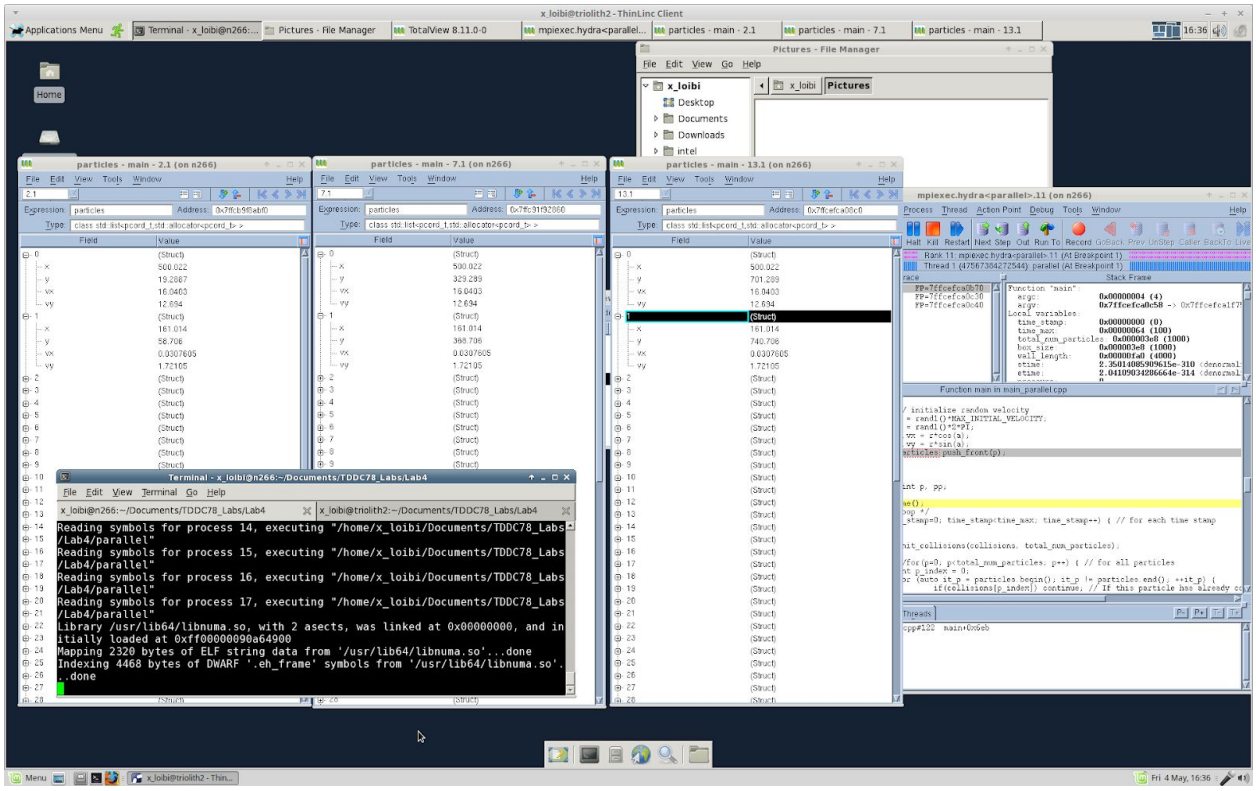
TDDC78 - Report Lab4 - loibi806 - goubh275

Tools

During the labs, we didn't had to use the tools at any point, the following examples are "created scenarios".

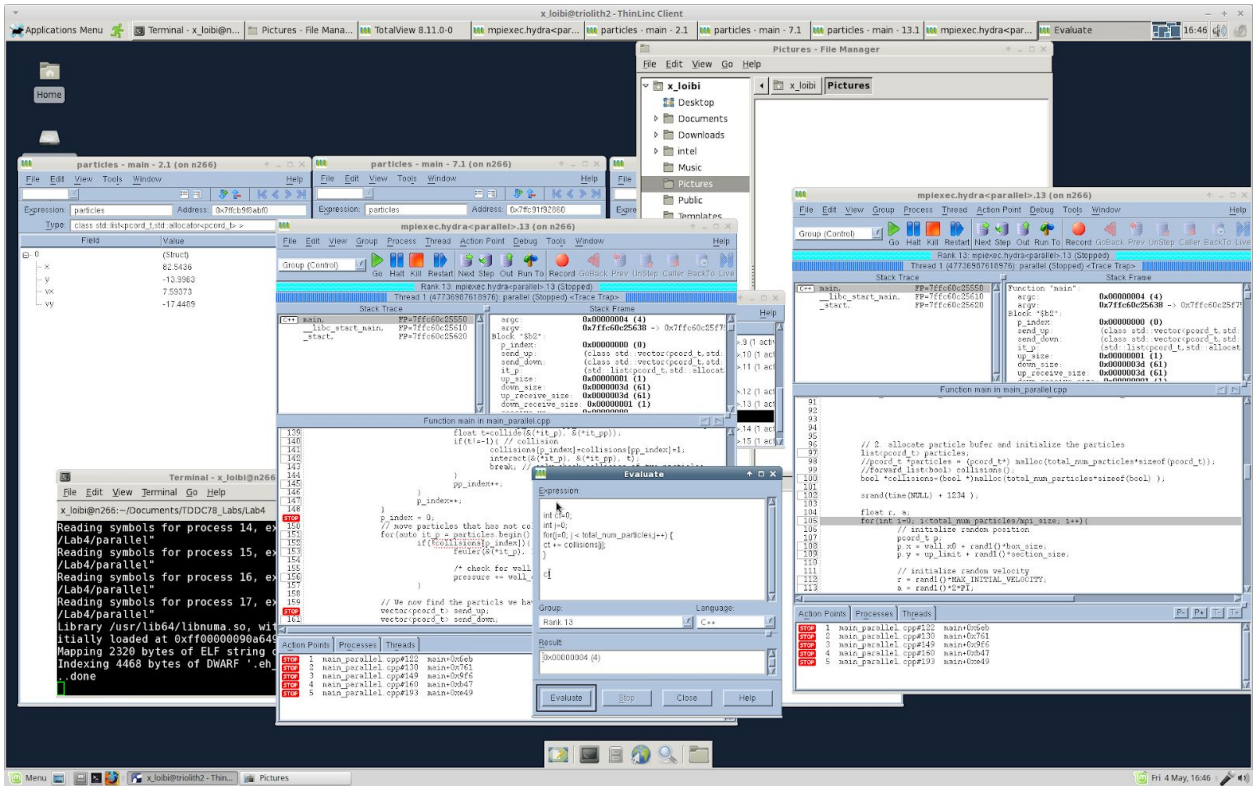
Totalview

We run total view on Lab 5, and decided to try some of the application functions, and explore our code to see if we could find some strange behaviour. It had the same classical features as other debugger (e.g. gdb), such as breakpoint, continue until breakpoint, next and step, that allow to move inside the program line by line, or until a critical section. It doesn't seem possible to put different breakpoints for each thread, but once all the threads have stopped at a breakpoint, we can go and look at each one's variables. We decided to look into different variable to see if we could spot a mistake, and we found a potentially problematic behavior. In Lab 5, each thread will generate random particle using *(time(NULL) + 1234)* as the random generator seed. The problem with that is that the particles will have the same repartition on each of the thread, making it from a global point not really random. We can see it in the following screenshot :



The right window is the main window from totalview, where we set a breakpoint after the generation of particles. We then looked into the *list<pcord_t>* of particles for threads 1, 6 and 12, and we can see the values inside the 3 left windows. If you look at the first 2 particles generated for each thread, you can see that their x coordinate is the same for each thread For the y coordinate, we observe a similar behaviour, but with an offset since the complete volume is splitted by row.

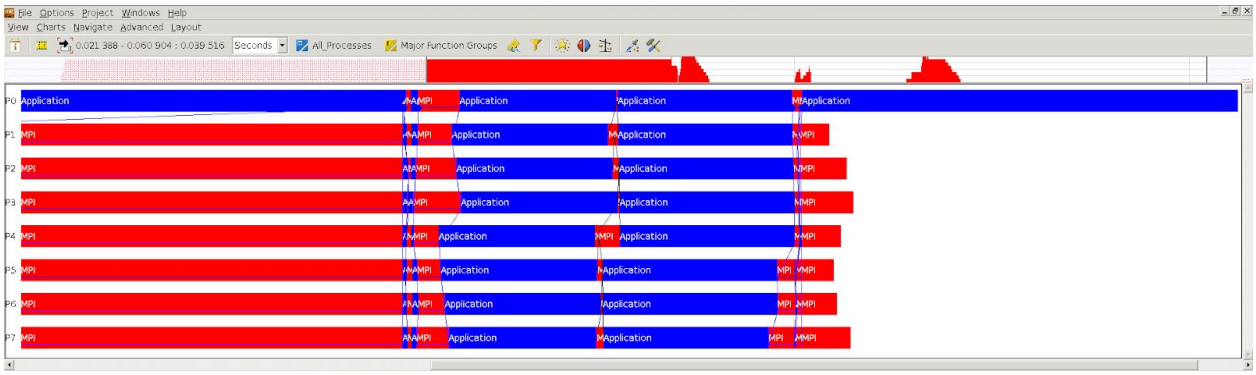
We tried to find other functions available, and we found one that seemed useful. Sometimes, it would be useful to add some code to understand better what is happening, such as computing the sum of an array instead of looking manually at all the values. We can do this using the *evaluate* functionality :



In Lab 5, we have an array that indicates the particles that have collided. We wanted to know how many of them were they, so we used the evaluate functionality and added a for loop that would go over this array and sum all the values. By evaluating this expression, we got a result, 4, meaning that 2 particles had collided in thread 13.

ITAC

We decided to trace Lab1 - blur with ITAC. We run it without adding custom sections, and we got the following timeline :



We can clearly see what is happening. The first third of the program correspond to the initialization. All the workers have already been created, and the workers 0 is currently reading the image from the file (Hence the blue/application section). All the other workers are currently waiting, hence the red/MPI section. The small blue column correspond to the moment where each workers computes its row boundaries. It is followed by an MPI section in red, were the image is scattered on all workers. Following this, we see a long blue/application section; This is when the horizontal blurring is happening. In the middle of the image, we have a new MPI section, that corresponds to the transfer of border rows to adjacent processes, followed again by a new application section (the vertical blur). We end with a gather form all workers resulting in the last red/MPI section, followed by a long application/blue section on the worker 0, that will write the image back, and print a few informations to the console.

Our code it almost perfectly balanced. The only problem comes from reading and writing the image. It can't be done in parallel, so only worker 0 does something, the other ones have to wait.

SOMETHING ELSE ?