FINAL EXAM

PARALLEL PROCESSING

1. Write a parallel program using MPI that would compute the arithmetic series given by equation (1) above. Your program should read the first term and the number of terms in the arithmetic sequence using the following command line:

mpirun –n <number of processes> ProgramName <first term> <number of terms>

Your program should use cyclic allocation of terms of the arithmetic sequence. Each processor should compute a local sum of the terms allocated to it. The processors should then perform a sum reduction. A way of double-checking the result, processor 0 should also compute and print the value the arithmetic series using equation (2).

/\* File:     mpi\_arithmetic\_series.c

 \* Purpose: To calculate the sum of n terms in the arithmetic series using formula done by master processor of rank 0 and execution of local sum computed by other processes allocated to it

 \*             an = a+(n-1)d

 \*              Sn =

 \*

 \*

 \* Compile:  mpicc -o mpi\_ari mpi\_arithmetic\_series.c

 \*

 \* Run:     mpirun -n <number of processes> mpi\_ari <first\_term> <difference> <number of terms>

               mpirun -n 4 mpi\_arithmetic\_series 2 3 10

 \*           n is the number of terms of the Arithmetic series to use

 \*           a is the first term of the arithmetic series

 \*           d is the difference between two terms in the arithmetic series

 \*

 \* Input:   command line arguments number of processes, first term, difference and number of terms

 \* Output:  sum1 value using the computed formula by processor of rank 0

 \*              sum value using reduction of local sum computation of number of processes given

 \*

 \*/

Source code:

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#define MASTER 0

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

{

int rank, comm\_sz; /\* process tank and number of processes\*/

int i, no\_of\_terms, first\_term, diff\_between\_terms, sum, global\_sum, step, local\_sum\_of\_each\_process; /\*declaration of number of terms, first term,difference between two terms,sum, global sum\*/

int\* a; /\*declaration of pointer to an integer\*/

int source=0, destination=0;

MPI\_Status status;

double start\_time, /\* starting time \*/

end\_time, /\* ending time \*/

computation\_time; /\* time for computing value of PI \*/

MPI\_Init(&argc, &argv);

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

MPI\_Comm\_size(MPI\_COMM\_WORLD, &comm\_sz);

first\_term = atoi(argv[1]);

diff\_between\_terms = atoi(argv[2]);

no\_of\_terms = atoi(argv[3]);

a = (int\*) malloc (no\_of\_terms\*sizeof(int)); /\*dynamic allocation of array a for no\_of\_terms\*/

local\_sum\_of\_each\_process = 0;

global\_sum = 0;

start\_time = MPI\_Wtime();

if(rank==0){

for(i=1; i <= no\_of\_terms; i++) { /\*Generation of arithmetic terms for no\_of\_terms\*/

a[i-1] = first\_term + (i-1)\*diff\_between\_terms;

}

step = 2\*first\_term + (no\_of\_terms-1)\*diff\_between\_terms;

sum = 0.5\*no\_of\_terms\*step;

printf("sum of the arithmetic series computed by the formula is: %d\n", sum );

for(destination = 0; destination < comm\_sz; destination++){

for(i = destination; i < no\_of\_terms; i += comm\_sz){

MPI\_Send(&a[i], 1, MPI\_INT, destination, 1, MPI\_COMM\_WORLD); /\*Sending the terms allocated for each process\*/

}

}

}

MPI\_Bcast(&diff\_between\_terms, 1, MPI\_INT, 0, MPI\_COMM\_WORLD); /\*broadcasting diff\_between\_terms, first\_term, no\_of\_terms to all the processes\*/

MPI\_Bcast(&first\_term, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&no\_of\_terms, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

for(i = rank; i < no\_of\_terms; i += comm\_sz ) /\*Receiving the terms allocated for each process\*/

{

MPI\_Recv(&a[i], 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, &status);

}

for(i = rank; i < no\_of\_terms; i += comm\_sz ) /\*calculation of local sum by each process\*/

{

local\_sum\_of\_each\_process = local\_sum\_of\_each\_process +a[i];

}

printf("local sum computed by rank %d is: %d\n", rank, local\_sum\_of\_each\_process);

MPI\_Reduce(&local\_sum\_of\_each\_process, &global\_sum, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

if(rank == 0){

printf("final sum of the arithmetic series computed by each process is %d\n", global\_sum);

end\_time = MPI\_Wtime();

computation\_time = end\_time - start\_time;

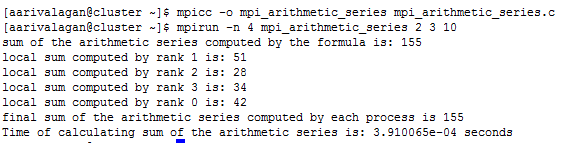
printf("Time of calculating sum of the arithmetic series is: %e seconds\n", computation\_time);

}

MPI\_Finalize();

}

Output:



1. Rewrite the same program using Pthreads, but this time use block allocation of terms of the arithmetic sequence.

/\* File:     pi\_arithmetic\_series..c

 \* Purpose: To calculate the sum of n terms in the arithmetic series using formula done by master processor of rank 0 and recution of local  \*                sum computed by other processes allocated to it

 \*             an = a+(n-1)d

\*             Sn =

 \*

 \*

 \* Compile:  gcc -o pi\_arithmetic\_series pi\_arithmetic\_series.c -lpthread

 \*

 \* Run:     ./pi\_arithmetic\_series <number of processes> <number of terms> <first\_term> <difference>

               ./omp\_ari 4 10 2 3

 \*           number of terms is the number of terms of the Arithmetic series to use

 \*           first term is the first term of the arithmetic series

 \*           difference is the difference between two terms in the arithmetic series

 \*

 \* Input:    command line arguments number of processes, first term, difference and number of terms

 \* Output:  sum1 value using the computed formula by processor of rank 0

 \*              sum value using reduction of local sum computation of number of processes given

 \*

 \*/

Source code:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <pthread.h>

#include "timer.h"

#define BLOCK\_LOW(rank, comm\_sz, number\_of\_terms) ((rank\*number\_of\_terms)/comm\_sz)

#define BLOCK\_HIGH(rank, comm\_sz,number\_of\_terms) (BLOCK\_LOW((rank+1), comm\_sz, number\_of\_terms) - 1)

const int MAX\_THREADS = 1024;

int rank, thread\_count;

int\* a;

int number\_of\_terms, first\_term, diff, sum1, sum, nprocs, step;

pthread\_mutex\_t mutex;

void\* Thread\_sum(void\* rank);

/\* Only executed by main thread \*/

void Get\_args(int argc, char\* argv[]);

void Usage(char\* prog\_name);

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

long thread; /\* Use long in case of a 64-bit system \*/

pthread\_t\* thread\_handles;

double start, finish, elapsed;

int i;

/\* Get number of threads and number of terms from command line \*/

Get\_args(argc, argv);

a = (int\*) malloc (number\_of\_terms\*sizeof(int));

thread\_handles = (pthread\_t\*) malloc (thread\_count\*sizeof(pthread\_t));

pthread\_mutex\_init(&mutex, NULL);

GET\_TIME(start);

sum = 0;

sum1 = 0;

printf("Generated values of the arithmetic series for %d terms \n",number\_of\_terms );

for(i=1; i <= number\_of\_terms; i++){

a[i-1] = first\_term + (i-1)\*diff;

}

for (thread = 0; thread < thread\_count; thread++)

pthread\_create(&thread\_handles[thread], NULL,

Thread\_sum, (void\*)thread);

for (thread = 0; thread < thread\_count; thread++)

pthread\_join(thread\_handles[thread], NULL);

step = 2\*first\_term + (number\_of\_terms-1)\*diff;

sum = 0.5\*number\_of\_terms\*step;

printf("sum is %d\n", sum );

GET\_TIME(finish);

elapsed = finish - start;

printf("With n = %lld terms,\n", number\_of\_terms);

printf(" Computation of sum of the arithmetic series for %d terms using formula = %d\n", number\_of\_terms,sum);

printf(" Computation of sum of the arithmetic series for %d terms by all processess = %d\n", number\_of\_terms,sum1);

printf(" Elapsed time = %e seconds\n", elapsed);

pthread\_mutex\_destroy(&mutex);

free(thread\_handles);

return 0;

} /\* main \*/

/\*------------------------------------------------------------------

\* Function: Thread\_sum

\* Purpose: Add in the terms computed by the thread running this

\* In arg: rank

\* Ret val: ignored

\* Globals in: n, thread\_count

\* Global in/out: sum

\*/

void\* Thread\_sum(void\* rank) {

long my\_rank = (long) rank;

double factor;

int i;

int local\_sum;

int BLOCK\_LOW = (my\_rank \* number\_of\_terms)/thread\_count;

int BLOCK\_HIGH = ((my\_rank+1) \* number\_of\_terms)/thread\_count;

for(i=BLOCK\_LOW; i < BLOCK\_HIGH; i++ )

{

local\_sum = local\_sum + a[i];

}

printf("local sum of rank %d is: %d\n", rank,local\_sum);

pthread\_mutex\_lock(&mutex);

sum1 = sum1 + local\_sum;

pthread\_mutex\_unlock(&mutex);

return NULL;

} /\* Thread\_sum \*/

/\*------------------------------------------------------------------

\* Function: Get\_args

\* Purpose: Get the command line args

\* In args: argc, argv

\* Globals out: thread\_count, n

\*/

void Get\_args(int argc, char\* argv[]) {

if (argc != 5) Usage(argv[0]);

thread\_count = strtol(argv[1], NULL, 10);

if (thread\_count <= 0 || thread\_count > MAX\_THREADS) Usage(argv[0]);

number\_of\_terms = strtoll(argv[2], NULL, 10);

if (number\_of\_terms <= 0) Usage(argv[0]);

first\_term = strtoll(argv[3], NULL, 10);

diff = strtoll(argv[4], NULL, 10);

} /\* Get\_args \*/

/\*------------------------------------------------------------------

\* Function: Usage

\* Purpose: Print a message explaining how to run the program

\* In arg: prog\_name

\*/

void Usage(char\* prog\_name) {

fprintf(stderr, "usage: %s <number of threads> <n>\n", prog\_name);

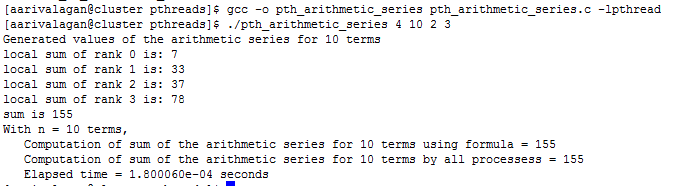
fprintf(stderr, " number\_of\_terms is the number of terms and should be >= 1\n");

fprintf(stderr, " number\_of\_terms should be evenly divisible by the number of threads\n");

exit(0);

} /\* Usage \*/

Output:



1. Repeat question 2 using OpenMP.

/\* File:     omp\_arithmetic\_series..c

 \* Purpose: To calculate the sum of n terms in the arithmetic series using formula done by master processor of rank 0 and execution of local sum computed by other processes allocated to it

 \*             an = a+(n-1)d

\*             Sn =

 \*

 \*

 \* Compile:  gcc -fopenmp -o omp\_arithmetic\_series omp\_arithmetic\_series.c

 \*

 \* Run:     ./omp\_ari <number of processes> <first\_term> <difference> <number of terms>

               ./omp\_ari 4 2 3 10

 \*           n is the number of terms of the Arithmetic series to use

 \*           a is the first term of the arithmetic series

 \*           d is the difference between two terms in the arithmetic series

 \*

 \* Input:    command line arguments number of processes, first term, difference and number of terms

 \* Output:  sum1 value using the computed formula by processor of rank 0

 \*              sum value using reduction of local sum computation of number of processes given

 \*

 \*/

Souce code:

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

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

{

int rank, comm\_sz; /\* process tank and number of processes\*/

int number\_of\_terms, i, first\_term, difference\_between\_terms, sum, step; /\*declaration of number of terms, first term,difference between two terms,sum\*/

int global\_sum = 0; /\*global sum\*/

int local\_sum = 0; /\*local\_sum\*/

i=0;

int\* a;

double time0, time1;

comm\_sz = atoi(argv[1]);

first\_term = atoi(argv[2]);

difference\_between\_terms = atoi(argv[3]);

number\_of\_terms = atoi(argv[4]);

omp\_set\_num\_threads (comm\_sz);

rank = omp\_get\_thread\_num();

a = (int\*) malloc (number\_of\_terms\*sizeof(int)); /\*dynamic allocation of array a for no\_of\_terms\*/

for(i=1; i <= number\_of\_terms; i++){ /\*Generation of arithmetic terms for no\_of\_terms\*/

a[i-1] = first\_term + (i-1)\*difference\_between\_terms;

}

time0 = omp\_get\_wtime();

#pragma omp parallel num\_threads(comm\_sz) shared(number\_of\_terms, a, first\_term, difference\_between\_terms, global\_sum, BLOCK\_LOW, BLOCK\_HIGH) private(i,local\_sum,rank)

{

rank = omp\_get\_thread\_num();

int local\_sum = 0;

#pragma omp parallel for

for(i=rank; i < number\_of\_terms; i += comm\_sz ) /\*calculation of local sum by each process using pragma parallel for functiom\*/

{

local\_sum = local\_sum + a[i];

}

printf("local\_sum of rank %d is : %d\n",rank,local\_sum);

#pragma omp for reduction(+:global\_sum) /\*calculation of global sum by each process using pragma directive with reduction clause\*/

for (i=0; i<comm\_sz; i++) {

global\_sum = global\_sum + local\_sum;

}

}

step = 2\*first\_term + (number\_of\_terms-1)\*difference\_between\_terms;

sum = 0.5\*number\_of\_terms\*step;

printf("sum is %d\n", sum );

time1 = omp\_get\_wtime();

printf("sum of %d terms of the arithmetic series is: %d\n", number\_of\_terms,global\_sum);

printf("Estimate of sum of %d terms of the arithmetic series using the formula: %d\n", number\_of\_terms,sum);

printf("Time in computing the sum of arithmetic series is: %e seconds\n", time1-time0);

}

Output:

