# CSE4001 - Parallel and Distributed Computing

# Lab 21+22

# Lab Task1

# Submitted by: Alokam Nikhitha

# Reg No:19BCE2555

# VECTOR ADDITION

# 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 4 //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);

// 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]);

}

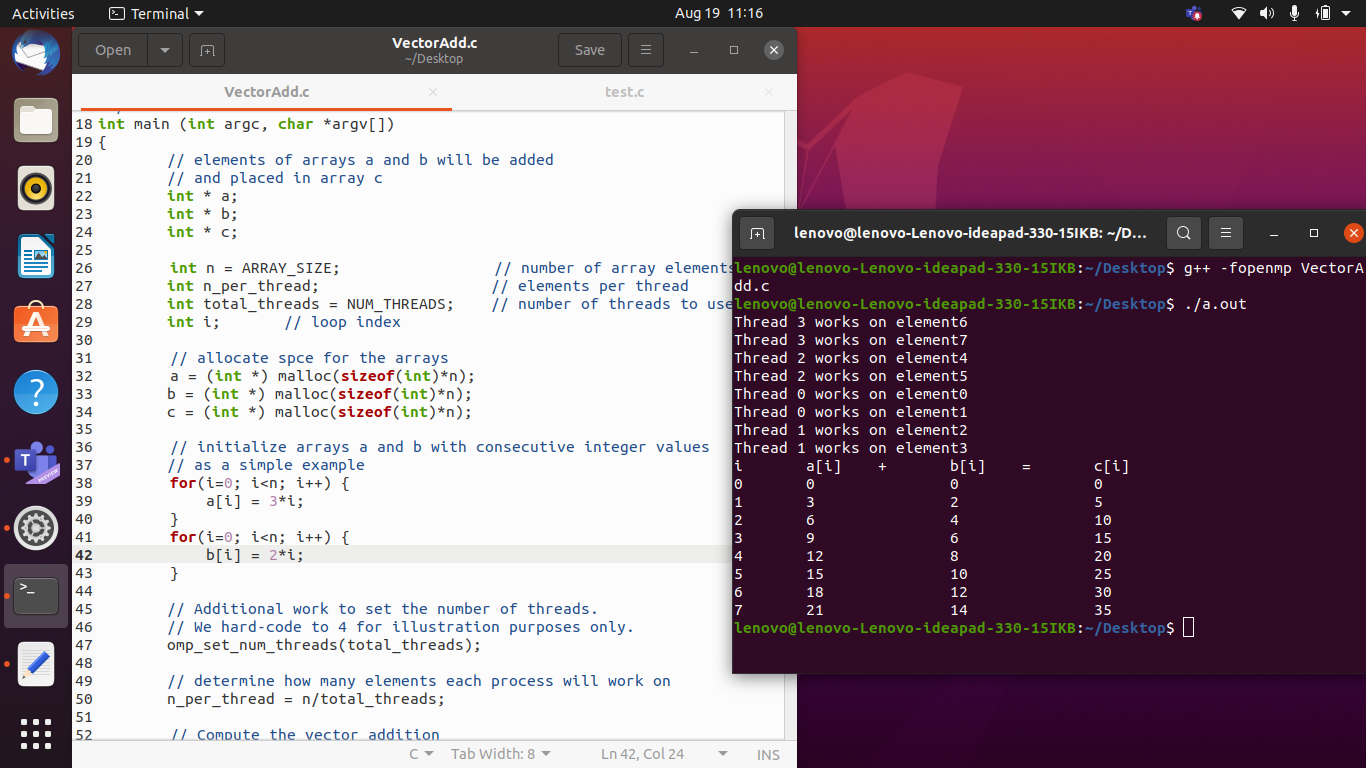
// clean up memory

free(a); free(b); free(c);

return 0;

}

# OUTPUT:



# TEST

# CODE:

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

int main(int argc,char \*argv[]) {

double sum,a[256],b[256];

int i,n=256;

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

a[i]=i\*1.5;

b[i]=i\*3.0;

}

sum=0;

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

sum+=a[i] \*b[i];

printf("a\*b = %f\n",sum);

}

# OUTPUT:

# 

# SUMMATION CODE:

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

int main(int args, char \*argv[]) {

int partial\_Sum;

int total\_Sum;

int i;

#pragma omp parallel private(partial\_Sum) shared(total\_Sum)

{

partial\_Sum=0;

total\_Sum=0;

#pragma omp

{

for(i=1;i<=500;i++)

{

partial\_Sum += i;

}

}

#pragma omp critical

{

total\_Sum += partial\_Sum;

}

}

printf("Total Sum: %d\n", total\_Sum);

return 0;

}

# OUTPUT:

