

Name

Gender

My work in my organization

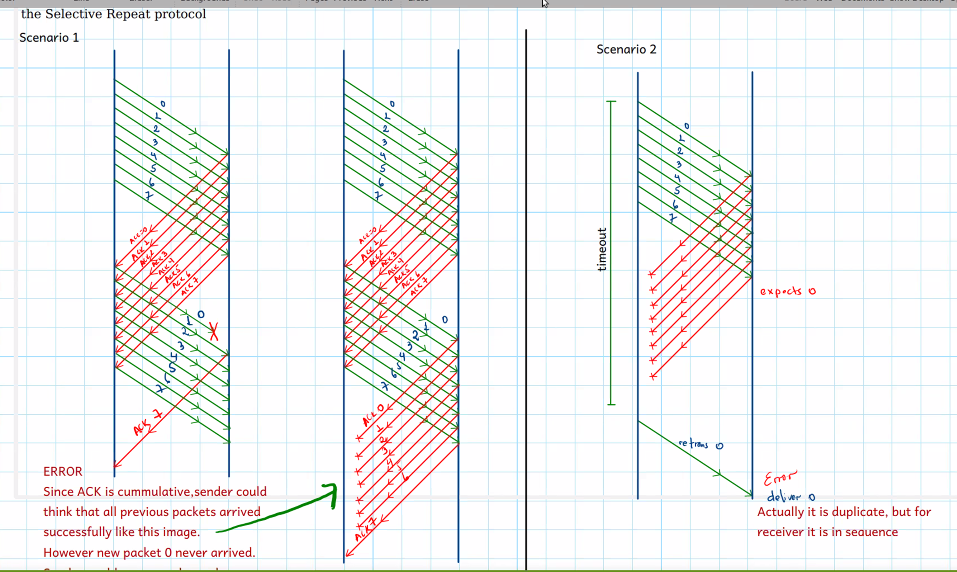
Second part

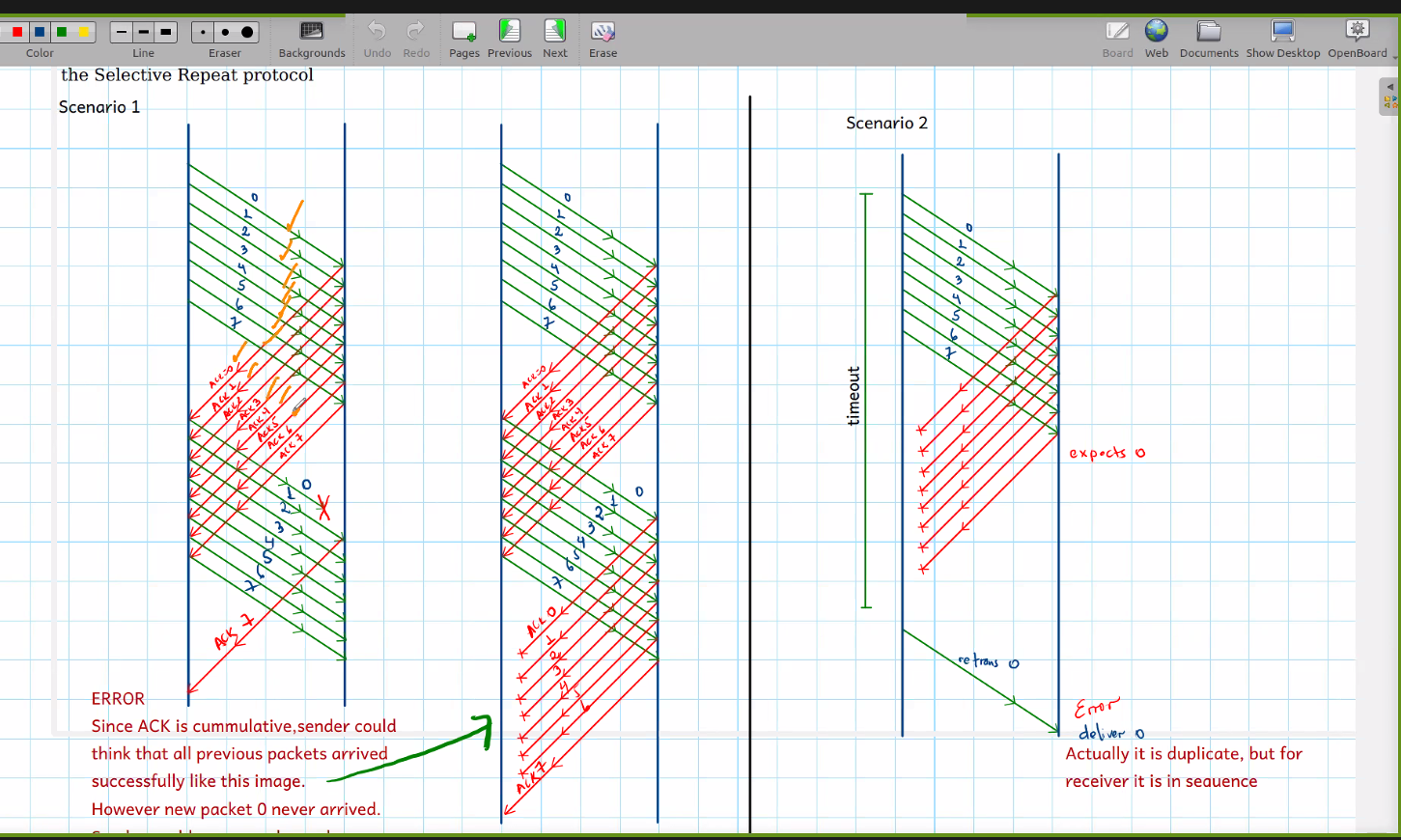
Wahte were the good practices or challenges at tthose moments

* The firewall could be blocking the request from Nessus and OpenVM to my personal machine, and could be the reason why we are not obtaining any vulnerability.
* We believe it could be due to the configuration of Nexus and OpenVM. However, with the other virtual machines it worked correctly.
* We will explore future analisys on the machine in order to have a better understanding and a report on this.

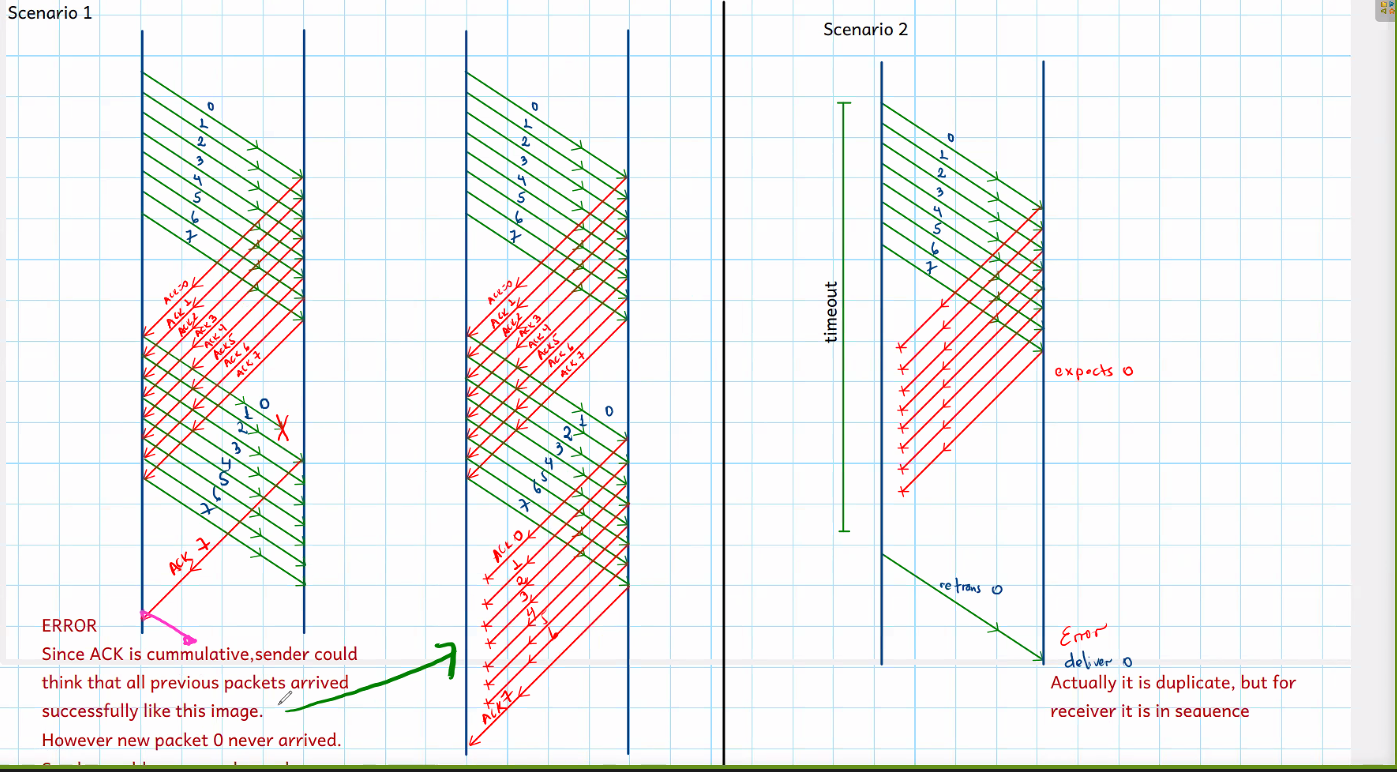


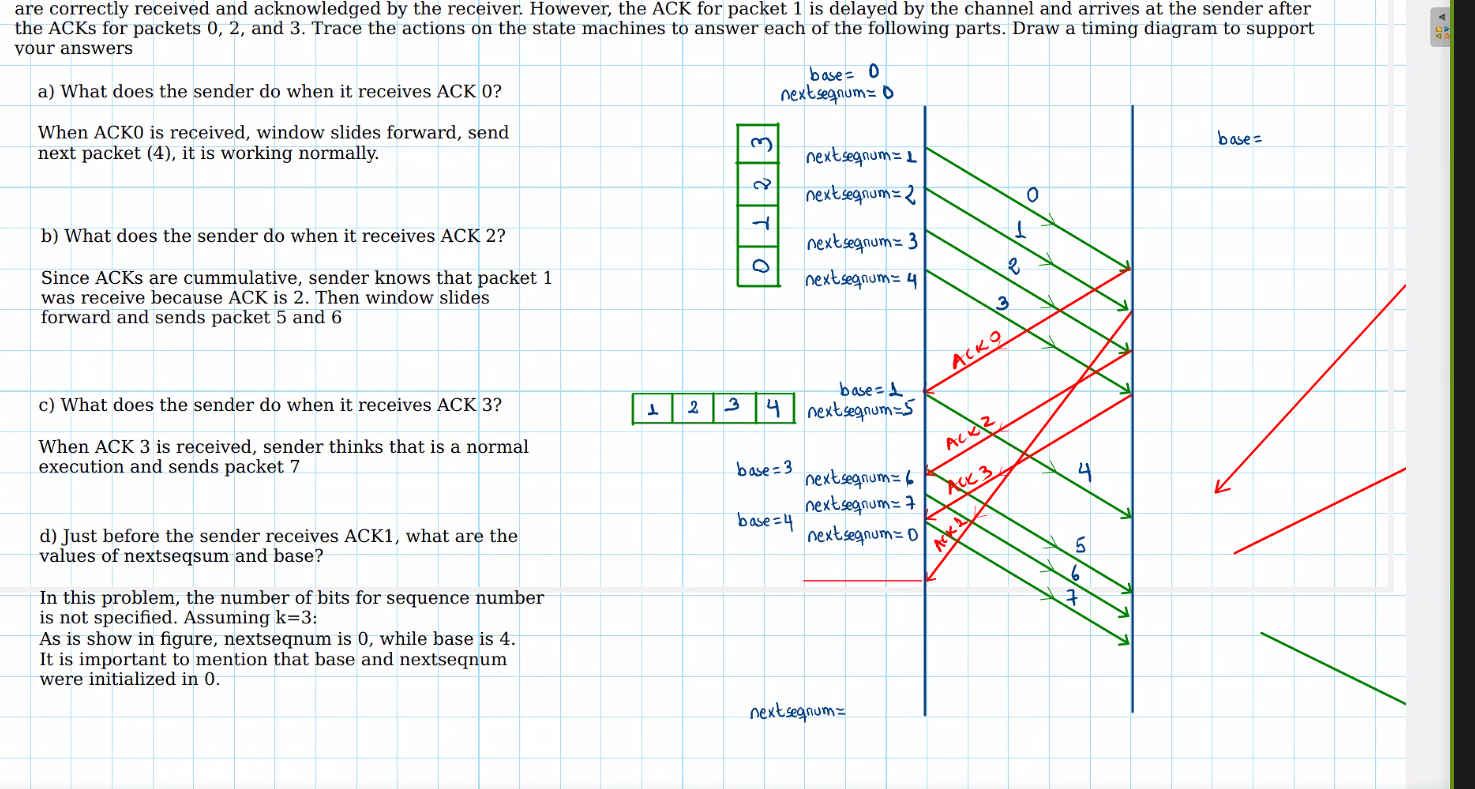
Hola a todos,



. 

.





CG

Run: ./cetus -parallelize-loops=4 cg.c

BT:

./cetus -parallelize-loops=1 add.c adi.c bt.c error.c exact\_rhs.c exact\_solution.c initialize.c rhs.c set\_constants.c solve\_subs.c verify.c x\_solve.c y\_solve.c z\_solve.c

./cetus bt.c initialize.c exact\_solution.c exact\_rhs.c set\_constants.c adi.c rhs.c x\_solve.c y\_solve.c solve\_subs.c z\_solve.c add.c error.c verify.c ../common/print\_results.c ../common/c\_timers.c ../common/wtime.c

DC:

./cetus -parallelize-loops=4 adc.c dc.c extbuild.c jobcntl.c rbt.c

EP

./cetus -parallelize-loops=1 ep.c

IS

./cetus -parallelize-loops=1 is.c

LU

./cetus -parallelize-loops=0 lu.c read\_input.c domain.c setcoeff.c setbv.c exact.c setiv.c erhs.c ssor.c rhs.c l2norm.c jacld.c blts.c jacu.c buts.c error.c syncs.c pintgr.c verify.c ../common/print\_results.c ../common/c\_timers.c ../common/wtime.c

./cetus -parallelize-loops=4 blts.c buts.c domain.c erhs.c error.c exact.c jacld.c jacu.c l2norm.c lu.c pintgr.c read\_input.c rhs.c setbv.c setcoeff.c setiv.c ssor.c verify.c

./cetus -parallelize-loops=4 lu.c read\_input.c domain.c setcoeff.c setbv.c exact.c setiv.c erhs.c ssor.c rhs.c l2norm.c jacld.c blts.c jacu.c buts.c error.c pintgr.c verify.c ../common/print\_results.c ../common/c\_timers.c ../common/wtime.c

gcc -O3 -fopenmp -mcmodel=large -o output lu.c read\_input.c domain.c setcoeff.c setbv.c exact.c setiv.c erhs.c ssor.c rhs.c l2norm.c jacld.c blts.c jacu.c buts.c error.c pintgr.c verify.c ../common/print\_results.c ../common/c\_timers.c ../common/wtime.c -lm

MG

./cetus -parallelize-loops=4 mg.c

SP

./cetus -parallelize-loops=0 add.c adi.c error.c exact\_rhs.c exact\_solution.c initialize.c ninvr.c pinvr.c rhs.c set\_constants.c sp.c txinvr.c tzetar.c verify.c x\_solve.c y\_solve.c z\_solve.c.

./cetus -parallelize-loops=0 add.c adi.c error.c exact\_rhs.c exact\_solution.c initialize.c ninvr.c pinvr.c rhs.c set\_constants.c sp.c txinvr.c tzetar.c verify.c x\_solve.c y\_solve.c z\_solve.c. ../common/print\_results.c ../coon/print\_results.c ../common/randdp.c ../common/c\_timers.c ../common/wtime.c

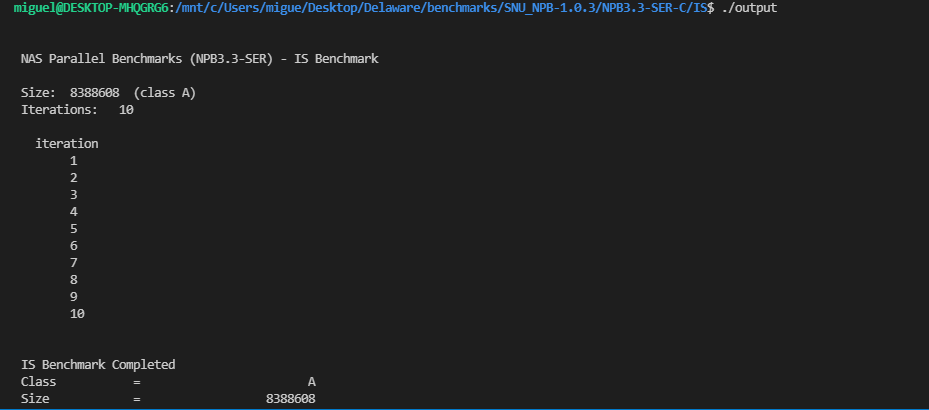
FT:

./cetus -parallelize-loops=4 appft.c auxfnct.c fft3d.c mainft.c verify.c

Executable files:

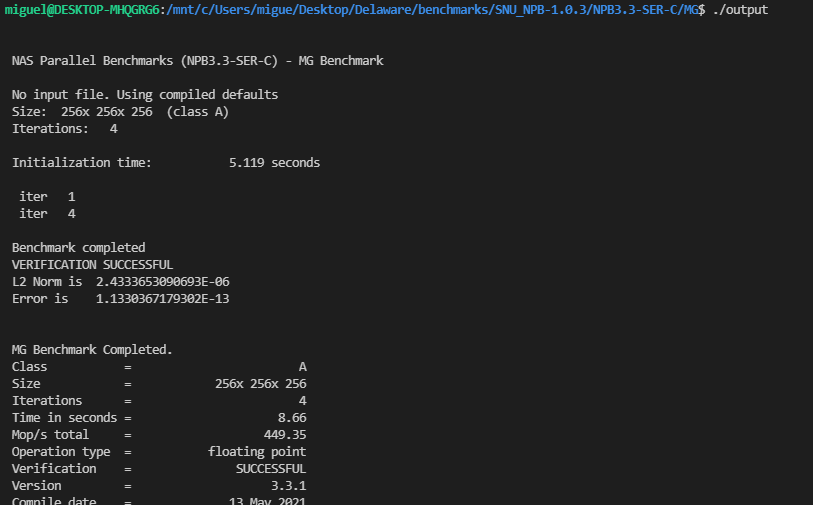
**IS BENCHMARK**

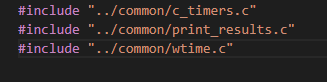
**Command: gcc -fopenmp -o output is.c**



**Issues on MG Benchmark:**

**Command: gcc -fopenmp -o output cetus\_output/mg.c -lm**

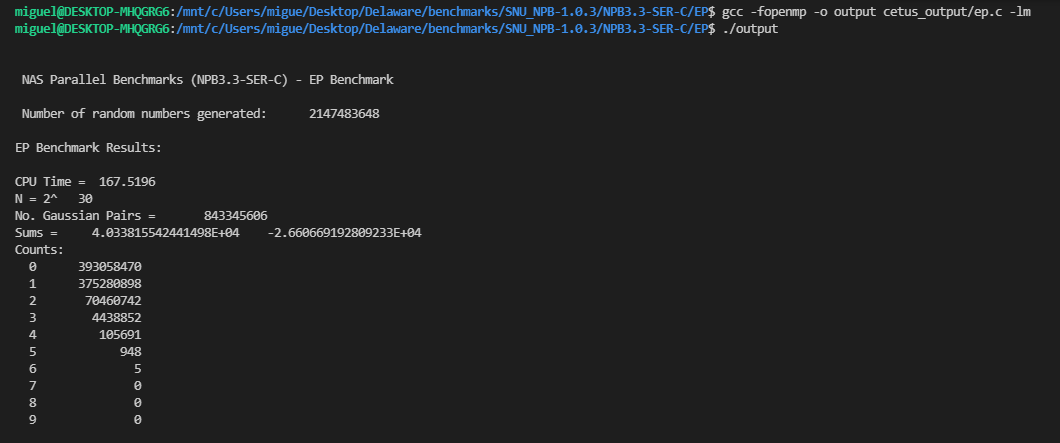




**EP Benchmark**

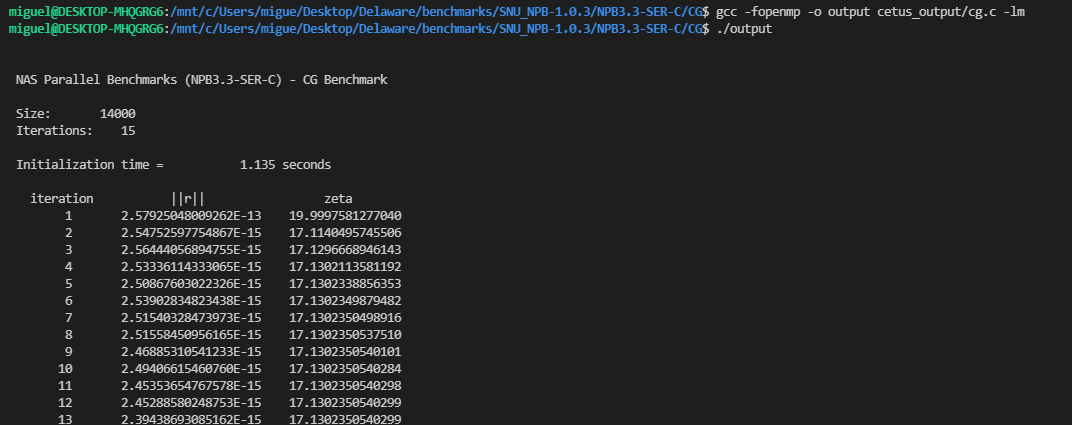
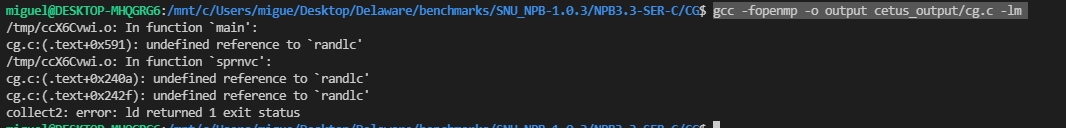
Command: gcc -fopenmp -o output cetus\_output/ep.c -lm

Picture:



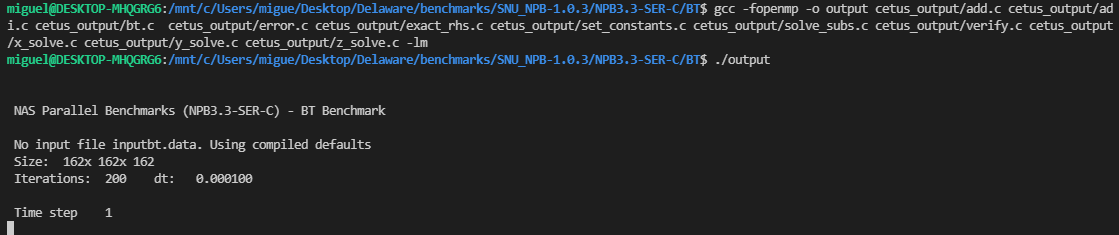
**CG Benchmark:**

Command: gcc -fopenmp -o output cetus\_output/cg.c -lm

**BT Benchmark**

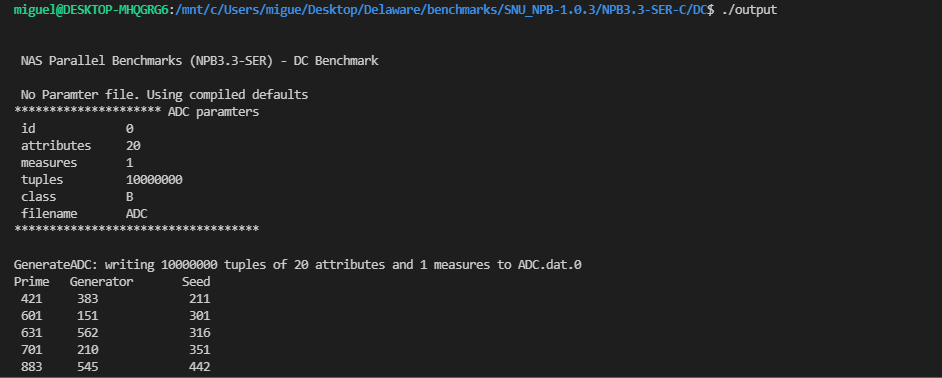
Command: gcc -fopenmp -o output cetus\_output/add.c cetus\_output/adi.c cetus\_output/bt.c cetus\_output/error.c cetus\_output/exact\_rhs.c cetus\_output/exact\_solution.c cetus\_output/initialize.c cetus\_output/rhs.c cetus\_output/set\_constants.c cetus\_output/solve\_subs.c cetus\_output/verify.c cetus\_output/x\_solve.c cetus\_output/y\_solve.c cetus\_output/z\_solve.c -lm



**DC Benchmark:**

Command: gcc -fopenmp -o output cetus\_output/adc.c cetus\_output/dc.c cetus\_output/extbuild.c cetus\_output/jobcntl.c cetus\_output/rbt.c -lm

Executable file:

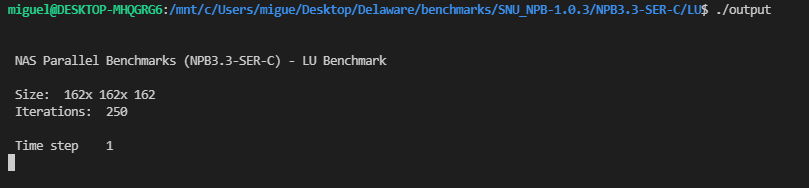


**FT Benchmark:**

Command:

**LU BENCHMARK:**

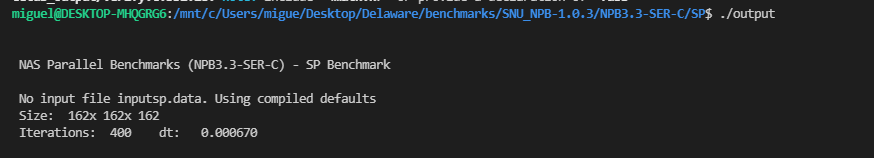
Command: gcc -fopenmp -o output cetus\_output/blts.c cetus\_output/buts.c cetus\_output/domain.c cetus\_output/erhs.c cetus\_output/error.c cetus\_output/exact.c cetus\_output/jacld.c cetus\_output/jacu.c cetus\_output/l2norm.c cetus\_output/lu.c cetus\_output/pintgr.c cetus\_output/read\_input.c cetus\_output/rhs.c cetus\_output/setbv.c cetus\_output/setcoeff.c cetus\_output/setiv.c cetus\_output/ssor.c cetus\_output/verify.c -lm



**SP BENCHMARK:**

Command: gcc -fopenmp -o output cetus\_output/add.c cetus\_output/adi.c cetus\_output/error.c cetus\_output/exact\_rhs.c cetus\_output/exact\_solution.c cetus\_output/initialize.c cetus\_output/ninvr.c cetus\_output/pinvr.c cetus\_output/rhs.c cetus\_output/set\_constants.c cetus\_output/sp.c cetus\_output/txinvr.c cetus\_output/tzetar.c cetus\_output/verify.c cetus\_output/x\_solve.c cetus\_output/y\_solve.c cetus\_output/z\_solve.c -lm

Command: gcc -fopenmp -o output add.c adi.c error.c cetus\_output/exact\_rhs.c xact\_solution.c nitialize.c ninvr.c pinvr.c hs.c set\_constants.c sp.c txinvr.c tzetar.c verify.c x\_solve.c y\_solve.c z\_solve.c -lm



Homework

Make a plan for the paper, we want to be ready in September

An interesting project would be major in the current version of cetus what are the effects of these transformations, we have done it twice so far, the blue tony paper And the Mustafa paper that have major it about 10 years, would be awesome to have another paper like this, but also including some of the transformations that loop inner change.

1. Getting more familiar with the tunning environment, you have done some of this already right?

Mustafa really measure the effect of individual transformation. When the professor says individual transformations what does he mean with that?

Mustafa found the best combination of techiniques by measuring and tunning the programs at run time.

The professor mentioned that you have a some kind of tunning environment inside cetus that will allow me to do such as experiments .

Run many many versions of the program and find which is the best combination of techniques.

With the variant experiment control language you could say run the experiments so you take the best and switch off an individual technique.

The professor mentioned that you have an environment where those experiments can easily be created. Where is that? How can I test it? How can I start?

1. Do you know how Mustafa worked to majoring the effect of the various transformations’ techniques? R// You measure the effect of a transformation by measuring two things, one is the performance of the program with full optimization and the second one is performance of the program with one optimization switch off and then the performance difference gives the measure of the power of that transformation.
2. Which applications will I use to test it? NAS PARRALL BECNHMARKS
3. Where can I find another suit benchmarks?
4. What is a better benchmark suit? What benchmarks suits do you want to measure your results on?
5. Which benchmarks was previously tested on Mustafa paper? I think he used the Nas Parallel benchmarks and the expect OMP benchmarks I believe
6. If I am going to use the parallel benchmarks why I am going to do all the experiments again? Because we have new machines 10 tears later and new techniques.
7. What is going to be the mayor contribution of this paper?
8. How the findings change when we go to different benchmarks?
9. I need to ask Akshay if the optimization options are the same than the techniques they were talking about in the meeting, ejem privatization, is that relevant in todays benchmarks and improving the performance?

Questions

1. Getting more familiar with the tunning environment, the professor said that I had done that before. For example: Run many many versions of a program and find which is the best combination of techniques.

The professor mentioned that you have an environment where those experiments can easily be created. Where is that? How can I test it? How can I start

1. I know that Mustafa really measure the effect of individual transformation. When the professor says individual transformations what does he mean with that?
2. Which applications will I use to test it? Mustafa used NAS PARRALL BECNHMARKS AND SPEC OMP BENCHMARKS2 012. As far as I know The IS, ART, and EQUAKE benchmarks have low parallel coverage, these are not amenable to effective automatic parallelization.
3. Where can I find another suit benchmarks?
4. What is a better benchmark suit? What benchmarks suits do you want to measure your results on?
5. The optimization options that you show me on CETUS are the same than the techniques you were talking about in the meeting, example privatization.
6. Mustafa mentioned in his paper that hid framework currently tunes two optimization options: loop parallelization and function inlining. Which ones should I use in my work?
7. What is a tunning window? Mustafa mentioned in his paper that the tunning window size is the number of loop nests it contains.
8. In Mustafa paper he mentioned that CETUS flexible support for inlining. What is a inlining function call and how can I enable inlining in CETUS? I know that inlining replaces a subprogram invocation with a copy of the invoked subprogram (if the invoked and invoking subprograms are in the same program unit)
9. What is a instrumentation call? I know that the instrumentation can be enable/disable by compiler flags
10. I did not understand the difference between the parallel coverage and speedup. Based on what I learned parallel coverage exhibits the potential parallel performance while speedup shows actual parallel performance.
11. Mustafa in his paper used the Intel icc compiler version 11.1 and gcc com-piler version 4.1.2. Which compilers should I use in my project?
12. Mustafa mentioned that he used the W data set in the NPB Benchamarks and the train data set in SPEC OMP 2001. Which data sets should I use in my study, A, B, C D?
13. In the study they conducted experiments using a single-user x86-64 machine with two 2.5 GHz Quad-Core AMD 2380 processors and a 32GB memory. Should I only use a type of computer to test the experiments?

Login into DARWIN

ssh [miguelro@darwin.hpc.udel.edu](mailto:miguelro@darwin.hpc.udel.edu)

scp -r tuning10.jar miguelro@darwin.hpc.udel.edu:MiguelEnvironment/TestNewScript/

password: MikeAngel@31860

cd SNU\_NPB-1.0.3/NPB3.3-SER-C

cd MiguelEnvironment/TestNewScript

cd SNU\_NPB-1.0.3/NPB3.3-SER-C/BT

See the jobs in our partition

sproject jobs -g parot --detail

sworkgroup --workgroup=parot --limits

Login into our group partition

workgroup -g parot

scp -r CG miguelro@darwin.hpc.udel.edu:MiguelEnvironment/TestNewScript/SNU\_NPB-1.0.3/NPB3.3-SER-C

Login into Cavanass

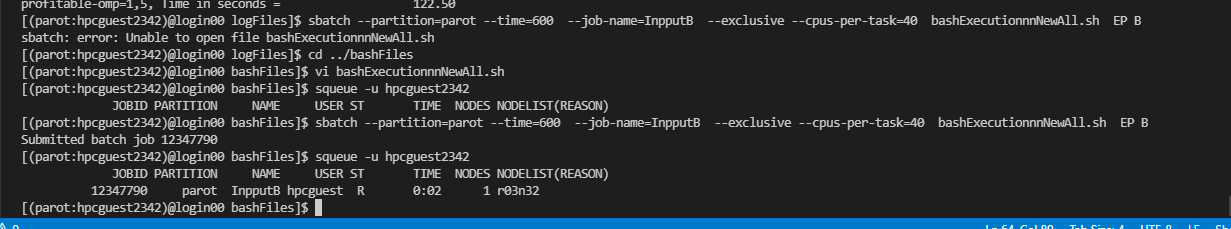
ssh [hpcguest2342@caviness.hpc.udel.edu](mailto:hpcguest2342@caviness.hpc.udel.edu)

password: X2uxlmoBCEuxWtRK

scp usuario1@dominio1.com:/home/usuario1/archivo.txt usuario2@dominio2.com:/home/usuario2/

scp [hpcguest2342@caviness.hpc.udel.edu:logFiles](mailto:hpcguest2342@caviness.hpc.udel.edu:logFiles) mike@192.168.0.16:/home

cd SNU\_NPB-1.0.3/NPB3.3-SER-C



scp -p logFiles [mike@192.168.0.16](mailto:mike@192.168.0.16):

Jobid example:17104513

acceder a grupo: workgroup -g parot

run job: sbatch bashCG\_Ap1i3.sh

Deleting job: scancel idjob

sbatch --partition=parot --time=300 bashExecutionC.sh

salloc --partition=devel --time=120 bashCompilationC.sh

salloc --partition=parot --cpus-per-task=40 --time=300

sbatch --partition=devel --mail-user=miguel980429@gmail.com --time=120 bashCG\_Ap1i3.sh

scontrol

scontrol is used for monitoring and modifying queued jobs, as well as holding and releasing jobs. One of its most powerful options is the scontrol show job option with the JobID.

scontrol show job 1354

sbatch --partition=devel --time=300 --cpus-per-task=4 bashExecutionA.sh

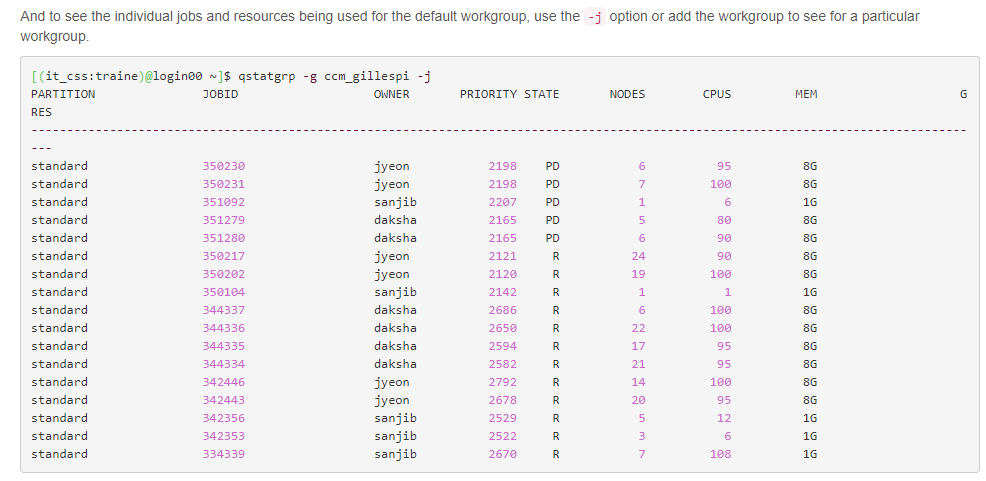
sbatch --partition=parot --time=300 --cpus-per-task=40 bashExecutionNewAllPoly.sh correlation

scp -r CG [hpcguest2342@caviness.hpc.udel.edu](mailto:hpcguest2342@caviness.hpc.udel.edu):MiguelTestEnvironment/SNU\_NPB-1.0.3/NPB3.3-SER-C

scp ~~r SanityCheck hpcguest2342@caviness.hpc.udel.edu:MiguelR-HC~~

squeue -u hpcguest2342

salloc --partition=parot --time=300 --exclusive



Command: qstatgrp -g parot -j

scontrol show partition parot

**Actual state of environment:** squota -g parot

**Limits of our partition:** sworkgroup -g parot --limits

EXECUTION AND COMPILATION

sbatch --partition=parot --time=120 bashCompilationAll.sh CG B

sbatch --partition=parot --time=300 --job-name=InpputA --nodelist=r03n32 --exclusive --cpus-per-task=40 bashExecutionnnAll.sh CG A

sbatch --partition=parot --time=600 --job-name=InpputB --nodelist=r03n33 --exclusive --cpus-per-task=40 --mem= 187G bashExecutionnnAll.sh CG B

sbatch --partition=parot --time=600 --job-name=InpputC --nodelist=r03n33 --cpus-per-task=40 bashExecutionnnC.sh CG C

scontrol show partition parot

sinfo -o "%g %.10R %.20l"

salloc --job-name=test --partition= \_workgroup\_



Ask them if all nodes will have the exact same performance, tell them that you need to do very low level precise timing results that need to be consistence from one day to the next.

Measure the best combination of options and also the contribution of individuals techniques.

Variation. Is the correct word.

**It takes the file as input, reads line by line and for every line it calls Cetus with his options, creates an executable and then run the executable a number of times.**

**The actual scripts reads that line compile cetus with this options, creates an executable and runs it**

**How the script handle the make and execute parameters that you need for an application? I assume, if you are dealing just with the NAS Parallel benchmarks the make and execute information is probably the same for all benchmarks, the make is basically you compile cetus, and then you compile with gcc.o3 or something like this you make an executable and then you run this executable, and to run it your probably do not need an input file.**

**The compilation information needs class A,B,and C because this is hard, and when you execute I believe they do not need and additional input file**

Hello,

I have a question, each partition has a specific node where the jobs are running on or it has more than one, for example Does Parrot partition have more than one node? And if the answer is that we have many nodes, my next question would be, all the nodes will have the exact same performance? I need to do very low-level precise trimming results that new to be consistence from one day to the next.

Best,

I told them that I need to do very low-level precise timing results that need to be consistent from one day to the next, but they told me that on an HPC cluster that's just not easy to ensure, because there are multiple generations of node in Caviness, with Intel processors from different generations of that company's CPU roadmap. Also, there's an operating system and HPC management infrastructure running on all nodes that will, from time to time, consume CPU cycles .

It means that if a job does not occupy a full node other jobs may also be running on it, sharing CPU cache and memory bandwidth with my job. So,

Assuming I am able to ensure the same generation of node in Caviness for each run and each run makes exclusive use of the entire node, and I eliminate all network-shared i/o from the critical region(s) that I am timing, then my statistics should not very different from day to day.

**Analysis Techniques**

Alias=N

=0 assume all locations are aliased

=1 advanced interprocedural analysis (default)

=3 assume no alias exists

-ddt=N

Perform Data Dependence Testing

=1 banerjee-wolfe test

=2 range test

-parallelize-loops=N

Annotate loops with Parallelization decisions

=0 do not parallelize

=1 parallelizes outermost loops

-privatize=N

Perform scalar/array privatization analysis

=0 force to disable

=1 enable only scalar privatization

=2 enable scalar and array privatization

-range=N

Specify the accuracy of symbolic analysis with value ranges

=0 disable range computation (minimal symbolic analysis)

=1 enable local range computation (default)

=2 enable inter-procedural computation (experimental)

-reduction=N

Perform reduction variable analysis (ON=2)

=0 force to disable

=1 enable only scalar reduction analysis

=2 enable array reduction analysis and transformation

**Transformation Analysys**

-induction=N

Perform induction variable substitution (ON=3)

=0 force to disable

=1 enable substitution of linear induction variables

=2 enable substitution of generalized induction variables

=3 enable insertion of runtime test for zero-trip loops

-normalize-loops

Normalize for loops so they begin at 0 and have a step of 1

=0

=1

-profitable-omp

Inserts runtime for selecting profitable omp parallel region

=0 disable

=1 Model-based loop selection

**Alias=N**

=0 assume all locations are aliased

=1 advanced interprocedural analysis (default)

=3 assume no alias exists

Alias:

./cetus -tinline=2:fuctions=x\_solve,y\_solve,z\_solve add.c adi.c error.c exact\_rhs.c exact\_solution.c initialize.c ninvr.c pinvr.c rhs.c set\_constants.c sp.c txinvr.c tzetar.c verify.c x\_solve.c y\_solve.c z\_solve.c ../common/print\_results.c ../common/randdp.c ../common/c\_timers.c ../common/wtime.c

Loops taeking the majority of Time:

LU:

Class=ssor.c

Line: 129 For-Loop

The important loop in line 129 has a function call and cetus cannot parallelize loops with function calls, in the hand parallel version it has been parallelized

MG: Class= mg.c

Function= resid

Line: 613

SP: CLASS: Function

**compute\_rhs(),**

**And also:**

**x\_solve(): Line 48**

**y\_solve(): Line 48**

**z\_solve(): Line 48**

**These important loops have function calls inside, and the hand parallel version is parallelizing the outer most loops, but cetus only parallelize the inner loops.**

**CG: CLASS: cg.c**

**Line: Loop 487**

**BT: The loops that are taking the majority of time are in the classes:**

**X\_solve: Line 70**

**Y\_solve: Line 69**

**Z\_solve: Line 69**

**These important loops have function calls inside, and the hand parallel version is parallelizing the outer most loops, but cetus only parallelize the inner loops.**

**IS: The loop tha is taking the majority of time is in the**

**Line: 745.**

**Cetus is not parallelizing the loop that is taking the majority of time, for that reason we are obtaining the same time per each technique. On the other hand, the hand parallel version is not parallelizing the loop either.**

**EP: Line: 182**

**BT:** For Bt there is not speedup for any of the techniques, the reason of that is because cetus is not parallelizing the outermost loop that takes the highest time, so it means BT is a really good candidate for inlining. We can also say that for BT for each techniques the time obtanined is similar to the serial code.

**EP:** Enabling privatization makes the most impact in the performance, because for most of the techniques we are obtaining a speedup similar to 1,22 , but when we enable privatization we obtain a speedup of 1,24

**LU:** For LU Cetus does not match with the time of the serial code.

**CG:** cetus is performing well in CG, and the most importan technique is privatization.

**SP:** We can say that for most of the techniques the performance matches with the serial code, and also we can say that privatization is the most important techniques, because when we disable privatization the performance drops.

Summary:

**Identifying Pitfalls in Automatic Parallelization of NAS Parallel Benchmarks**

S.Prema, R. Jehadeesan and Panigrahi prensented an examination of the efficiency of auto-parallelizes on the NAS parallel benchmarks and spotlighted the limitations of them while parallelization of the serial codes. On the other hand, on his work is being underlined the need of a user-interactive environment that highlights the issues found during parallelization. It also underscores the importance of minimal human interventions regarding code changes to resolve the problematic code section to parallelize the code. Furthermore, the authors concluded that ICC and Par4all could parallelize the Nas Parallel Benchmarks without any human intervention, but CETUS and ROSE required minimal intervention to obtain an efficient parallelization.

**Source-to-Source Parallelization Compilers for Scientific Shared-Memory Multi-core and Accelerated Multiprocessing: Analysis, Pitfalls, Enhancement and**

**Potential**

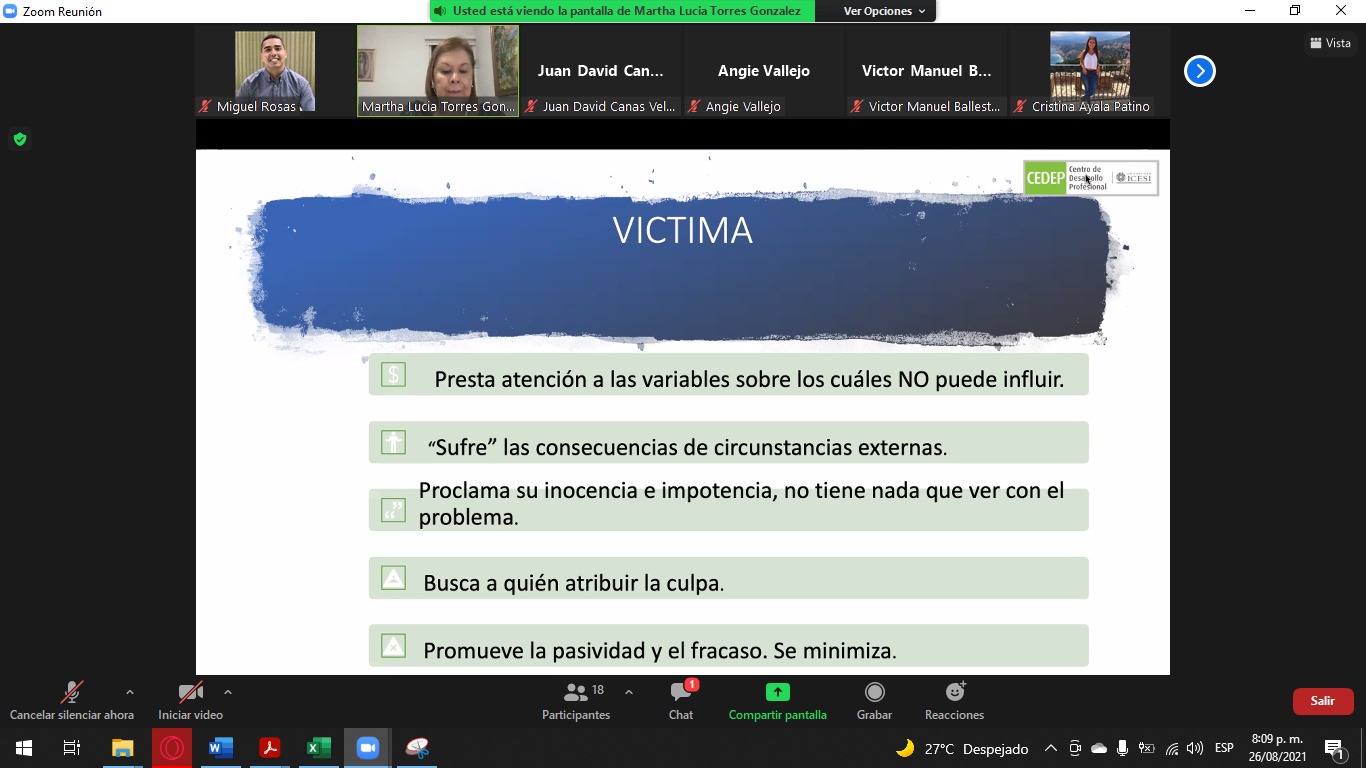
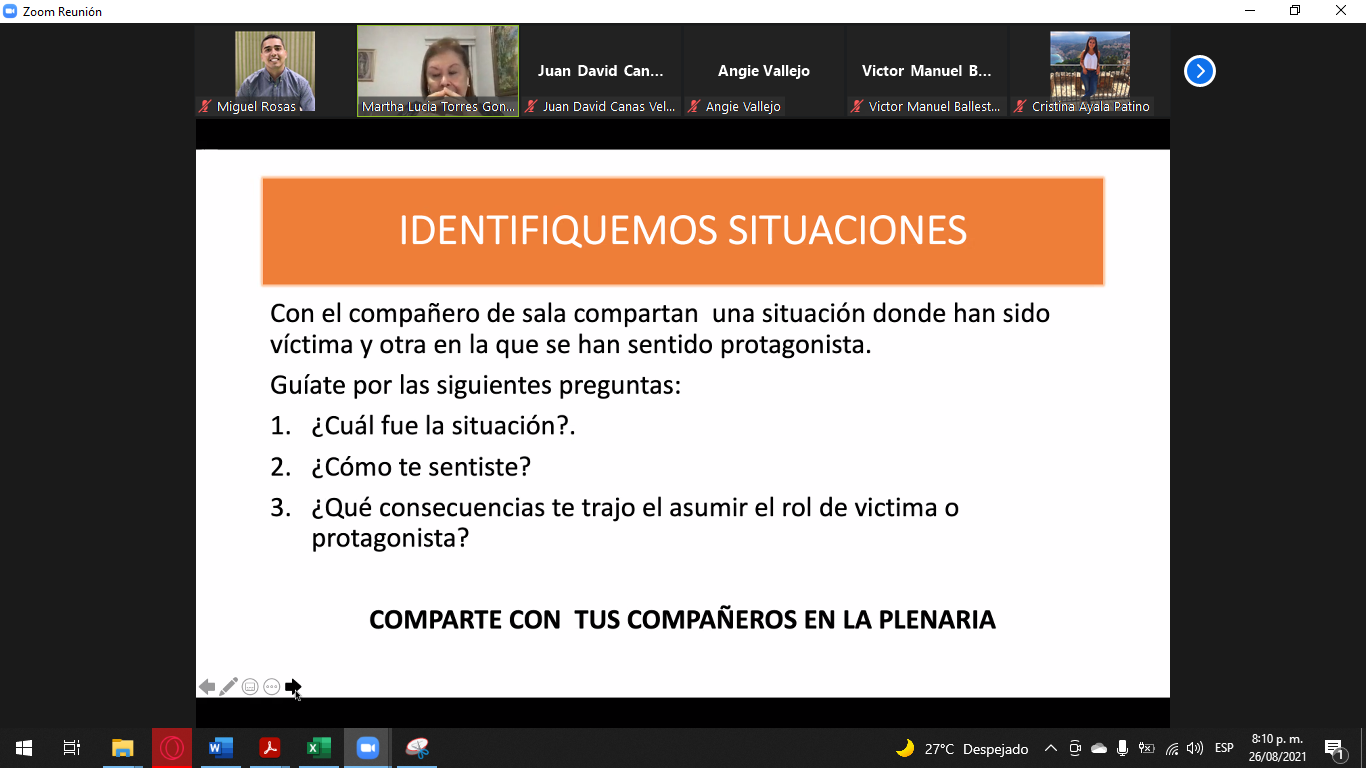
This paper focuses on three sources to source compilers, AutoPar, Par4All, and Cetus. They analyzed their performances, inspect their capabilities and suggested new paths for enhancement. Also, the authors described the need for parallelism in modern architectures and the difficulty of integrating that on the existing codes. The compilers were compared regarding their performance and the authors concluded that Cetus can provide a great service not only for Linux users, but also for the operating systems. Also, Cetus is able to generate reduction clauses on arrays, but this reduction clause could be invalid, causing a critical problem to compile the code. For Par4All was found that this compiler can handle most of the cases automatically without minimal user intervention.

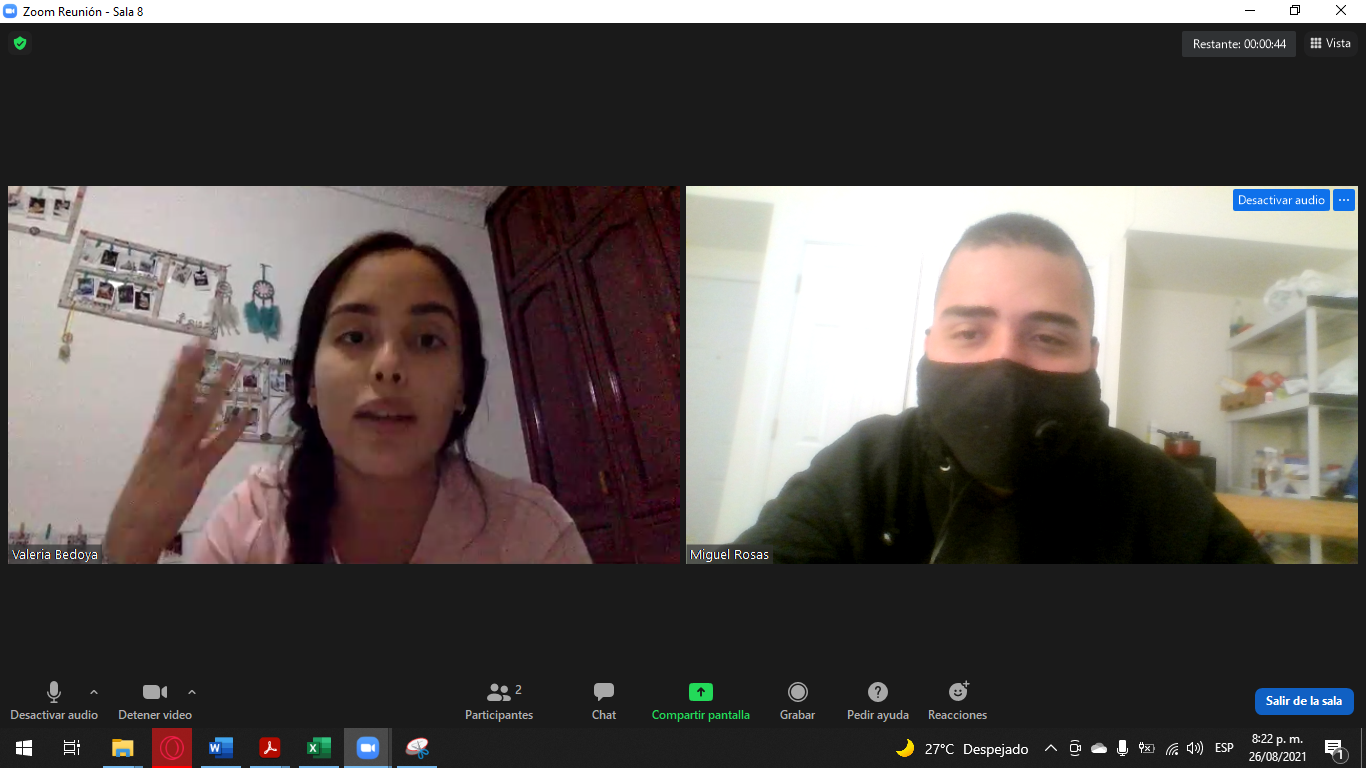
**AutoParBench: A Unified Test Framework for OpenMP-based Parallelizers**

On this paper the authors presented a framework to test OpenMP based automatic parallelization tools called AutoParBench. The main idea of the framework is a representation called “JSON snapshot”, whose objective is normalizing the output by auto-parallelizes. The authors of this paper designed a representation that normalizes programs annotated with OpenMP 4.5 pragmas. AutoParBench contains the necessary infrastructure to compare the output of different automatic parallelization tools. This framework provides developers with a collection of 99 programs, with 1579 loops whose code contain OpenMP directives manually annotated. The Authors concluded that AutoParBench has allowed them to discover several bugs on the automatic parallelization tools that were acknowledged by their developers.

**Performance Analysis and Tuning of Automatically Parallelized OpenMP Applications.**

On this paper the authors combined automatic parallelization with tunning techniques as an alternative to manual parallelization of sequential programs to explore the increased computational power that current multi-core systems offers. Also, they studied the gap performance between automatic and hand parallel OpenMP applications. They implemented an empirical tunning framework and proposed an algorithm that partitions program into different sections and tunes each one section individually. Besides, they presented results of automatically parallelizes and tuned versus the hand parallelized OpenMP programs on the NAS and SPEC OMP benchmarks. The authors concluded that partition the program into sections and tunes each section individually, significantly reduces the search space of optimizations variants and thus tunning time.

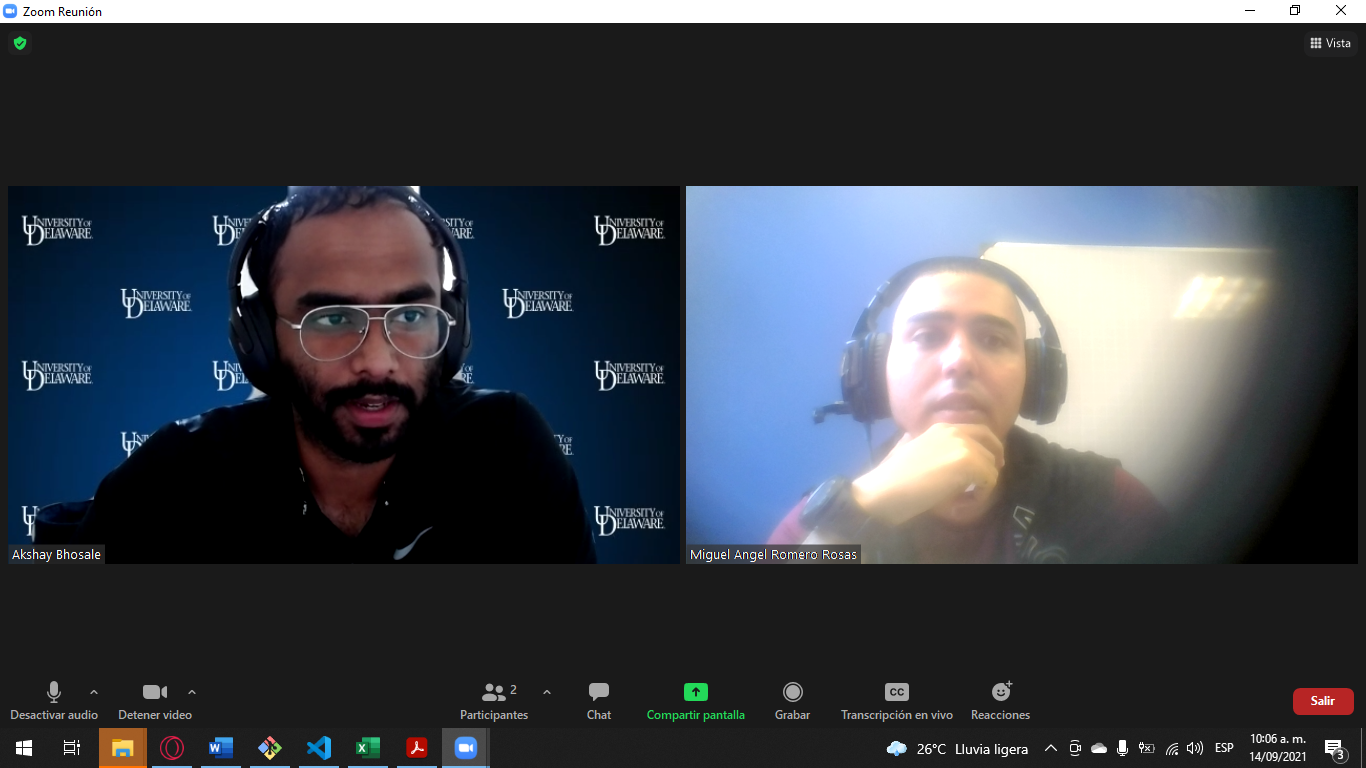
 

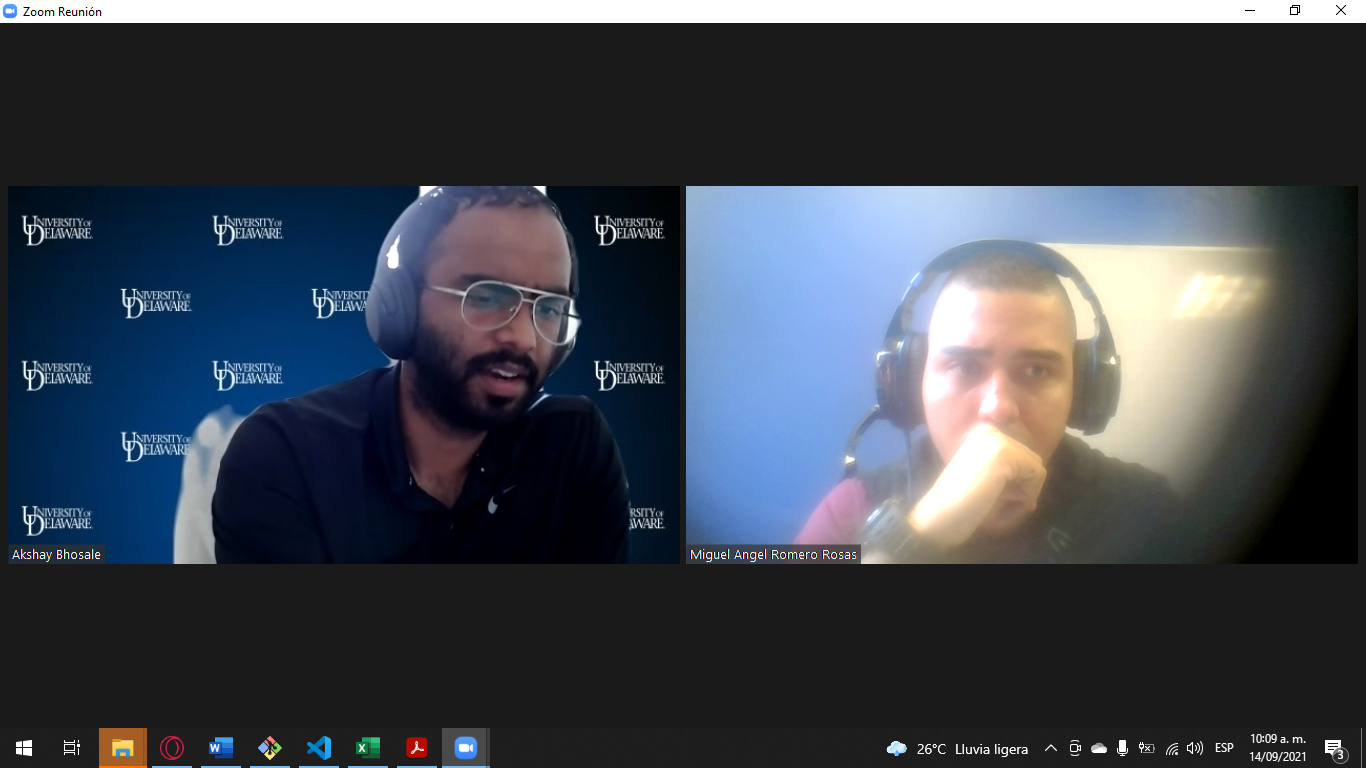


Polybench

gcc -O3 -I utilities -I ../gemm ../../../utilities/polybench.c gemm.c -o gemm\_Serial

gcc -O3 -I utilities -I ../gemm ../../../utilities/polybench.c cetus\_output/gemm.c -fopenmp -o gemm\_Parallel





The only differences are: **GCC 64 Bit O3 is marginally better than ICC 64 Bit O2**. The difference is less than 3%. ... The results are almost similar to Perlbench benchmark, the only difference between that ICC O2 & O3 have better runtime than corresponding GCC levels.

gcc -O3 -I utilities -I ../correlation ../../utilities/polybench.c correlation.c -o correlation\_Serial -lm

gcc -O3 -I utilities -I ../correlation ../../utilities/polybench.c -fopenmp cetus\_output/correlation.c -o correlation\_Parallel -lm