# Question 1:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Threads** | **cores** | **Sequential** | **Executing time** | **Speedup** | **Processor Utilization** |
| 1 | 4 | 1936 | 1936 | 1 | 0.25 |
| 2 | 4 | 1936 | 1161 | 1.667528 | 0.416882 |
| 3 | 4 | 1936 | 965 | 2.0062176 | 0.5015544 |
| 4 | 4 | 1936 | 901 | 2.1487236 | 0.53718091 |
| 5 | 4 | 1936 | 859 | 2.2537835 | 0.56344587 |
| 6 | 4 | 1936 | 848 | 2.2830189 | 0.57075472 |
| 7 | 4 | 1936 | 827 | 2.3409915 | 0.58524788 |
| 8 | 4 | 1936 | 801 | 2.4169788 | 0.60424469 |
| 9 | 4 | 1936 | 787 | 2.4599746 | 0.61499365 |
| 10 | 4 | 1936 | 782 | 2.4757033 | 0.61892583 |

Sometimes the additional threads don’t reduce the executing time, because with more thread the cpu has to take more time to switch between threads.

Code:

int main()

{

const int n = 100000000;

double\* a;

a = (double \*)malloc(n \* sizeof(double));

clock\_t t;

int number;

for (int i = 0; i < n; i++) {

a[i] = i + 1;

}

cout << "Please input the thread number...";

cin >> number;

t = clock();

omp\_set\_num\_threads(number);

#pragma omp parallel for

for (int i = 0; i < n; i++) {

double d = log10(a[i]);

}

printf("Elapsed Time : %d\n", clock() - t);

free(a);

return 0;

}

# Question 2:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Threads** | **cores** | **Sequential** | **Executing time** | **Speedup** | **Processor Utilization** |
| 1 | 4 | 22531 | 22531 | 1 | 0.25 |
| 2 | 4 | 22531 | 17907 | 1.258223 | 0.31455576 |
| 3 | 4 | 22531 | 14634 | 1.539634 | 0.384908432 |
| 4 | 4 | 22531 | 13812 | 1.631263 | 0.407815668 |
| 5 | 4 | 22531 | 11896 | 1.893998 | 0.473499496 |
| 6 | 4 | 22531 | 11566 | 1.948037 | 0.487009338 |
| 7 | 4 | 22531 | 11086 | 2.032383 | 0.508095797 |
| 8 | 4 | 22531 | 10634 | 2.11877 | 0.529692496 |
| 9 | 4 | 22531 | 10357 | 2.175437 | 0.543859226 |
| 10 | 4 | 22531 | 10914 | 2.064413 | 0.51610317 |

First solution use some shared variables and changed these variables in each thread. So it does not work with paralleling in the first solution. The second solution worked.

void f2(void) {

const int n = 100000;

//const int n = 4000; //For this small data, it might work for paralization.

int \*a, \*b, \*c;

a = (int \*)malloc(n \* sizeof(int));

b = (int \*)malloc(n \* sizeof(int));

c = (int \*)malloc(2 \* n \* sizeof(int));

for (int i = 0; i < n; i++) {

a[i] = 2 \* i;

b[i] = 2 \* i + 1;

}

int number;

cout << "Please input the thread number...";

cin >> number;

int x = 0, y = 0;

clock\_t t;

t = clock();

omp\_set\_num\_threads(number);

#pragma omp parallel for

for (int i = 0; i < n; i++) {

int j = 0;

while (a[i] >= b[j] && j < n) {

j++;

}

c[i + j] = a[i];

}

omp\_set\_num\_threads(number);

#pragma omp parallel for

for (int i = 0; i < n; i++) {

int j = 0;

while (b[i] >= a[j] && j < n) {

j++;

}

c[i + j] = b[i];

}

/\*for (int i = 0; i < 2 \* n; i++) {

printf("Number %d is %d\n", i, c[i]);

}\*/

printf("Time used: %d", clock() - t);

free(a);

free(b);

free(c);

}