Lab 7. OpenACC – generalities, simple example and matrix operations

Main goals of the assignment

- Understand the differences between CPU and GPU usage.
- Learn about performance through experiments.

A simple example

Consider the example of adding two vectors using OpenACC:

```
#include <stdio.h>
#include <stdlib.h>
#include <openacc.h>

void vecaddgpu( float *restrict r, float *a, float *b, int n ){
#pragma acc kernels loop copyin(a[0:n],b[0:n]) copyout(r[0:n])
    for( int i = 0; i < n; ++i ) r[i] = a[i] + b[i];
}
int main( int argc, char* argv[] ){
        int n; /* vector length */
        float * a; /* input vector 1 */
        float * b; /* input vector 2 */
        float * r; /* output vector */
        float * e; /* expected output values */
        int i, errs;

        if( argc > 1 ) n = atoi( argv[1] );
        else n = 100000; /* default vector length */
        if( n <= 0 ) n = 100000;
        a = (float*)malloc( n*sizeof(float) );
        b = (float*)malloc( n*sizeof(float) );
        r = (float*)malloc( n*sizeof(float) );
        e = (float*)malloc( n*sizeof(float) );
        for( i = 0; i < n; ++i ){
                a[i] = (float)(i+1);
                b[i] = (float)(1000*i);
        }

        /* compute on the GPU */
        vecaddgpu( r, a, b, n );

        /* compute on the host to compare */
        for( i = 0; i < n; ++i ) e[i] = a[i] + b[i];

        /* compare results */
        errs = 0;
        for( i = 0; i < n; ++i )
                if( r[i] != e[i] ) ++errs;
        printf( ''%d errors found\n'', errs );
        return errs;
}
```

1. comment the differences from using OpenMP
2. change kernels to parallel (hints: see the textbook); test the code without copyin, copyout.

Second example

The problem to solve

A OpenACC-based parallel version of matrix multiplication is requested. To investigate its efficiency it should be compared with the sequential version on a laptop and on a GPU cluster.

To do

1. Reuse the code from Lab 2 related to OpenMP and try different versions for the three cycles registering the computation time for large SIZE (e.g. 1000) and checking the correctness of the result:

```
/* Version 1 */
#pragma acc data copyin(a,b) copy(c)
#pragma acc kernels
#pragma acc loop tile(32,32)
  for (i = 0; i < SIZE; ++i) {
    for (j = 0; j < SIZE; ++j) {
#pragma acc loop reduction(+:tmp)
      for (k = 0; k < SIZE; ++k) {

/*Version 2*/
#pragma acc parallel loop gang collapse(2)
  for (i = 0; i < SIZE; ++i)
    for (j = 0; j < SIZE; ++j) {
      for (k = 0; k < SIZE; ++k) {

/*Version 3*/
#pragma acc kernels copyin(a,b) copyout(c)
 #pragma acc loop independent
  for (i = 0; i < SIZE; ++i) {
 #pragma acc loop independent
    for (j = 0; j < SIZE; ++j) {
      for (k = 0; k < SIZE; ++k) {

/*Version 4*/
#pragma acc data copyin(a,b) copyout(c)
 #pragma acc region loop independent vector(32)
  for (i = 0; i < SIZE; ++i) {
 #pragma acc region loop independent vector(32)
    for (j = 0; j < SIZE; ++j) {
      for (k = 0; k < SIZE; ++k) {

/*Version 5*/
#pragma acc parallel copyin(a[0:SIZE][0:SIZE],b[0:SIZE][0:SIZE]) copyout(c[0:SIZE][0:SIZE])
 #pragma acc loop gang
  for (i = 0; i < SIZE; ++i) {
 #pragma acc loop worker
    for (j = 0; j < SIZE; ++j) {
  #pragma acc loop vector reduction(+:c[i][j])
     for (k = 0; k < SIZE; ++k) {

/*Version 6*/
 #pragma acc kernels copy(a[0:SIZE][0:SIZE],b[0:SIZE][0:SIZE],c[0:SIZE][0:SIZE])
    #pragma acc loop independent
    for (j = 0; j < SIZE; j++) {
        #pragma acc loop independent
        for (i = 0; i < SIZE; i++){
            #pragma acc loop seq
            for (k = 0; k < SIZE ; k++)

/*Version 7*/
#pragma acc kernels copy(a[0:SIZE][0:SIZE],b[0:SIZE][0:SIZE],c[0:SIZE][0:SIZE])
    #pragma acc loop independent
    for (j = 0; j < SIZE; j++) {
        #pragma acc loop independent
```

```
        for (i = 0; i < SIZE; i++) {
            #pragma acc loop independent reduction(+:c[i][j])
            for (k = 0; k < SIZE ; k++)

/*Version 8*/
#pragma acc parallel copyin(a[0:SIZE][0:SIZE],b[0:SIZE][0:SIZE]) copyout(c[0:SIZE][0:SIZE])
    #pragma acc loop gang
    for (j = 0; j < SIZE; j++) {
        #pragma acc loop worker
        for (i = 0; i < SIZE; i++) {
           #pragma acc loop vector reduction(+:c[i][j])
           for (k = 0; k < SIZE ; k++)

/*Version 9*/
 #pragma acc copyin(a[0:SIZE][0:SIZE],b[0:SIZE][0:SIZE]) copyout([0:SIZE][0:SIZE])
    #pragma acc parallel loop gang worker
    for (j = 0; j < SIZE; j++) {
        #pragma acc loop vector
        for (i = 0; i < SIZE; i++) {
            #pragma acc loop seq
            for (k = 0; k < SIZE ; k++)
```

2. Study the speedup and compare it with the case of using OpenMP.

3. Visit the site of our HPC center, hpc.uvt.ro. Connect to the BID cluster and run the code on one NVIDIA Tesla V100.

4. Rewrite the code of one variant to use multiple GPUs using OpenMP and OpenACC (hint: split the matrix A and C in a number of bands equal with the number of GPUs and use OpenMP to launch different threads for different GPUs).