|  |
| --- |
| **Parallel and Distributed Computing (CS3006)** |
| **Course Instructor(s):** |
| Dr. Muhammasd Arshad Islam, Mr. Farrukh Bashir, Mr. Aadil ur Rehman, Mr. Fahad Shafique Ali Ashraf,  **Section(s): (A, B, C, D, E, F, G H, J, K)** |

|  |  |
| --- | --- |
| **Sessional-I Exam** | |
| **Total Time (Hrs):** | **1** |
| **Total Marks:** | **50** |
| **Total Questions:**  **Date:** Feb 22, 2025 | **5** |

**\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Roll No Course Section Student Signature**

**Do not write below this line.**

**Attempt all the questions.**

**[CLO 1: Demonstrate understanding of various concepts involved in parallel and distributed computer architectures.]**

**Question:** Use your understanding of mutexes and ciriticial sections to provide the output of the code.  **[5 marks]**

**#define NUM\_THREADS 4**

**#define ITERATIONS 2**

**int shared\_counter = 0;**

**int thread\_turn = 3;**

**pthread\_mutex\_t lock;**

**pthread\_cond\_t cond;**

**void\* thread\_function(void\* arg) {**

**int tid = \*((int\*)arg);**

**for (int i = 0; i < ITERATIONS; i++) {**

**pthread\_mutex\_lock(&lock);**

**while (tid != thread\_turn) {**

**pthread\_cond\_wait(&cond, &lock);**

**}**

**shared\_counter++;**

**printf("Thread %d (Iteration %d): Counter = %d \n", tid, i + 1, shared\_counter);**

**thread\_turn = (thread\_turn + 1) % NUM\_THREADS;**

**pthread\_cond\_broadcast(&cond);**

**pthread\_mutex\_unlock(&lock);**

**}**

**printf("Thread %d: Completed all iterations.\n", tid);**

**return NULL;**

**}**

**int main() {**

**pthread\_t threads[NUM\_THREADS];**

**int thread\_ids[NUM\_THREADS];**

**pthread\_mutex\_init(&lock, NULL);**

**pthread\_cond\_init(&cond, NULL);**

**for (int i = 0; i < NUM\_THREADS; i++) {**

**thread\_ids[i] = i;**

**pthread\_create(&threads[i], NULL, thread\_function, &thread\_ids[i]);**

**}**

**for (int i = 0; i < NUM\_THREADS; i++) {**

**pthread\_join(threads[i], NULL);**

**}**

**pthread\_mutex\_destroy(&lock);**

**pthread\_cond\_destroy(&cond);**

**printf("\nFinal Value of shared\_counter: %d\n", shared\_counter);**

**return 0;**

**}**

Thread 3 (Iteration 1): Counter = 1

Thread 0 (Iteration 1): Counter = 2

Thread 1 (Iteration 1): Counter = 3

Thread 2 (Iteration 1): Counter = 4

Thread 3 (Iteration 2): Counter = 5

Thread 3: Completed all iterations.

Thread 0 (Iteration 2): Counter = 6

Thread 0: Completed all iterations.

Thread 1 (Iteration 2): Counter = 7

Thread 1: Completed all iterations.

Thread 2 (Iteration 2): Counter = 8

Thread 2: Completed all iterations

Final Value of shared\_counter: 8

**[CLO 1: Demonstrate understanding of various concepts involved in parallel and distributed computer architectures.]**

**Q2:** Following is the incomplete code of adding two matrices using pthreads. Provide two functions that use block and cyclic strategy to add the matrices. **[5+5 marks]**

**const int MAX = 400;**

**const int MAX\_THREAD = 4; // Maximum number of threads**

**int matAdd[MAX][MAX]; // to store results**

**// Function to perform blockwise matrix Addition**

**void Block\_Addition(int matA[][MAX], int matB[][MAX])**

**{**

**int index = step\_add++;**

**// Each thread computes 1/4th of matrix addition**

**int start = index \* (MAX / 4);**

**int end = (index + 1) \* (MAX / 4);**

**for (int i = start; i < end; i++) {**

**for (int j = 0; j < MAX; j++) {**

**matAdd[i][j] = matA[i][j] + matB[i][j];**

**}**

**}**

**}**

**void Cyclic\_Addition(int matA[][MAX], int matB[][MAX])**

**{**

**int tid =pthread\_self();**

**// Each thread processes rows in a cyclic manner**

**for (int i = tid; i < MAX; i += NUM\_THREADS) {**

**for (int j = 0; j < MAX; j++) {**

**matAdd[i][j] = matA[i][j] + matB[i][j];        }**

**}**

**pthread\_exit(NULL);**

**}**

**int main(){**

**int matA[MAX][MAX], matB[MAX][MAX]; /assume matrices are filled**

**thread add\_thread[MAX\_THREAD];**

**for (int i = 0; i < MAX\_THREAD; i++)**

**add\_thread[i] = thread(BlockAddition, matA, matB);**

**for (int i = 0; i < MAX\_THREAD; i++)**

**add\_thread[i] = thread(CyclicAddition, matA, matB);**

**// Waiting for all threads to finish**

**for (int i = 0; i < MAX\_THREAD; i++) {**

**add\_thread[i].join();**

**sub\_thread[i].join();**

**}**

**return 0;**

**}**

**[CLO 1: Demonstrate understanding of various concepts involved in parallel and distributed computer architectures.]**

**Q3: Provide a AVX512/NEON equivalent of the following function. Assume each integer is 32 bit. [10 marks]**

**static const int length = 1024\*8;**

**static float a[length];**

**float scalarAverage() {**

**float sum = 0.0;**

**for (uint32\_t j = 0; j < \_countof(a); ++j) {**

**sum += a[j];**

**}**

**return sum / \_countof(a);**

**}**

float avxAverage () {

\_\_m256 sumx8 = \_mm256\_setzero\_ps();

for (uint32\_t j = 0; j < \_countof(a); j = j + 8) {

sumx8 = \_mm256\_add\_ps(sumx8, \_mm256\_loadu\_ps(&(a[j])));

}

float sum = sumx8.m256\_f32[0] + sumx8.m256\_f32[1] +

sumx8.m256\_f32[2] + sumx8.m256\_f32[3] +

sumx8.m256\_f32[4] + sumx8.m256\_f32[5] +

sumx8.m256\_f32[6] + sumx8.m256\_f32[7];

return sum / \_countof(a);

}

**[CLO 2: 2. Implement different parallel and distributed programming paradigms and algorithms using Message-Passing Interface (MPI) and OpenMP. ]**

**Q4: Calculate Sum of an array using MPI under following preconditions. [15 marks]**

* The size of the array as well as number of processes is unknown at compile time
* Rank=0 will distribute the array among processes
* Use of scatter and gather functions is not allowed.

**int a[] ; // To be summed, assume it is filled**

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

**{**

**int pid, np,elements\_per\_process,n\_elements\_recieved;**

**// np -> no. of processes**

**// pid -> process id**

**MPI\_Status status;**

**MPI\_Init(&argc, &argv);**

**// find out process ID, and how many processes were started**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &pid);**

**MPI\_Comm\_size(MPI\_COMM\_WORLD, &np);**

**// master process**

**if (pid == 0) {**

**int index, i;**

**elements\_per\_process = n / np;**

**int sum = 0;**

**if (np > 1) { // check if more than 1 processes are run**

**//distributes the portion of array to child processes to //calculate their partial sums**

**for (i = 1; i < np - 1; i++) {**

**index = i \* elements\_per\_process;**

**MPI\_Send(&elements\_per\_process,1, MPI\_INT, i, 0,**

**MPI\_COMM\_WORLD);**

**MPI\_Send(&a[index],elements\_per\_process,**

**MPI\_INT, i, 0,MPI\_COMM\_WORLD);**

**}**

**// last process adds remaining elements**

**index = i \* elements\_per\_process;**

**int elements\_left = n - index;**

**MPI\_Send(&elements\_left,1, MPI\_INT,i, 0, MPI\_COMM\_WORLD);**

**MPI\_Send(&a[index],elements\_left,MPI\_INT, i, 0,**

**MPI\_COMM\_WORLD);**

**}**

**// master process add its own sub array**

**for (i = 0; i < elements\_per\_process; i++)**

**sum += a[i];**

**// collects partial sums from other processes**

**int tmp;**

**for (i = 1; i < np; i++) {**

**MPI\_Recv(&tmp, 1, MPI\_INT,MPI\_ANY\_SOURCE, 0, MPI\_COMM\_WORLD, &status);**

**int sender = status.MPI\_SOURCE;**

**sum += tmp;**

**}**

**// prints the final sum of array**

**printf("Sum of array is : %d\n", sum);**

**}**

**else {// slave processes**

**MPI\_Recv(&n\_elements\_recieved,1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD,&status);**

**// stores the received array segment in local array a2**

**MPI\_Recv(&a2, n\_elements\_recieved,MPI\_INT, 0, 0,**

**MPI\_COMM\_WORLD,&status);**

**// calculates its partial sum**

**int partial\_sum = 0;**

**for (int i = 0; i < n\_elements\_recieved; i++)**

**partial\_sum += a2[i];**

**// sends the partial sum to the root process**

**MPI\_Send(&partial\_sum, 1, MPI\_INT,0, 0, MPI\_COMM\_WORLD);**

**}**

**MPI\_Finalize();**

**return 0;**

**}**