#include <stdio.h>  
#include <stdlib.h>  
#include <sys/time.h>  
#include <time.h>  
#include <math.h>  
#include <string.h>  
#ifdef \_OPENMP  
#include <omp.h>  
#include <unistd.h>  
void print\_progress(int \*progress) {  
 while (\*progress < 100) {  
 printf("Progress: %d%%\n", \*progress);  
 sleep(1);  
 }  
 printf("Progress: 100%%\n");  
}  
#endif  
  
int main(int argc, char\* argv[]) {  
  
 int i, N, A;  
 N = atoi(argv[1]);  
 A = atoi(argv[2]);  
 char\* schedule\_type = argv[4];  
 int chunk\_size = atoi(argv[5]);  
  
#ifdef \_OPENMP  
 omp\_sched\_t schedule\_kind;  
  
 omp\_set\_num\_threads(atoi(argv[3]));  
  
 if (strcmp(schedule\_type, "static") == 0) {  
 schedule\_kind = omp\_sched\_static;  
 } else if (strcmp(schedule\_type, "dynamic") == 0) {  
 schedule\_kind = omp\_sched\_dynamic;  
 } else if (strcmp(schedule\_type, "guided") == 0) {  
 schedule\_kind = omp\_sched\_guided;  
 } else {  
 fprintf(stderr, "Ошибка: Неверный тип schedule: %s\n", schedule\_type);  
 return 1;  
 }  
 omp\_set\_schedule(schedule\_kind, chunk\_size);  
#endif  
#ifdef \_OPENMP  
 double start\_time, end\_time;  
 start\_time = omp\_get\_wtime();  
#else  
 printf("Программа выполняется последовательно");  
 struct timeval T1, T2;  
 long delta\_ms;  
 gettimeofday(&T1, NULL);  
#endif  
 //Этап 1  
 double \*M1 = malloc(sizeof(double)\* N);  
 double \*M2 = malloc(sizeof(double)\* N/2);  
 unsigned int seed = time(NULL);  
#ifdef \_OPENMP  
 int progress = 0;  
#pragma omp parallel sections  
 {  
#pragma omp section  
 {  
 print\_progress(&progress);  
 }  
#pragma omp section  
 {  
#endif  
 for (i=0; i<100; i++) {  
 srand (i);  
#ifdef \_OPENMP  
 #pragma omp atomic write  
 progress = i;  
#endif  
 // Заполнение M1  
 for (int i = 0; i < N; i++) {  
 M1[i] = 1.0 + (A - 1.0) \* ((double)rand\_r(&seed) / RAND\_MAX);  
 }  
 // Заполнение M2  
 for (int i = 0; i < N / 2; i++) {  
 M2[i] = A + (10 \* A - A) \* ((double)rand\_r(&seed) / RAND\_MAX);  
 }  
#ifdef \_OPENMP  
#pragma omp parallel for schedule(runtime) default(none) private(i) shared(N, M1)  
#endif  
 // Этап 2: Map (гиперболический синус в квадрат)  
 for (int i = 0; i < N; i++) {  
 M1[i] = pow(sinh(M1[i]), 2);  
 }  
 // Создаем копию M2  
 double M2\_copy[N / 2];  
#ifdef \_OPENMP  
#pragma omp parallel for schedule(runtime) default(none) private(i) shared(N, M2\_copy, M2)  
#endif  
 for (int i = 0; i < N / 2; i++) {  
 M2\_copy[i] = M2[i];  
 }  
#ifdef \_OPENMP  
#pragma omp parallel for schedule(runtime) default(none) private(i) shared(N, M2, M2\_copy)  
#endif  
 // Этап 2: Map для M2 (кумулятивная сумма + sqrt(e \* сумма))  
 for (int i = 0; i < N / 2; i++) {  
 double prev = (i == 0) ? 0 : M2\_copy[i - 1]; // Предыдущий элемент или 0  
 M2[i] = sqrt(M\_E \* (M2\_copy[i] + prev));  
 }  
#ifdef \_OPENMP  
#pragma omp parallel for schedule(runtime) default(none) private(i) shared(M2, N, M1)  
#endif  
 // Этап 3: Merge (умножение элементов M2 и M1 с одинаковыми индексами)  
 for (int i = 0; i < N / 2; i++) {  
 M2[i] = M2[i] \* M1[i]; // Умножение  
 }  
 // Этап 4: Sort (сортировка вставками для M2) // распараллеленная  
#ifdef \_OPENMP  
 #pragma omp parallel sections  
 {  
#pragma omp section  
 {  
 // Сортировка первой половины (M2[0:N/4])  
 for (int i = 1; i < N / 4; i++) {  
 double key = M2[i];  
 int j = i - 1;  
 while (j >= 0 && M2[j] > key) {  
 M2[j + 1] = M2[j];  
 j = j - 1;  
 }  
 M2[j + 1] = key;  
 }  
 }  
  
#pragma omp section  
 {  
 // Сортировка второй четверти (M2[N/4:N/2])  
 for (int i = N / 4; i < N / 2; i++) {  
 double key = M2[i];  
 int j = i - 1;  
 while (j >= N / 4 && M2[j] > key) {  
 M2[j + 1] = M2[j];  
 j = j - 1;  
 }  
 M2[j + 1] = key;  
 }  
 }  
 }  
  
 // Однопоточное слияние  
 double merged[N / 2];  
 int left = 0, right = N / 4, k = 0;  
  
 while (left < N / 4 && right < N / 2) {  
 if (M2[left] < M2[right])  
 merged[k++] = M2[left++];  
 else  
 merged[k++] = M2[right++];  
 }  
 while (left < N / 4)  
 merged[k++] = M2[left++];  
 while (right < N / 2)  
 merged[k++] = M2[right++];  
  
 memcpy(M2, merged, sizeof(double) \* (N / 2)); // Копируем  
#else  
 // Этап 4: Sort (сортировка вставками для M2)  
 for (int i = 1; i < N / 2; i++) {  
 double key = M2[i];  
 int j = i - 1;  
  
 // Сдвиг элементов массива, которые больше ключа  
 while (j >= 0 && M2[j] > key) {  
 M2[j + 1] = M2[j];  
 j = j - 1;  
 }  
  
 M2[j + 1] = key;  
 }  
#endif  
 // Этап 5: Reduce (расчет суммы синусов для определенных элементов)  
 double min\_value = M2[0];  
#ifdef \_OPENMP  
#pragma omp parallel for schedule(runtime) default(none) private(i) shared(min\_value, N, M2)  
#endif  
 // Поиск минимального ненулевого элемента в M2  
 for (int i = 1; i < N / 2; i++) {  
 if (M2[i] != 0 && M2[i] < min\_value) {  
 min\_value = M2[i];  
 }  
 }  
  
 // Расчет суммы синусов  
 double X = 0.0;  
#ifdef \_OPENMP  
#pragma omp parallel for schedule(runtime) default(none) private(i) shared(X, N, min\_value, M2)  
#endif  
 for (int i = 0; i < N / 2; i++) {  
 // Проверка, делится ли M2[i] на минимальный элемент с четной целой частью  
 if ((int)(M2[i] / min\_value) % 2 == 0) {  
 X += sin(M2[i]);  
 }  
 }  
 }  
#ifdef \_OPENMP  
}  
progress = 100;  
}  
#endif  
  
 free(M1); free(M2);  
#ifdef \_OPENMP  
 end\_time = omp\_get\_wtime();  
 printf("\nN=%d. Milliseconds passed: %lf\n", N, (end\_time - start\_time) \* 1000);  
#else  
 gettimeofday(&T2, NULL);  
 delta\_ms = (T2.tv\_sec - T1.tv\_sec) \* 1000 +  
 (T2.tv\_usec - T1.tv\_usec) / 1000;  
 printf ("\nN=%d. Milliseconds passed: %ld\n", N, delta\_ms);  
#endif  
 return 0;  
}