Practical 1: Speeding up performance with openmp

GAHAN SARAIYA (18MCEC10)

18mcec10@nirmauni.ac.in

I. AIM

- Speeding up performance with openmp

II. INTRODUCTION

Generic steps:

- Making code sequential code to parallel using openmp
- Do speedup calculation – Sequential/parallel
- compute the same speedup for various values

III. IMPLEMENTATION

Implementing Vector Addition Steps:

- Implement Sequential code
- Implement Parallel code
- Implement function for verification of output (for accuracy calculation)
- Observe time for consumed for execution of sequential and parallel code over 100 iterations (repeating operation for 100 times)
- Calculate speedup for various input size

I. Sequential Code

```
// -----------------------------------------------------------------------------------------
// Author: Gahan Saraiya
// GiT: http://github.com/gahan9/
// StackOverflow: https://stackoverflow.com/users/story/7664524
// Website: http://gahan9.github.io/
// -----------------------------------------------------------------------------------------
// Making code sequential code to parallel using openmp
// Do speedup calculation -- Sequential/parallel
// Show profiling and total execution time
// Standard vector addition (serial, without OMP)
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <omp.h>

#define ARRAY_SIZE 1000000000
#define REPEAT     100


void vector_add(double* vector1, double* vector2, double* result) {
        for (int i=0; i < ARRAY_SIZE; i++){
                result[i] = vector1[i] + vector2[i];
    }
}


double* generate_array(int no_of_elements) {
    // no_of_elements : Generate array of `no_of_elements` number of elements
        double* array = (double*) malloc(no_of_elements*sizeof(double));
        for(int i=0; i<no_of_elements; i++)
                array[i] = rand()%10000;
        return array;
}

int verify(double* vector1, double* vector2) {
    // Verifying accuracy of computation
        double *adder = (double*) malloc(ARRAY_SIZE*sizeof(double));
        double *verifier = (double*) malloc(ARRAY_SIZE*sizeof(double));
        vector_add(vector1, vector2, adder);

        for(int i=0; i<ARRAY_SIZE; i++) {
                verifier[i] = vector1[i] + vector2[i];
    }
        for(int i=0; i<ARRAY_SIZE; i++){
                if(verifier[i] != adder[i]){
                        return 0;
        }
    }
        return 1;
}


int main() {
        // Generate input vectors and destination vector
        double *vector1 = generate_array(ARRAY_SIZE);
        double *vector2 = generate_array(ARRAY_SIZE);
        double *result_vector = (double*) malloc(ARRAY_SIZE*sizeof(double));
```

```
        // Double check vector_add is correct
        if(!verify(vector1, vector2)) {
                printf("vector_add does not match oracle\n");
                return 0;
        }

        // Test framework that sweeps the number of threads and times each
    // runs for iteration REPEAT
        double start_time, run_time;
        int num_threads = 1;
        for(int i=1; i<=num_threads; i++) {
                omp_set_num_threads(i);
                start_time = omp_get_wtime();
                for(int j=0; j<REPEAT; j++){
                        vector_add(vector1, vector2, result_vector);
        }
                run_time = omp_get_wtime() - start_time;
          printf(" %d thread(s) took %f seconds\n",i,run_time);
        }
}
```

II. Parallel Code

```
// -----------------------------------------------------------------------------------------------
// Author: Gahan Saraiya
// GiT: http://github.com/gahan9/
// StackOverflow: https://stackoverflow.com/users/story/7664524
// Website: http://gahan9.github.io/
// -----------------------------------------------------------------------------------------------
// Making code sequential code to parallel using openmp
// Do speedup calculation -- Sequential/parallel
// Show profiling and total execution time
// Standard vector addition (serial, without OMP)

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <omp.h>

#define ARRAY_SIZE 100000000
#define REPEAT     100
#define NUM_THREADS 4

void vector_add(double* vector1, double* vector2, double* result) {
    int i;
        int size_chunks = ARRAY_SIZE/NUM_THREADS;
    #pragma omp parallel
        {
```

```
        // #pragma omp parallel for num_threads(NUM_THREADS)
        // #pragma omp parallel for
        // for (i=0; i < ARRAY_SIZE; i++){
                for(int i=size_chunks*omp_get_thread_num(); i < size_chunks*(omp_get_thread_num()+1)
            result[i] = vector1[i] + vector2[i];
        }
    }
}


double* generate_array(int no_of_elements) {
    // no_of_elements : Generate array of `no_of_elements` number of elements
        double* array = (double*) malloc(no_of_elements*sizeof(double));
        for(int i=0; i<no_of_elements; i++)
                array[i] = rand()%10000;
        return array;
}

int verify(double* vector1, double* vector2) {
    // Verifying accuracy of computation
        double *adder = (double*) malloc(ARRAY_SIZE*sizeof(double));
        double *verifier = (double*) malloc(ARRAY_SIZE*sizeof(double));
        vector_add(vector1, vector2, adder);

        for(int i=0; i<ARRAY_SIZE; i++) {
                verifier[i] = vector1[i] + vector2[i];
    }
        for(int i=0; i<ARRAY_SIZE; i++){
                if(verifier[i] != adder[i]){
                        return 0;
        }
    }
        return 1;
}


int main() {
        // Generate input vectors and destination vector
        double *vector1 = generate_array(ARRAY_SIZE);
        double *vector2 = generate_array(ARRAY_SIZE);
        double *result_vector = (double*) malloc(ARRAY_SIZE*sizeof(double));

        // Double check vector_add is correct
        if(!verify(vector1, vector2)) {
                printf("vector_add does not match actual result\n");
                return 0;
        }
```

```
    // Test framework that sweeps the number of threads and times each
// runs for iteration REPEAT
    double start_time, run_time;
start_time = omp_get_wtime();

for(int j=0; j<REPEAT; j++){
    vector_add(vector1, vector2, result_vector);
}
run_time = omp_get_wtime() - start_time;
      printf(" %d thread(s) took %f seconds\n",NUM_THREADS,run_time);

}
```

IV. Analysis

Below is the result of observation over 100 iterations for various input size

Data size	Sequential Time	Parallel Time$_1$	Parallel Time$_2$	Speedup$_1$	Speedup$_2$
10^5	0.003093	0.002066	0.00227	1.497095837	1.362555066
10^6	0.031473	0.029124	0.022613	1.08065513	1.391810021
10^7	0.366141	0.305616	0.313129	1.198042642	1.169297638
10^8	3.490682	3.014184	2.952571	1.15808524	1.182251672
10^9	35.395422	29.491622	29.188446	1.200185666	1.212651814

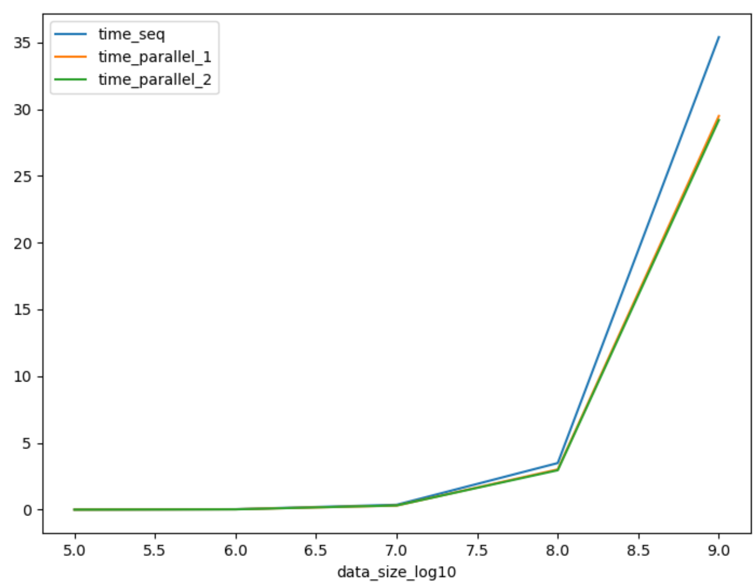

Figure 1: Analysis of Time consumed by thread(s) for calculation of number of inputs (x-axis scaled to the log base 10)

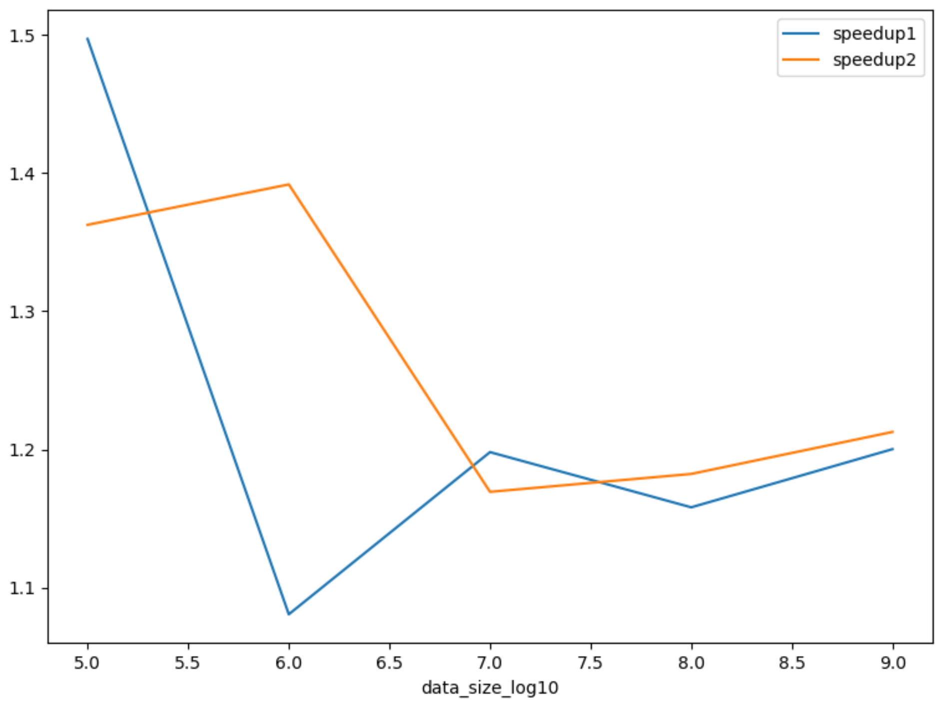

Figure 2: Comparison of speed up achieved from two method of parallelism