96,02,558      cycles                    #    3.458 GHz
      29,16,96,399      stalled-cycles-frontend   #   14.44% frontend cycles idle
      77,59,60,967      stalled-cycles-backend    #   38.42% backend cycles idle
    2,52,92,64,919      instructions              #    1.25  insn per cycle
                                                  #    0.31  stalled cycles per insn
      47,24,37,359      branches                  #  808.856 M/sec
          4,55,553      branch-misses             #    0.10% of all branches

       0.324119474 seconds time elapsed

       0.519555000 seconds user
       0.063155000 seconds sys



real	0m0.373s
user	0m0.527s
sys	0m0.085s
```



```
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ time sudo perf stat mpirun -np 2 ./mul 512
Matrix multiplication is done

 Performance counter stats for 'mpirun -np 2 ./mul 512':

         12,833.00 msec task-clock                #    1.945 CPUs utilized
             3,387      context-switches          #    0.264 K/sec
                22      cpu-migrations            #    0.002 K/sec
             6,858      page-faults               #    0.534 K/sec
   45,93,45,49,930      cycles                    #    3.579 GHz
    6,52,19,68,601      stalled-cycles-frontend   #   14.20% frontend cycles idle
   21,14,19,48,746      stalled-cycles-backend    #   46.03% backend cycles idle
   47,52,44,82,886      instructions              #    1.03  insn per cycle
                                                  #    0.44  stalled cycles per insn
   10,95,04,81,209      branches                  #  853.307 M/sec
         28,93,933      branch-misses             #    0.03% of all branches

       6.599078551 seconds time elapsed

      12.678594000 seconds user
       0.146551000 seconds sys


real	0m6.646s
user	0m12.689s
sys	0m0.162s
```

6 : Matrix Multiplication using openmp

Code

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

#define THREADS 16

int main(int argc, char **argv) {
    // Initialize the matrices
    int SIZE = atoi(argv[1]);

    int matrix_a[SIZE][SIZE];
    int matrix_b[SIZE][SIZE];
    int matrix_c[SIZE][SIZE];

    for(int i = 0; i < SIZE; i++) {
        for(int j = 0; j < SIZE; j++) {
            matrix_a[i][j] = rand() % 10;
            matrix_b[i][j] = rand() % 10;
            matrix_c[i][j] = 0;
        }
    }

    omp_set_num_threads(THREADS);
    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        #pragma omp for
            for(int i = id*(SIZE/THREADS); i < (id + 1)*(SIZE/THREADS); i++) {
                for(int c = 0; c < SIZE; c++) {
                    matrix_c[i][c] = 0;
                    for(int k = 0; k < SIZE; k++) {
                        matrix_c[i][c] += matrix_a[i][k]*matrix_b[k][c];
                    }
                }
            }
    }

    return 0;
}
```

Observation

Size | Time Elapsed 128 0.003665319 seconds

256 0.009580259 seconds

512 0.137304077 seconds

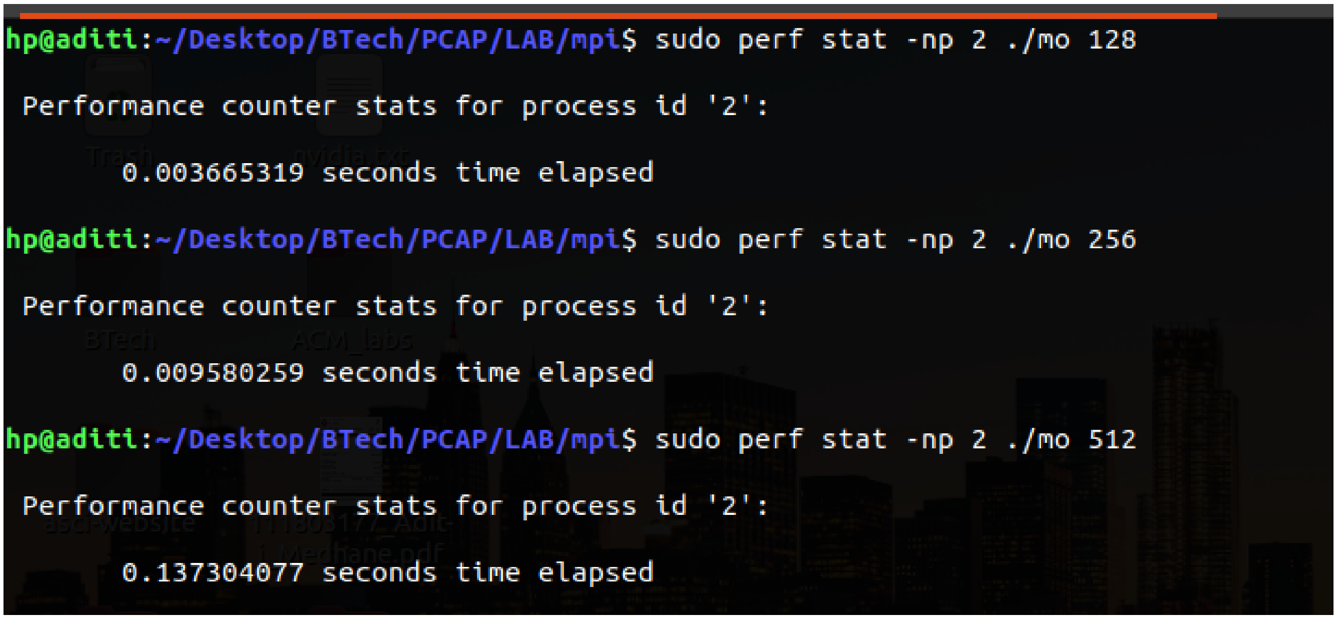
Output

Conclusion

MPI => Internodes Openmp => Intranodes

MPI : Runtime decreases as the numeber of core increases, upto a limit where there is not much improvement.

Openmp : Using, more threads decreases the run time, the desired result, but for small matrix sizes using more number of threads slow down the calculation.



```
hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ sudo perf stat -np 2 ./mo 128

 Performance counter stats for process id '2':

       0.003665319 seconds time elapsed

hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ sudo perf stat -np 2 ./mo 256

 Performance counter stats for process id '2':

       0.009580259 seconds time elapsed

hp@aditi:~/Desktop/BTech/PCAP/LAB/mpi$ sudo perf stat -np 2 ./mo 512

 Performance counter stats for process id '2':

       0.137304077 seconds time elapsed
```

Figure 5: Output