Amirkabir University of Technology

(Tehran Polytechnic)

Question 1: Answer the following questions.

a) Explain explicit, implicit, and automatic multi-threading, then write the pros and cons for each of them.

b) Describe false sharing. Can you think of a change to the CPU architecture that would fix this issue? would it improve general performance of the CPU? (explain)

c) Explain First Touch Policy in NUMA architectures and the way it influences parallel programs.

Question 2: The following program has been written by an LLM trying to calculate $\int_{-5}^{5} e^{-x^2} dx$ using OpenMP library. But we're getting inconsistent and wrong results. Fix and improve the performance of the given code and explain the problems is code.

```
#include <iostream>
#include <cmath>
#include <omp.h>

#define NUM_THREADS 8
#define NUM_STEPS 100000

int main() {
    static double step_len = 10.0 / (double) NUM_STEPS;
    double sum[NUM_THREADS] = {0.0};
    double result = 0.0;
    int num_threads;
    #pragma omp parallel num_threads(NUM_THREADS)
    {
        double x;
        int thread_id = omp_get_thread_num();
        num_threads = omp_get_num_threads();
        for (int i = thread_id; i < NUM_STEPS; i += NUM_THREADS) {
            x = (i + 0.5) * step_len - 5.0;
            sum[thread_id] += exp(-(x*x));
        }
    }
    for (int i = 0; i < num_threads; i++)
        result += sum[i] * step_len;
    std::cout << "Result: " << result << std::endl;
    return 0;
}
```

Question 3: Improve the code snippets below to enhance performance and reduce overheads and explain the reason for changes in each section. (Assume that thread pool is not enabled.)

a)

```
int M, N;
int a[M], b[N];
#pragma omp parallel for
for (int i = 0; i < m; i++)
    a[i] = rand();
#pragma omp parallel for
for (int i = 0; i < N; i++)
    b[i] = i;
```

b)

```
unsigned short nums[1000000];
unsigned long buckets[256] = {0};
#pragma omp parallel for
for (int i = 0; i < 1000000; i++) {
#pragma omp critical
    buckets[nums[i] % 256]++;
}
```

c) C

```
int N, sum = 0;
int a[N], b[N], c[N], d[N];
#pragma omp parallel for
for (int i = 0; i < N; i++)
    a[i] += b[i];
#pragma omp parallel for
for (int i = 0; i < N; i++)
    c[i] += d[i];
#pragma omp parallel for reduction(+:sum)
for (int i = 0; i < N; i++)
    sum += a[i] + c[i];
```

d)

```
int N;
int a[N];
double sum = 0;
#pragma omp parallel for
for (int i = 0; i < N; i++) {
#pragma omp critical
    sum += log2(a[i]) * log2(a[i]);
}
```

e) (This section has double the points of other sections in this question)

```
int N;
int mat[N][N];
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
#pragma omp parallel for
        for (int k = 0; k < N; k++)
#pragma omp critical
            mat[i][j] += mat[(i+k)%N][(j+k)%N]
```

Question 4: The following function convolves the input 2D matrix with a 3x3 kernel and performs 2x2 pooling without using any extra dynamic (based on matrix size) memory allocation. For simplicity assume that matrix size is always even.

```
void conv_pool(int** mat, int** kernel, int N){
    int sum;
    for (int i = 1; i < N-1; i++){
        for (int j = 1; j < N-1; j++) {
            sum = 0;
            for (int k = -1; k < 2; k++)
                for (int l = -1; l < 2; l++)
                    sum += mat[i+k][j+l]
            mat[i-1][j-1] = sum;
        }
    }

    for (int i = 0; i < N-2; i+=2){
        for (int j = 0; j < N-2; j+=2) {
            a[i/2][j/2] = (a[i][j] + a[i+1][j] + a[i][j+1] + a[i+1][j+1]) / 4;
        }
    }

    // equivalent operation: mat = mat[0:(N-2)/2][0:(N-2)/2]
    resize_mat(mat, 0, (N-2)/2, 0, (N-2)/2);

    return;
}
```

a) Using OpenMP parallelize this function and compare total memory usage in the serial and parallel versions. To simplify decomposition, assume that the number of threads is a square number and matrix size is an even multiple of square root of thread count.

b) (Bonus Question) Is it possible to parallelize this function without use of any extra dynamic memory allocation? If yes, Implement the code otherwise prove why it cannot be done.