CSE4001 - Parallel and Distributed Computing

Lab 21+22

Assessment-1

Submitted by: Alokam Nikhitha

Reg No:19BCE2555

Submitted to: HITESHWAR KUMAR AZAD

Question

1. Write an OpenMP code to find the Sum of Elements of a One-Dimensional Real Array using vector addition. Where the two vectors A and B are added into C by spawning a team of threads and assigning a chunk of work to each thread.

Note: Sets the environment variable omp_num_threads to 6.

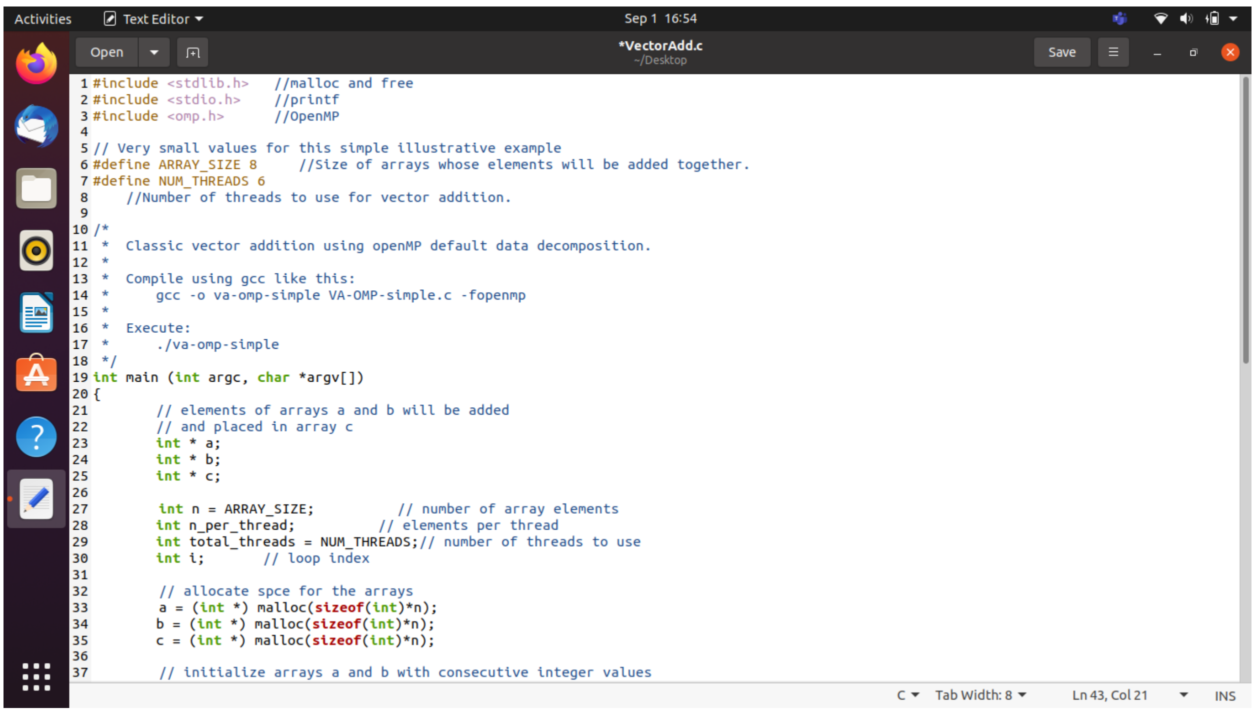
Code:

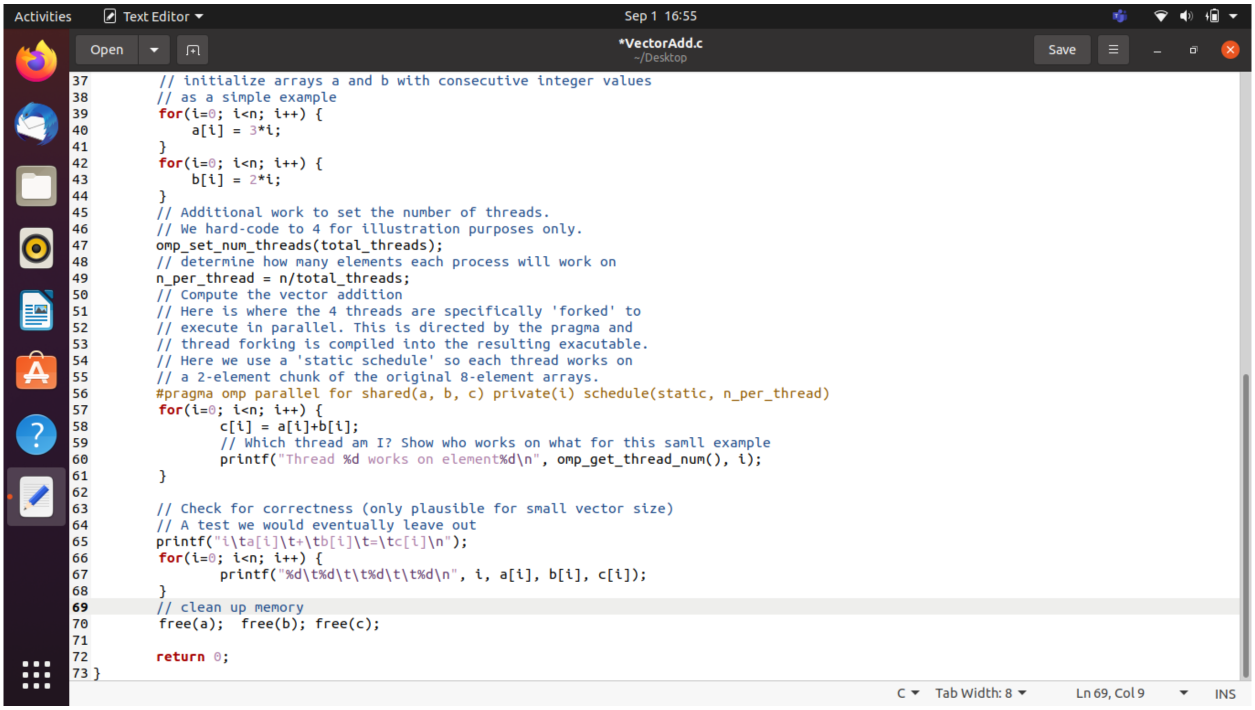
```
#include <stdlib.h> //malloc and free
#include <stdio.h> //printf
#include <omp.h> //OpenMP

// Very small values for this simple illustrative example
#define ARRAY_SIZE 8
//Size of arrays whose elements will be added together.
#define NUM_THREADS 10 //Number of threads to use for vector addition.

/*
* Classic vector addition using openMP default data decomposition.
*
* Compile using gcc like this:
* gcc -o VectorAdd -fopenmp VectorAdd.c
* or, g++ -fopenmp VectorAdd.c
*
* Execute:
* ./VectorAdd
* or, ./a.out
*/
```

```
int main (int argc, char *argv[])
{
// To pass command line arguments, we typically define main() with two arguments : first argument is the number of command line arguments and second is list of command-line arguments.
// int main(int argc, char *argv[]) { /* ... */ }
// argc (ARGument Count) is int and stores number of command-line arguments passed by the user including the name of the program. So if we pass a value to a program, value of argc would be 2 (one for argument and one for program name)
//The value of argc should be non negative.
//argv(ARGument Vector) is array of character pointers listing all the arguments.
//If argc is greater than zero,the array elements from argv[0] to argv[argc-1] will contain pointers to strings.
//Argv[0] is the name of the program , After that till argv[argc-1] every element is command -line arguments.



// elements of arrays a and b will be added
// and placed in array c
int * a;
int * b;
int * c;

int n = ARRAY_SIZE; // number of array elements
int n_per_thread; // elements per thread
int total_threads = NUM_THREADS; // number of threads to use
int i; // loop index

// allocate spce for the arrays
a = (int *) malloc(sizeof(int)*n);
b = (int *) malloc(sizeof(int)*n);
c = (int *) malloc(sizeof(int)*n);
```

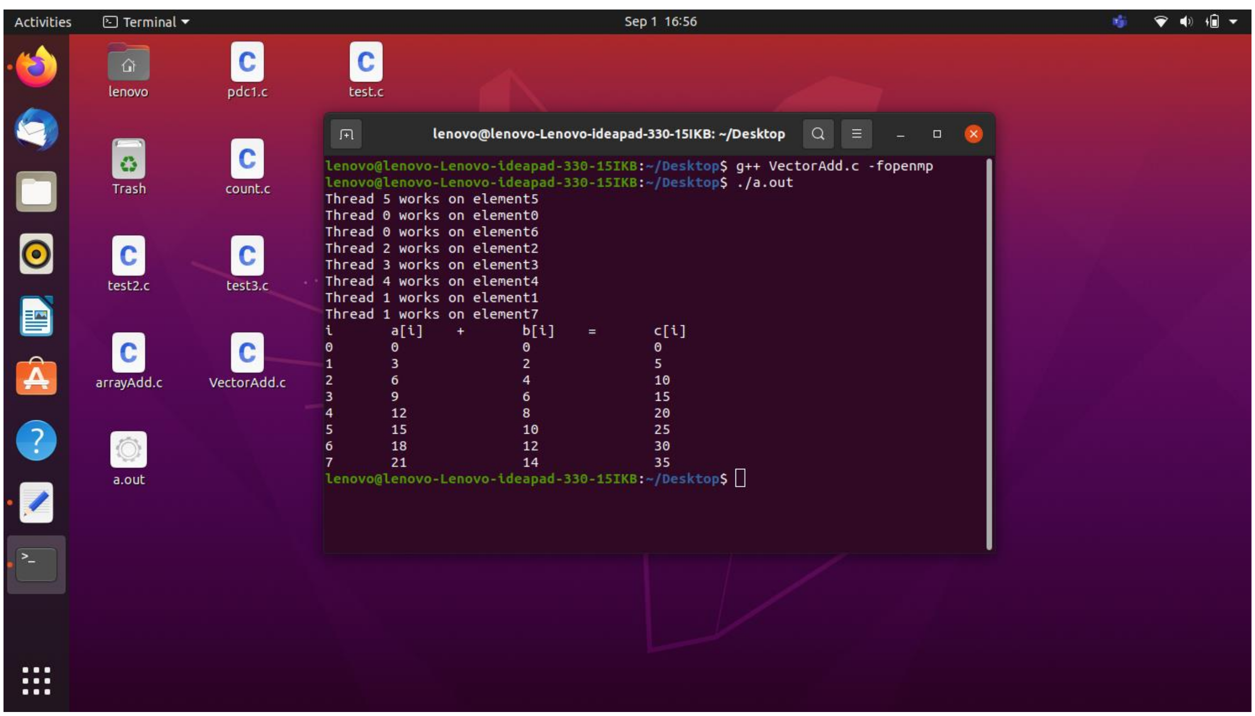
```
// initialize arrays a and b with consecutive integer values
// as a simple example
for(i=0; i<n; i++) {
a[i] = 3*i;
}
for(i=0; i<n; i++) {
b[i] = 2*i;
}

// Additional work to set the number of threads.
// We hard-code to 4 for illustration purposes only.
omp_set_num_threads(total_threads);

// determine how many elements each process will work on
n_per_thread = n/total_threads;

// Compute the vector addition
// Here is where the 4 threads are specifically 'forked' to
// execute in parallel. This is directed by the pragma and
// thread forking is compiled into the resulting exacutable.
// Here we use a 'static schedule' so each thread works on
// a 2-element chunk of the original 8-element arrays.
#pragma omp parallel for shared(a, b, c) private(i) schedule(static, n_per_thread)
for(i=0; i<n; i++) {
c[i] = a[i]+b[i];
// Which thread am I? Show who works on what for this samll example
printf("Thread %d works on element%d\n", omp_get_thread_num(), i);
}

// Check for correctness (only plausible for small vector size)
// A test we would eventually leave out
printf("i\ta[i]\t+\tb[i]\t=\tc[i]\n");
```

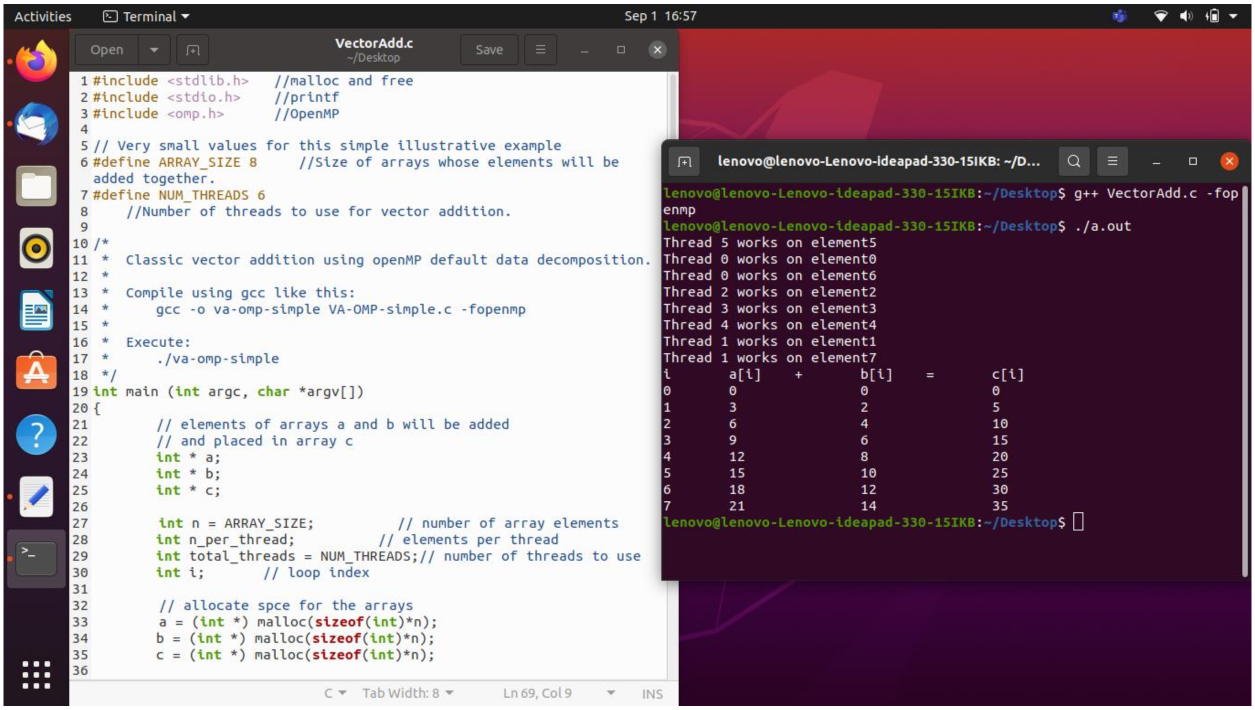
```
for(i=0; i<n; i++) {
printf("%d\t%d\t\t%d\t\t%d\n", i, a[i], b[i], c[i]);
}

// clean up memory
free(a); free(b); free(c);

return 0;
}
```




Activities
Text Editor
Sep 1 16:55
Open
*VectorAdd.c
~/Desktop
Save
// initialize arrays a and b with consecutive integer values
// as a simple example
for(i=0; i<n; i++) {
a[i] = 3*i;
}
for(i=0; i<n; i++) {
b[i] = 2*i;
}
// Additional work to set the number of threads.
// We hard-code to 4 for illustration purposes only.
omp_set_num_threads(total_threads);
// determine how many elements each process will work on
n_per_thread = n/total_threads;
// Compute the vector addition
// Here is where the 4 threads are specifically 'forked' to
// execute in parallel. This is directed by the pragma and
// thread forking is compiled into the resulting exacutable.
// Here we use a 'static schedule' so each thread works on
// a 2-element chunk of the original 8-element arrays.
#pragma omp parallel for shared(a, b, c) private(i) schedule(static, n_per_thread)
for(i=0; i<n; i++) {
c[i] = a[i]+b[i];
// Which thread am I? Show who works on what for this samll example
printf("Thread %d works on element%d\n", omp_get_thread_num(), i);
}
// Check for correctness (only plausible for small vector size)
// A test we would eventually leave out
printf("i\ta[i]\t+\tb[i]\t=\tc[i]\n");
for(i=0; i<n; i++) {
printf("%d\t%d\t\t%d\t\t%d\n", i, a[i], b[i], c[i]);
}
// clean up memory
free(a); free(b); free(c);
return 0;
}
C
Tab Width: 8
Ln 69, Col 9
INS

OUTPUT:


Activities
Terminal
Sep 1 16:56
lenovo
pdc1.c
test.c
Trash
count.c
test2.c
test3.c
arrayAdd.c
VectorAdd.c
a.out
lenovo@lenovo-Lenovo-ideapad-330-15IKB: ~/Desktop
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~/Desktop$ g++ VectorAdd.c -fopenmp
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~/Desktop$ ./a.out
Thread 5 works on element5
Thread 0 works on element0
Thread 0 works on element6
Thread 2 works on element2
Thread 3 works on element3
Thread 4 works on element4
Thread 1 works on element1
Thread 1 works on element7
i a[i] + b[i] = c[i]
0 0 0 0
1 3 2 5
2 6 4 10
3 9 6 15
4 12 8 20
5 15 10 25
6 18 12 30
7 21 14 35
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~/Desktop$


Activities
Terminal
Sep 1 16:57
Open
VectorAdd.c
~/Desktop
Save
#include <stdlib.h>   //malloc and free
#include <stdio.h>   //printf
#include <omp.h>   //OpenMP
// Very small values for this simple illustrative example
#define ARRAY_SIZE 8   //Size of arrays whose elements will be added together.
#define NUM_THREADS 6
//Number of threads to use for vector addition.
/*
 * Classic vector addition using openMP default data decomposition.
 *
 * Compile using gcc like this:
 *   gcc -o va-omp-simple VA-OMP-simple.c -fopenmp
 *
 * Execute:
 *   ./va-omp-simple
 */
int main (int argc, char *argv[])
{
// elements of arrays a and b will be added
// and placed in array c
int * a;
int * b;
int * c;
int n = ARRAY_SIZE;   // number of array elements
int n_per_thread;   // elements per thread
int total_threads = NUM_THREADS;// number of threads to use
int i;   // loop index
// allocate spce for the arrays
a = (int *) malloc(sizeof(int)*n);
b = (int *) malloc(sizeof(int)*n);
c = (int *) malloc(sizeof(int)*n);
C
Tab Width: 8
Ln 69, Col 9
INS
lenovo@lenovo-Lenovo-ideapad-330-15IKB: ~/D...
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~/Desktop$ g++ VectorAdd.c -fopenmp
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~/Desktop$ ./a.out
Thread 5 works on element5
Thread 0 works on element0
Thread 0 works on element6
Thread 2 works on element2
Thread 3 works on element3
Thread 4 works on element4
Thread 1 works on element1
Thread 1 works on element7
i a[i] + b[i] = c[i]
0 0 0 0
1 3 2 5
2 6 4 10
3 9 6 15
4 12 8 20
5 15 10 25
6 18 12 30
7 21 14 35
lenovo@lenovo-Lenovo-ideapad-330-15IKB:~/Desktop$