**COMPILER**

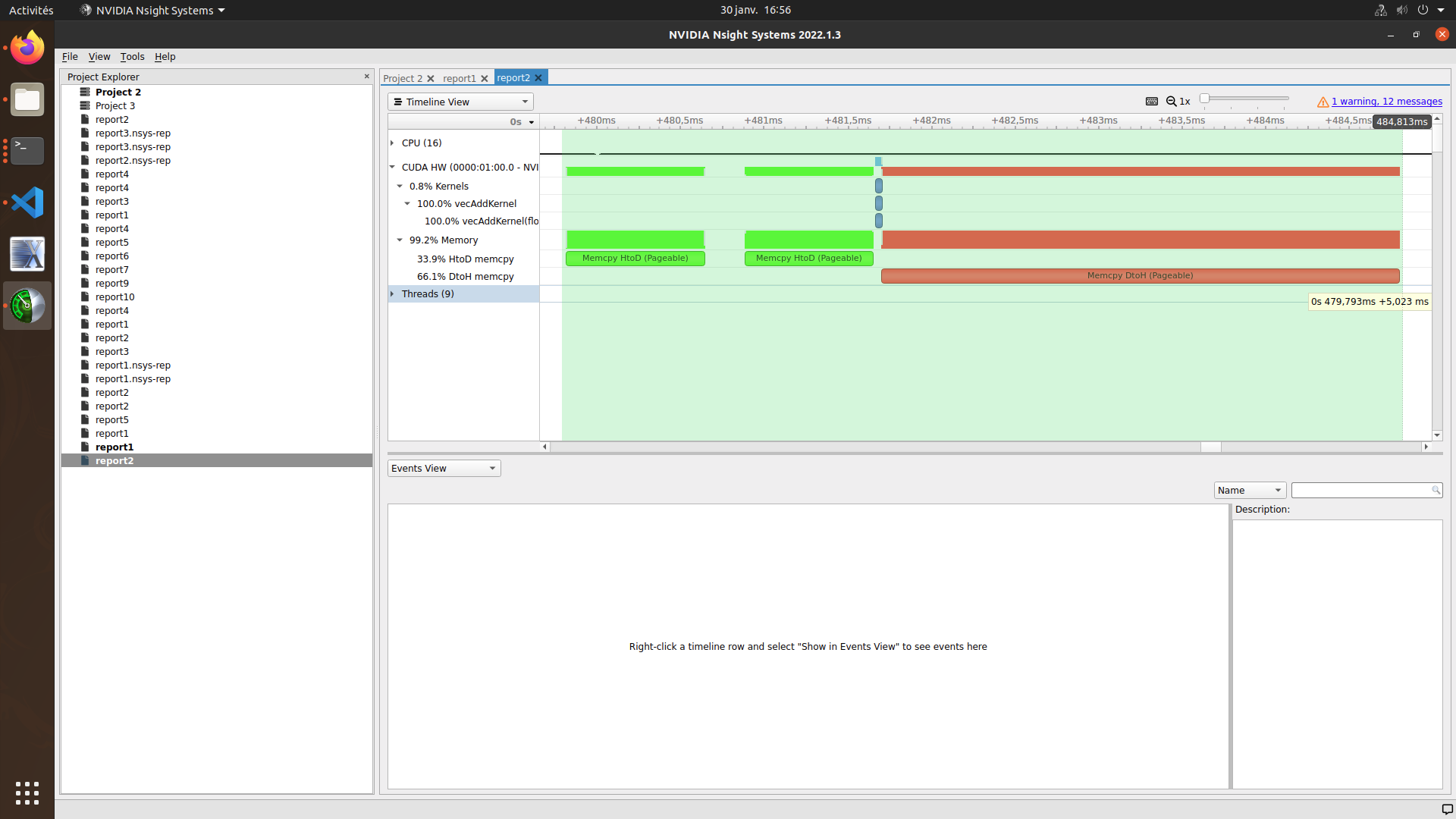
**nvc -acc=gpu -Minfo=accel 0-example-openacc.c**

EXO1

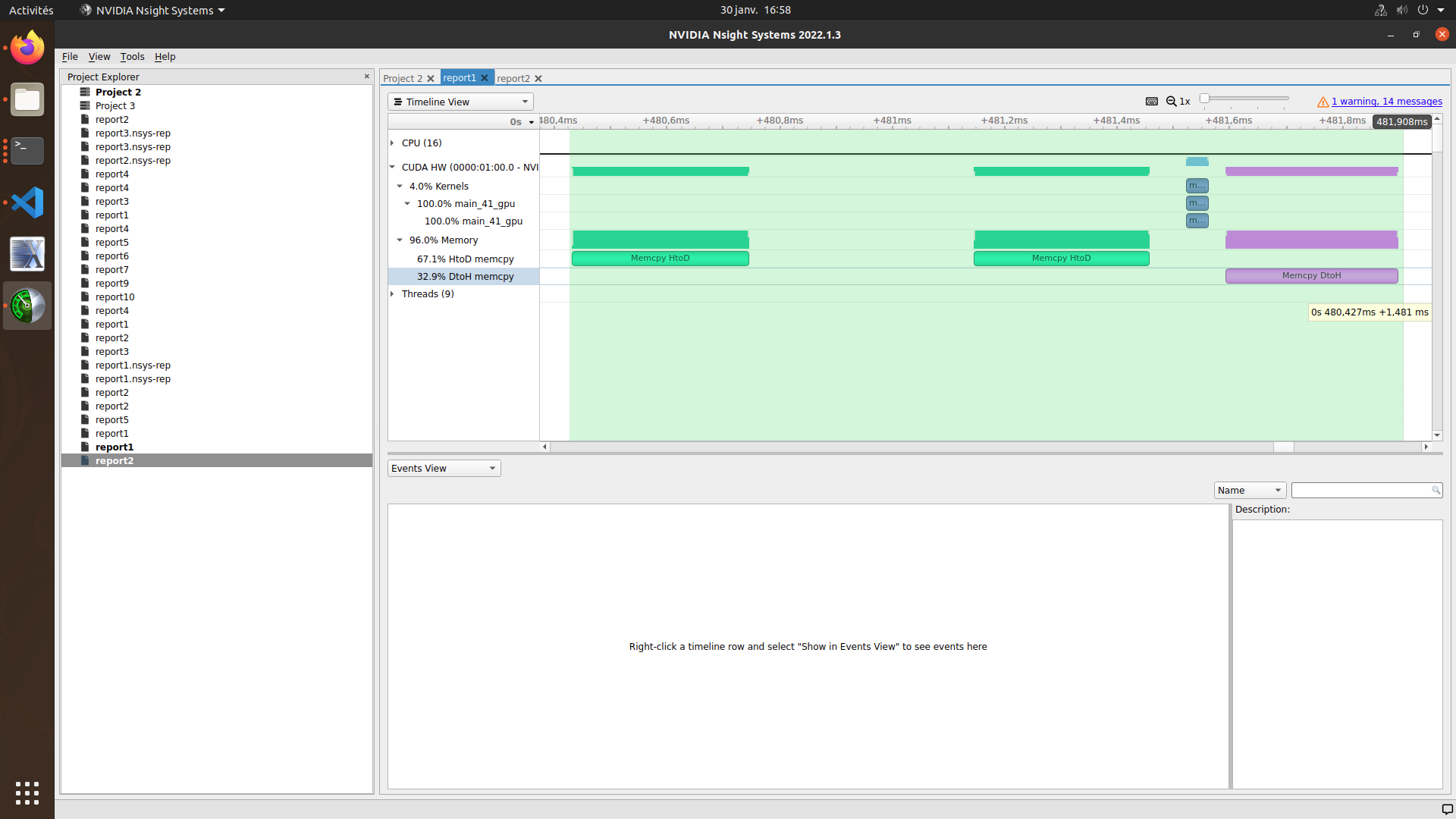
Use the profiler to compare the 2 versions of vector add : CUDA and OpenACC. Are the execution time of the whole algorithm (transfers + computations) are almost the same ? Explain ?

→ transfert et calcul

Profilage ancienne version :



Profilage nouvelle version :



→ Plusieurs copy entrée et sortie

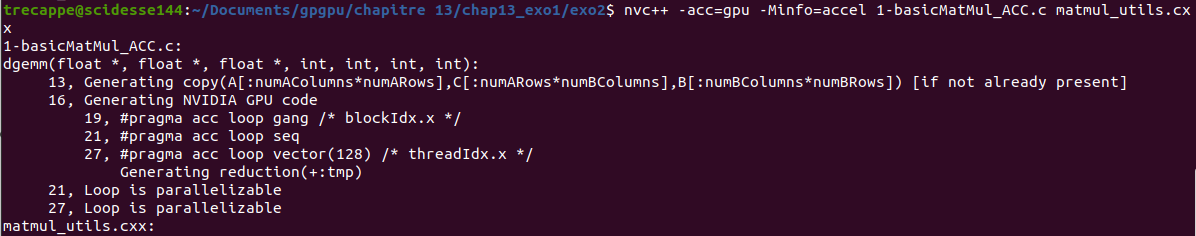
→Ligne OpenACC : enter data et exit data : voit apparaître région de données

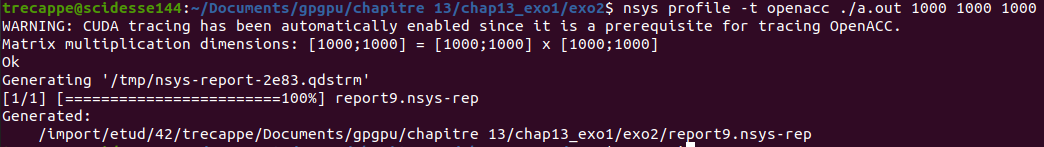
→Dure longtemps : quand entre dans région : copy in est faite (allocation) puis memcopy (fait automatiquement)

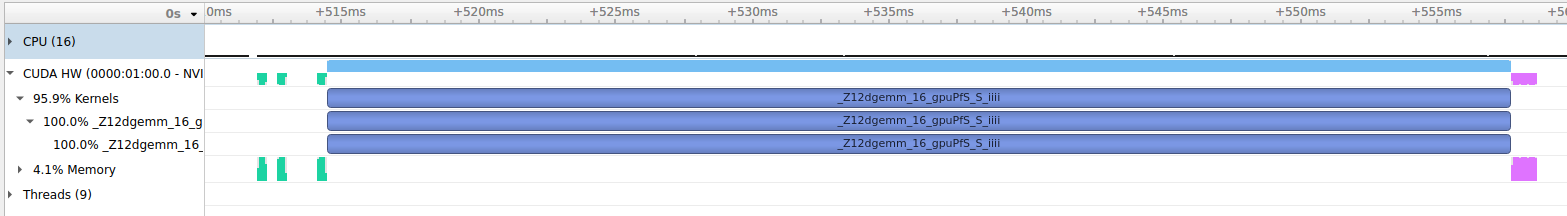
nvc++ -acc=gpu -Minfo=accel 1-basicMatMul\_ACC.c matmul\_utils.cxx

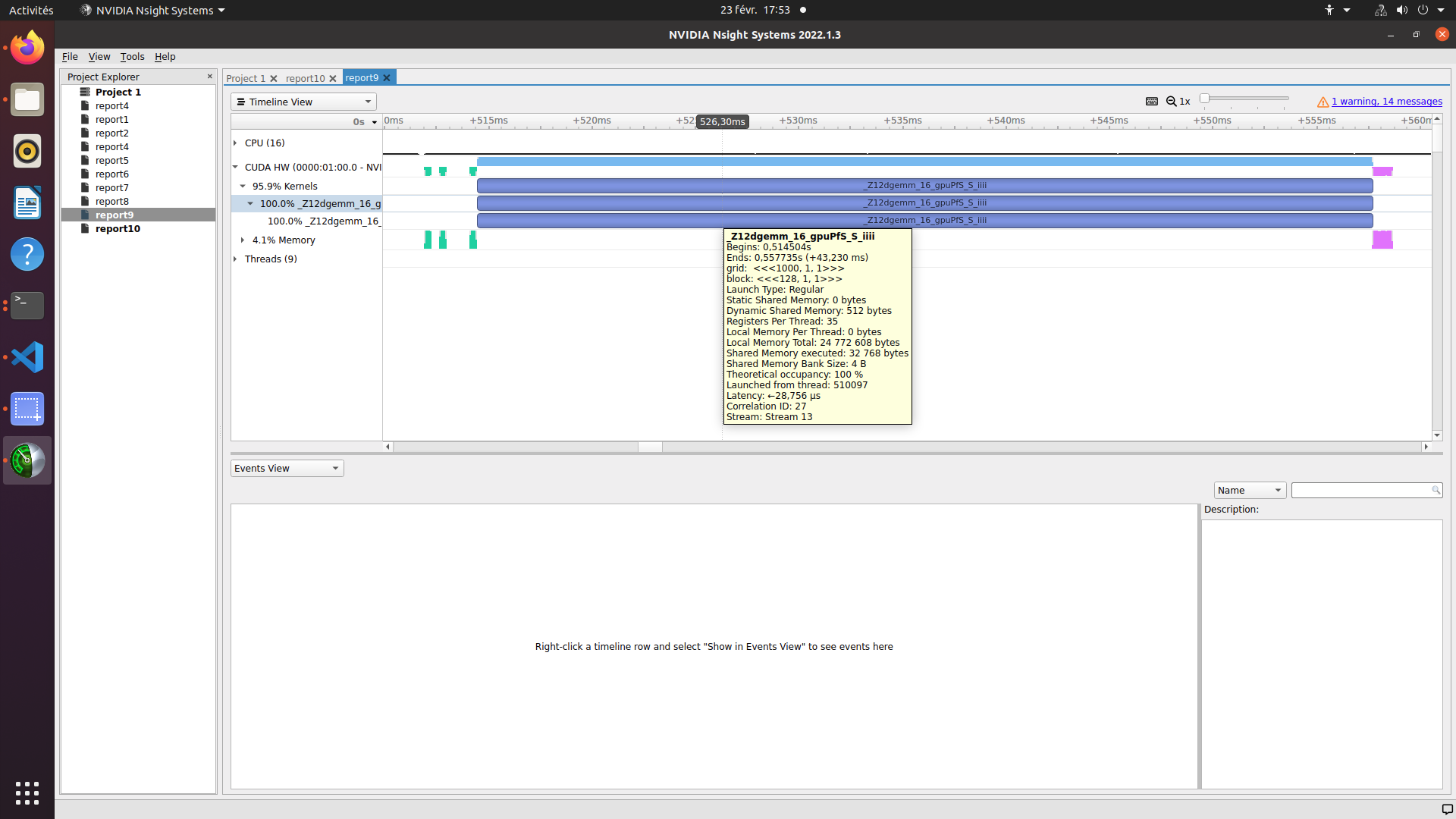
EXO2

acc

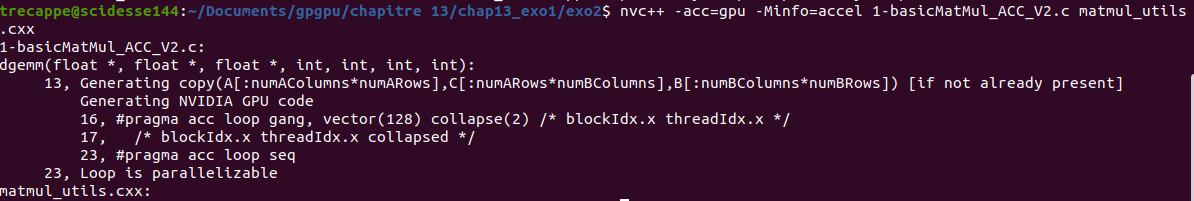


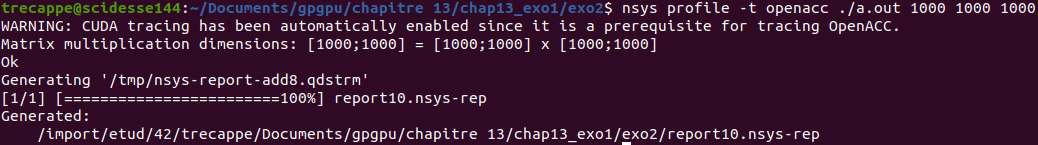


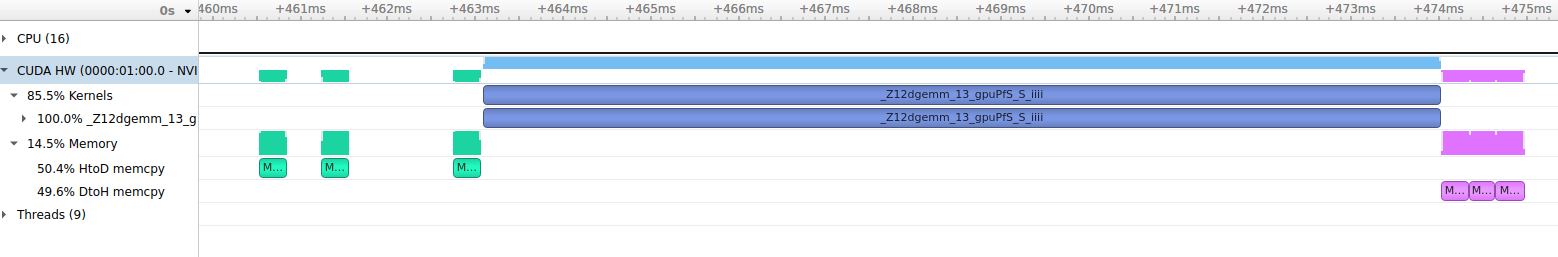


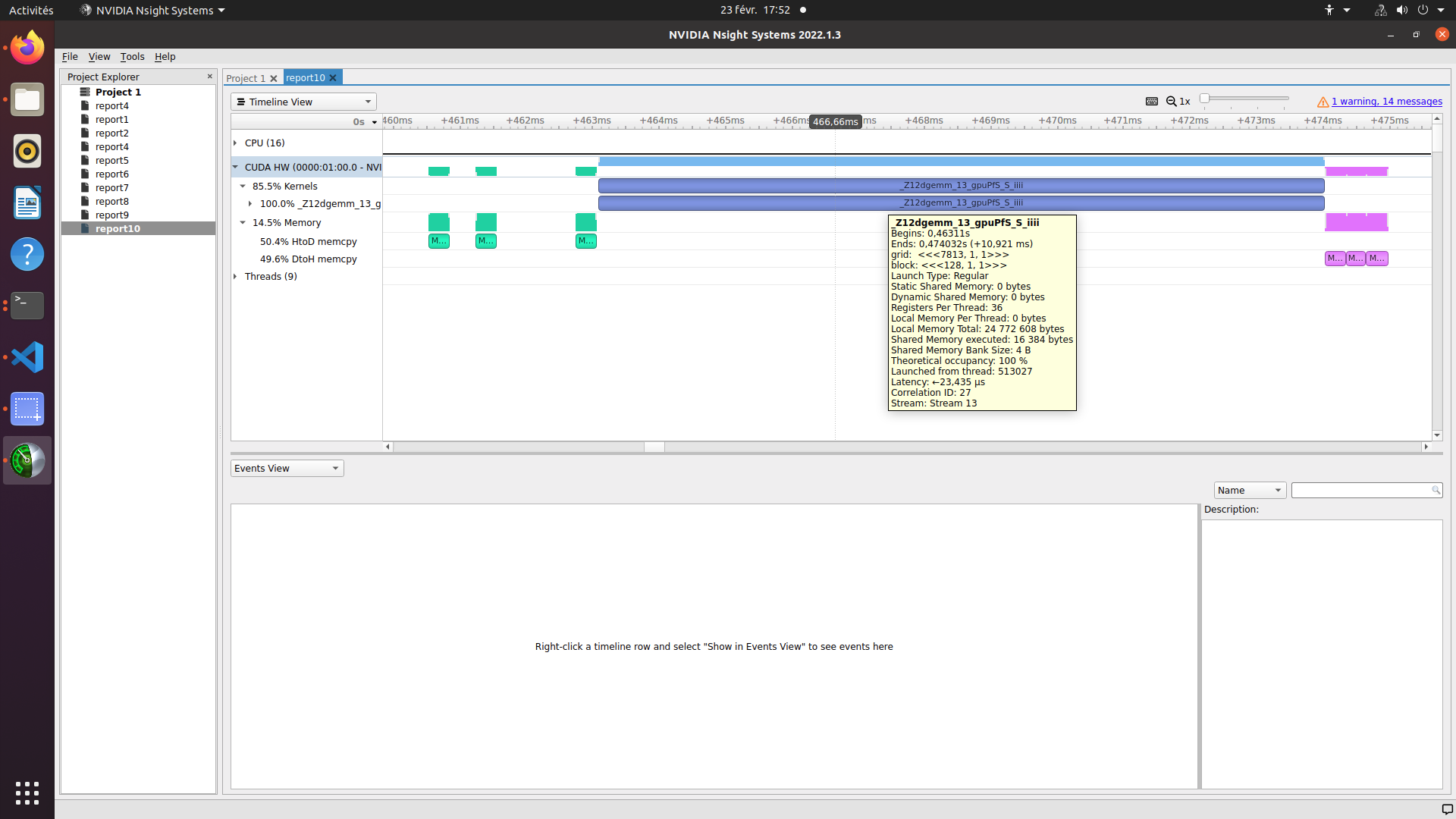


acc V2

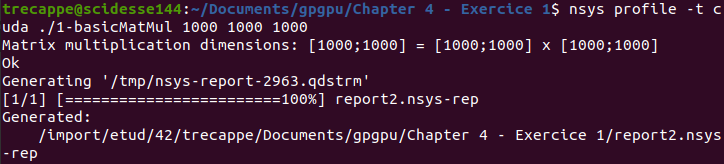




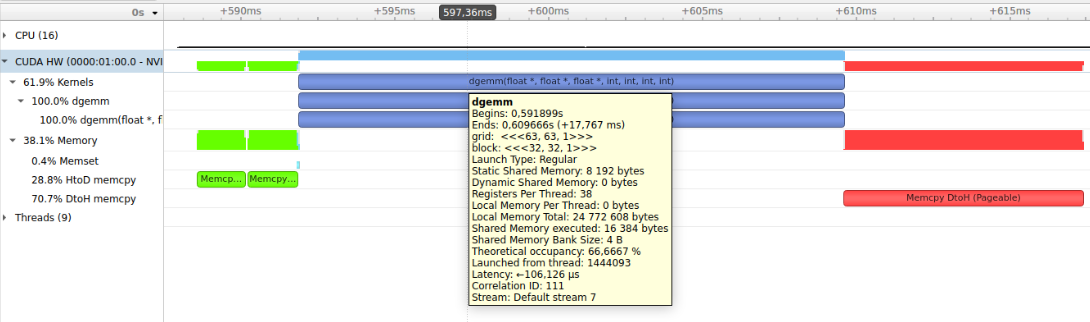




version 1 cuda



version 2 cuda avec tuille



#include <stdio.h>

#include <cuda.h>

#include <time.h>

#include "matmul\_utils.hpp"

// Cuda kernel

void dgemm(float \*restrict A, float \*restrict B, float \*restrict C,

int numARows, int numAColumns, int numBRows, int numBColumns) {

#pragma acc data copy(A[0:numARows\*numAColumns], B[0:numBRows\*numBColumns], C[0:numARows\*numBColumns])

{

#pragma acc parallel

{

#pragma acc loop

for (int x = 0; x<numBColumns; x++){

#pragma acc loop

for (int y =0; y<numARows; y++){

int C\_Offset = y\*numBColumns+x;

float tmp = 0;

#pragma acc loop reduction(+:tmp)

for(int k = 0; k < numAColumns; k++) {

int indice\_A = y\*numAColumns+k;

int indice\_B = k\*numBColumns+x;

tmp += A[indice\_A] \* B[indice\_B];

}

C[C\_Offset] = tmp;

}

}

}

}

}

int main(int argc, char \*\*argv)

{

if(argc!=4) {printf("Usage : %s [nb of rows for A] [nb of cols for A] [nb of cols for B]\n", argv[0]);exit(2);}

//initilize a pseudo-random number generator

srand(time(0));

int numARows, numAColumns,numBRows, numBColumns,numCRows, numCColumns;

// Read given dimensions

numARows = atoi(argv[1]); //C = A\*B

numAColumns = atoi(argv[2]);

numBColumns = atoi(argv[3]);

// Compute the remaining dimensions for given ones

numBRows = numAColumns; //@TODO@ ok

numCRows = numARows; //@TODO@ ok

numCColumns = numBColumns; //@TODO@ ok

printf("Matrix multiplication dimensions: [%d;%d] = [%d;%d] x [%d;%d]\n",

numCRows, numCColumns, numARows, numAColumns, numBRows, numBColumns);

// host pointers

float \*host\_a, \*host\_b, \*host\_c;

// Allocations on host

host\_a = (float \*)calloc(numARows\*numAColumns, sizeof(float));

host\_b = (float \*)calloc(numBRows\*numBColumns, sizeof(float));

host\_c = (float \*)calloc(numCRows\*numCColumns, sizeof(float));

// Initialize vectors

init(host\_a,host\_b,numARows, numAColumns, numBRows, numBColumns);

// Call the kernel

// @TODO@ : complete to call the kernel

dgemm(host\_a, host\_b, host\_c, numARows, numAColumns, numBRows, numBColumns);

// Check result

check(host\_a,host\_b,host\_c,numARows, numAColumns, numBRows, numBColumns);

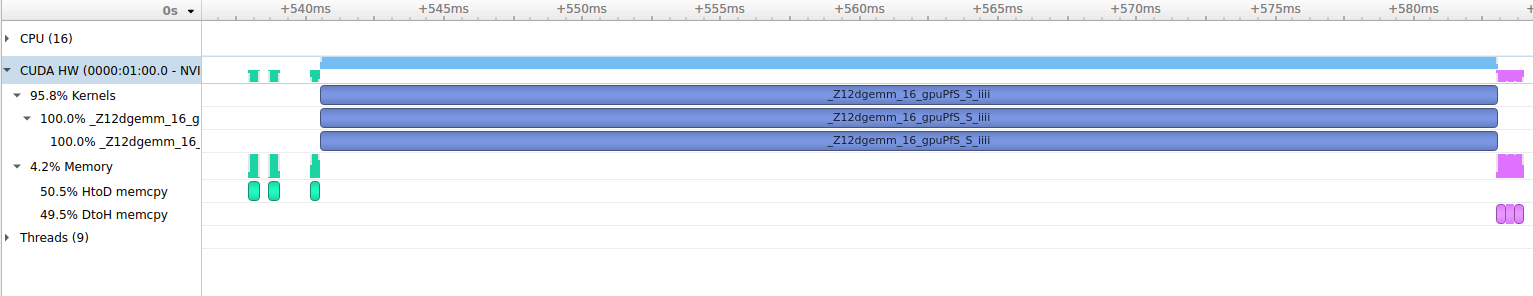
// Free device memory

// @TODO@ : complete to deallocate memory

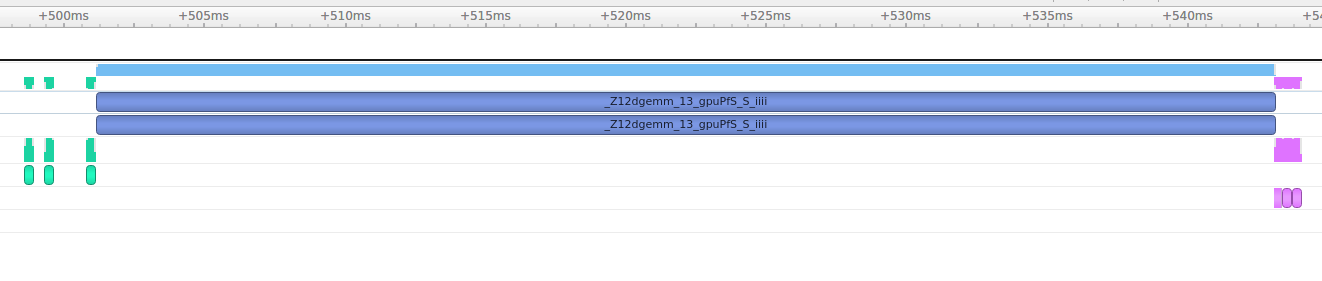
free(host\_a); free(host\_b); free(host\_c);

return 0;

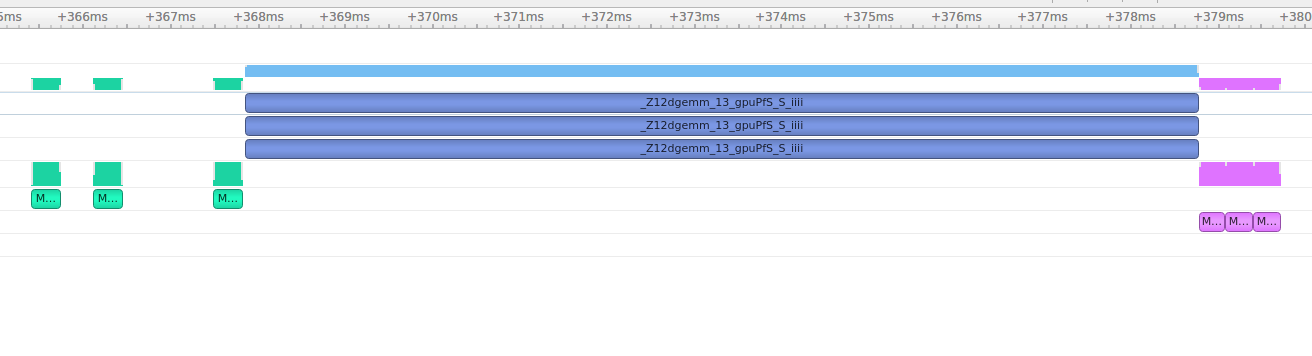
}



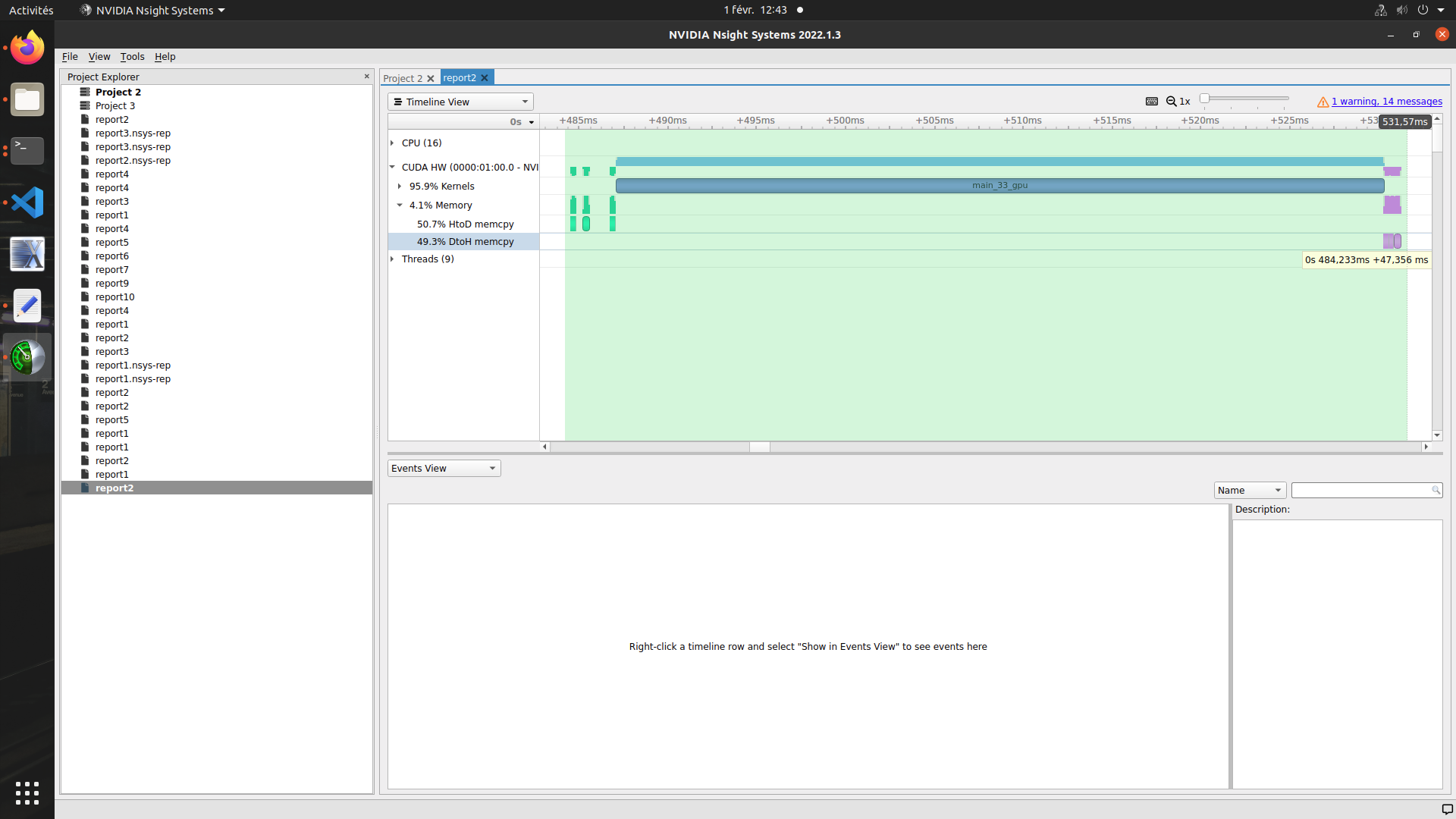
sans gang vect



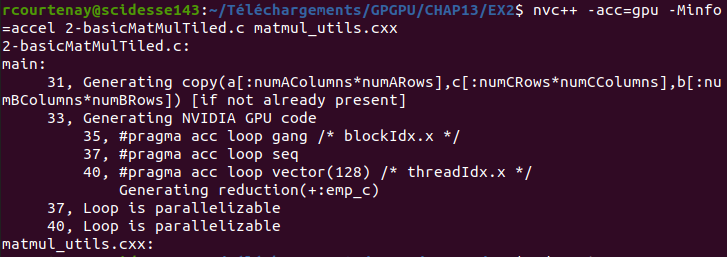
avec gang vect



REBECCA :



version1 :



version 2 :

https://github.com/rmfarber/ParallelProgrammingWithOpenACC/blob/master/Chapter06/matrix-acc-tile.c