**T02: OpenMP - Write Sum of N - Double precision Floating point Numbers**

**CS24M1005 – SINDHIYA R**

**Write OpenMP Parallel Code for Sum of N - Double Precision Floating Point Numbers. Give input very large at least 1 million - You can dump larger double precision values in a file and read from it and perform addition.**

**CODE FOR GENERATING INPUT:**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <cstdlib>  #include <ctime>  using namespace std;  #define N 1000000 // At least 1 million numbers  int main() {  ofstream fout("data.txt");  srand(time(0));  for (int i = 0; i < N; i++) {  fout << (double)rand() / RAND\_MAX \* 1000000.0 << "\n";  }  fout.close();  cout << "Data file generated with " << N << " numbers.\n";  return 0;  } |

**1) Parallel Code Using Reduction Construct (5 Marks)**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <vector>  #include <omp.h>  using namespace std;  #define N 1000000  int main() {  vector<double> numbers(N);  ifstream fin("data.txt");  for (int i = 0; i < N; i++)  fin >> numbers[i];  fin.close();  double sum = 0.0;  double start\_time = omp\_get\_wtime();    #pragma omp parallel for reduction(+:sum)  for (int i = 0; i < N; i++) {  sum += numbers[i];  }    double end\_time = omp\_get\_wtime();  cout << "Parallel Sum (Reduction) = " << sum << " Time = " << (end\_time - start\_time) << " sec\n";  return 0;  } |

**2) Parallel Code Using Critical Section (5 Marks)**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <vector>  #include <omp.h>  using namespace std;  #define N 1000000  int main() {  vector<double> numbers(N);  ifstream fin("data.txt");  for (int i = 0; i < N; i++)  fin >> numbers[i];  fin.close();  double sum = 0.0;  double start\_time = omp\_get\_wtime();  #pragma omp parallel  {  double local\_sum = 0.0;  #pragma omp for  for (int i = 0; i < N; i++) {  local\_sum += numbers[i];  }  #pragma omp critical  sum += local\_sum;  }  double end\_time = omp\_get\_wtime();  cout << "Parallel Sum (Critical) = " << sum << " Time = " << (end\_time - start\_time) << " sec\n";  return 0;  } |

**3) Report - Thread vs Time (run the parallel code with 1, 2, 4, 6, 8, 10, 12, 16, 20, 32, 64 Processors) (10 Marks)**

|  |  |  |  |
| --- | --- | --- | --- |
| **P** | **T(Pr)** | | **T(Pc)** |
| 1 | 0.00325322 | | 0.0020332 |
| 2 | 0.00182202 | | 0.001141 |
| 4 | 0.0147369 | | 0.0151883 |
| 6 | 0.0205372 | | 0.0150664 |
| 8 | 0.0158424 | | 0.0269291 |
| 10 | 0.0178276 | | 0.0269291 |
| 12 | 0.0214693 | | 0.0218793 |
| 16 | 0.0182642 | | 0.0359069 |
| 20 | 0.00229795 | | 0.0015131 |
| 32 | 0.00250355 | | 0.0021038 |
| 64 | 0.00467896 | | 0.0030895 |
| **In above table,**  **P** means number of Threads  **T(Pr)** means reduction time (in seconds)  **T(Pc)** means critical time (in seconds) | |

**4) Plot Speedup vs Processors (5 Marks)**

|  |  |  |
| --- | --- | --- |
| **P** | **S(Pr)** | **S(Pc)** |
| 1 |  | 1 |
| 2 | 1.785501806 | 1.781945662 |
| 4 | 0.220753347 | 0.1338662 |
| 6 | 0.158406209 | 0.134949291 |
| 8 | 0.205348937 | 0.075501966 |
| 10 | 0.182482219 | 0.075501966 |
| 12 | 0.151528927 | 0.092928019 |
| 16 | 0.178120038 | 0.056624214 |
| 20 | 1.415705303 | 1.343731412 |
| 32 | 1.299442791 | 0.966441677 |
| 64 | 0.695286987 | 0.658100016 |

**5) Estimate Parallelization fraction and Inference (5 Marks)**