MicroSAT-CUDA: A GPU-based SAT solver experiment

Massimo Piedimonte (164719), Giovanni Rosa (164716)

---

1 Introduction

The Boolean satisfiability problem[1] (sometimes called propositional satisfiability problem and abbreviated SATISFIABILITY, SAT or B-SAT) is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. In other words, it asks whether the variables of a given formula can be consistently replaced by the values TRUE or FALSE in such a way that the formula evaluates to TRUE . If this is the case, the formula is called satisfiable. On the other hand, if no such assignment exists, the function expressed by the formula is FALSE for all possible variable assignments and the formula is unsatisfiable. There is no known algorithm that efficiently solves each SAT problem, and it is generally believed that no such algorithm exists, resolving the question of whether SAT has a polynomial-time algorithm is equivalent to the P versus NP problem, which is a famous open problem in the theory of computing. Nevertheless, there are heuristic SAT-algorithms that are able to solve problem instances. There are two types of SAT solvers, complete and incomplete, where in the first case there is the certainty that the formula is satisfiable or unsatisfiable, in the latter there is no guarantee that it will eventually either report a satisfying assignment or declare that the given formula is unsatisfiable. Nowadays, GPUs are mainly used for areas of application in which many operations must be performed in parallel. We want to exploit this feature for the resolution of boolean formulas by porting a SAT solver in GPU and running many parallel instances for the simultaneous resolution of multiple formulas. To achieve this, for first we need to choose a library to interact with the GPUs, namely OpenGL or CUDA. The main difference is that OpenGL is cross-device meanwhile CUDA works only on NVIDIA GPUs. We decided to use CUDA because is more powerful and faster than OpenGL and both the cluster and the desktop pc that we will use for the experimentation have NVIDIA GPUs. Next, we need to choose with SAT solver implementation we want to port on GPU.

We mainly focused on MiniSAT and MicroSAT, where despite MiniSAT is more powerful and the most used, MicroSAT on the other hand is faster and lighter. The main problem in using MiniSAT is that it uses libraries that are not available on CUDA, so in that case it would have been very difficult to make everything work on GPU.

2 Background and Related Work

2.1 SAT solvers

Since the SAT problem is NP-complete, only algorithms with exponential worst-case complexity are known for it. In spite of this, efficient and scalable algorithms for SAT were developed during the 2000s and have contributed to dramatic advances in our ability to automatically solve problem instances involving tens of thousands of variables and millions of constraints (i.e. clauses). Examples of such problems in electronic design automation (EDA) include formal equivalence checking, model checking, formal verification of pipelined microprocessors, automatic test pattern generation, routing of FPGAs, planning, and scheduling problems, and so on. A SAT-solver is now considered to be an essential component in the EDA toolbox.

2.1.1 DPLL

A DPLL SAT solver employs a systematic backtracking search procedure to explore the (exponentially sized) space of variable assignments looking for satisfying assignments. The basic search procedure was proposed in two seminal papers in the early 1960s [2][3] and is now commonly referred to as the Davis–Putnam–Logemann–Loveland algorithm ("DPLL" or "DLL"). The algorithm picks a variable and assigns a value (decision), next it uses various techniques to simplify clauses (i.e. unit propagation) and continues with decisions and simplifications until reaches sat/unsat or gets a conflict, where using backtracking goes back to the previous decision where it makes a different assignment. Many modern approaches to practical SAT solving are derived from the DPLL algorithm and share the same structure. Often they only improve the efficiency of certain classes of SAT problems such as instances that appear in industrial applications or randomly generated instances. Theoretically, exponential lower bounds have been proved for the DPLL family of algorithms. Stochastic methods try to find a satisfying interpretation but cannot deduce that a SAT instance is unsatisfiable, as opposed to complete algorithms, such as DPLL.

2.1.2 CDCL

Conflict-driven solvers, such as conflict-driven clause learning (CDCL)[4], augment the basic DPLL search algorithm with efficient conflict analysis, clause learning, non-chronological backtracking (a.k.a. backjumping), as well as "two-watched-literals" unit propagation, adaptive branching, and random restarts. These "extras" to the basic systematic search have been empirically shown to be essential for handling the large SAT instances that arise in electronic design automation (EDA). Well known implementations include Chaff and GRASP. Look-ahead solvers have especially strengthened reductions (going beyond unit-clause propagation) and the heuristics, and they are generally stronger than conflict-driven solvers on hard instances (while conflict-driven solvers can be much better on large instances which actually have an easy instance inside).

2.1.3 MiniSAT

Modern SAT solvers are also having significant impact on the fields of software verification, constraint solving in artificial intelligence, and operations research, among others. Powerful solvers are readily available as free and open source software. In particular, the conflict-driven MiniSAT[1], which was relatively successful at the 2005 SAT competition, only has about 600 lines of code. A modern Parallel SAT solver is ManySAT. It can achieve super linear speed-ups on important classes of problems. An example for look-ahead solvers is march_dl, which won a prize at the 2007 SAT competition.

2.1.4 MicroSAT

MicroSAT[2] is a simple CDCL SAT solver, originally created by Marijn Heule and Armin Biere. It aims at being very short (300 lines of code including comments), it has neither position saving nor blocking literals but is highly optimized and its heuristics work well together. It is based on the CDCL procedure and performs unit propagation using two-watched literals. Being very simple compared to other sat solvers is extremely fast in solving small formulas but given the lack of some heuristics is not very performing with increasing size

[1]https://www.msoos.org/tag/minisat

[2]https://github.com/marijnheule/microsat

```
loop {
   Propagate()
   if ( no conflict ) {
        IF (all Variables assigned) {
            return SAT
        } else {
            Decide()
        }
    } else {
        Analyze()
       if (top level conflict found) {
            Return UNSAT
       } else {
           learnClause()
           Backjump()
}
```

Figure 1: Microsat

2.2 GPU SAT solvers

There are different works using GPUs in SAT solving. Costa [5] proposed a SAT solver architecture based on modules. The idea is to use both GPU and CPU, running the SAT solver on the first but using the latter for conflict analysis and clause learning. Another example is the work of Beckers et al. [6], where they run a MiniSAT solver on CPU and a local search based TWSAT on GPU which is used as heuristics to choose literals and make assignments. Moreover, the work presented by Dal Palù et al. [7] describes a DPLL-based SAT solver which runs also on both CPU and GPU in different working modes, varying the use of GPU capabilities for both unit propagation and variable assignments. What we can deduce is that the standard approach is to use the CPU together with the GPU, to get the best from both. In fact, the limit of our approach is that it works only for small formulas, but what we want to understand is how many formulas can be solved contemporaneously with a GPU. There are other some simple implementations that are publicly available on GitHub. For example, *opencl-satsolver*[3] is a GPU SAT solver using OpenCL libraries. It is a simple brute force that parses the input expression, and tries every single variable configuration. Due to this, above 24-26 variables calculation times can be very long.

[3]https://github.com/zbendefy/opencl-satsolver

Moreover, *cuda-sat-solver*[4] and *ringsat*[5] are based on CUDA and inspired by DPLL. They use backtracking and unit propagation. The main difference between those implementations and our approach is that they solve a single formula, instead we use parallel instances of Microsat to solve multiple formulas simultaneously.

2.3 CUDA

GPU is specialized for compute-intensive, highly parallel computation and therefore designed mainly for data processing rather than data caching and flow control. The NVIDIA GPU architecture [8] is built around a scalable array of multithreaded Streaming Multiprocessors (SMs). When a CUDA program on the host CPU invokes a kernel grid, the blocks of the grid are enumerated and distributed to multiprocessors with available execution capacity. Threads that execute the same instruction constitute warps. Several warps constitute a thread block. Several thread blocks are assigned to a Streaming Multiprocessor. As thread blocks terminate, new blocks are launched on the vacated multiprocessors. A multiprocessor is designed to execute hundreds of threads concurrently. To manage such a large amount of threads, it employs a unique architecture called SIMT (Single-Instruction, Multiple-Thread). The instructions are pipelined to leverage instruction-level parallelism within a single thread, as well as thread-level parallelism extensively through simultaneous hardware multithreading. Unlike CPU cores they are issued in order and there is no branch prediction and no speculative execution. The main difference between CPUs and GPUs is that for the first case there are fewer powerful cores, for the latter there are a lot of cores with less power. This means that GPUs work best with a lot of small parallel operations.

2.3.1 Programming model

There are different CUDA architectures with a specific compute capability represented by version number (SM version) and different architectures, where the perfomance and the features are different. Moreover, to run CUDA applications, the main approach is to write CUDA functions (called kernel) that make calls to CUDA Runtime API to execute source code on GPU. There are different versions of CUDA APIs which are not strictly dependent on SM version. Applications compiled to work on a specific CUDA version

[4]https://github.com/QuentinFiard/cuda-sat-solver
[5]https://github.com/ichorid/ringsat

will continue to work on subsequent versions, but this does not apply in the opposite case. Kernel functions, when called, are executed n times in parallel by n different threads. Such threads, are defined by the programmer. The CUDA programming model [8] assumes that the CUDA threads execute on a physically separate device that operates as a co-processor to the host running the program, for example, written in C. the main thing that distinguishes in the source code a normal function from a cuda kernel is the use of special keywords that represent specific calls to CUDA APIs. For example, the keyword *__global__* identifies the main kernel function, meanwhile the keyword *__device__* identifies a function that can be called by a *__global__* kernel. In fig. 2 there is an example of simple CUDA program to sum two vectors, written in C.

```
1  // Kernel definition
2  __global__ void VecAdd(float* A, float* B, float* C) {
3    int i = threadIdx.x;
4    C[i] = A[i] + B[i];
5  }
6
7  int main() {
8     ...
9     // Kernel invocation of 1 Block with 16 threads
10    VecAdd<<< 1, 16 >>>(A, B, C);
11    ...
12 }
```

Figure 2: An example CUDA program for the sum of two vectors.

2.3.2 Threads, grids and blocks

In CUDA, threads can be organized in grids and blocks [8]. In detail, a block is a collection of threads and a grid is a collection of blocks. In this way each thread is a 3-component vector, that can be referenced using *threadIdx*, where there will be *threadIdx.x*, *threadIdx.y* and *threadIdx.z*. The SM creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps. A warp executes one common instruction at a time, so full efficiency is realized when all 32 threads of a warp agree on their execution path. So full efficiency is realized when all 32 threads of a warp branch to the same execution path. If threads of a warp diverge via a data-dependent conditional branch, the warp serially executes each branch path taken, disabling threads that are not on that path, and when all paths com-

plete, the threads converge back to the same execution path. To maximize throughput, all threads in a warp should follow the same control-flow. For example, the code in fig. 3 will be executed as showed in the table, where SM skips execution of a core subjected to the branch conditions. All GPU tasks will be placed in one stream and executed subsequently, meanwhile for the calling program on CPU they are executed asynchronously. There are functions like *cudaDeviceSynchronize()* that keep the CPU program waiting for the GPU execution. The same is for the warps executed in the same block, where while all the threads in the same warps execute the same instructions, different warps are executed independently. The function *__syncthreads()* waits until all threads in all warps within the same block have finished the execution.

```
1 if (a[index]==0)
2     a[index]++;
3 else
4     a[index]--;
```

	c0 (a=3)	c1(a=3)	c2 (a=-3)	c3(a=7)	c4(a=2)	c5(a=6)	c6 (a=-2)	c7 (a=-1)
if a[index]==0	↓	↓	↓	↓	↓	↓	↓	↓
a[index]++	↓	↓		↓	↓	↓		
a[index]–			↓				↓	↓

Figure 3: An example CUDA program with branch divergence.

2.3.3 Memory model

Every SM has a shared memory accessible by all threads in the same block (fig. 4). Each thread has its own local memory. All threads and blocks can access to a global memory and constant memory (read only). There is a difference in terms of performance (shared memory is faster compared to local and global memory) but mainly the type of memory to use depends from the scope (thread, block or grid level). To manage memory on CUDA [8][9], the first step is to allocate memory on GPU with the function *cudaMalloc()*, next with *cudaMemcpy()* the variables currently on the RAM are copied to the global memory of the GPU, in the previous allocated areas. In this way, when the kernel will be launched from CPU all the data will be stored in GPU global memory. Subsequently, all the data needed for a block from the global memory can be copied to the shared memory if necessary. Otherwise, all thread can access to stored variables in global memory.

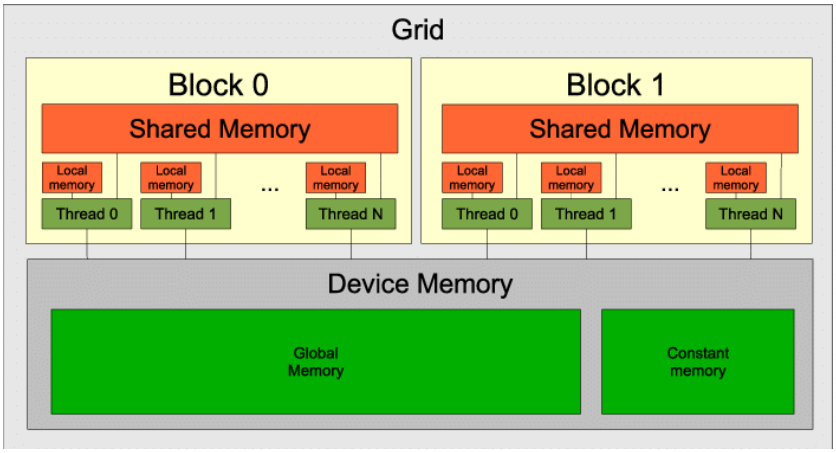


Figure 4: CUDA memory model.

3 Microsat to CUDA porting

For first, it is necessary to obtain a data set on which the SAT solver can be tested and verified. The dataset will consist of SAT and UNSAT formulas, generated using a CNF formula generator, CNFgen [6]. In particular, we used the *randkcnf* function which generates random formulas in DIMACS format (fig. 5) used as the standard format for boolean formulas in CNF. The lines starting with *c* are comments, while the line *"p cnf <variables> <clauses>"* describes the type of formula, the number of variables and the number of clauses. Each of the following lines indicates a clause with literals.

```
c A sample .cnf file.
p cnf 3 2
1 -3 0
2 3 -1 0
```

Figure 5: An example of DIMACS file format.

The first step to convert Microsat to CUDA was to understand how to represent the data structure, since in order to run code on the GPU you need all the data to be on it. Initially, we tried to allocate the data of the formulas directly on the GPU memory, but since we need to parse them from DIMACS

[6] https://github.com/MassimoLauria/cnfgen

files saved on filesystem that is not directly accessible by GPU, we have to do this first step using the CPU. Another limitation is that all GPU memory areas must be allocated by CPU using a specific command (*cudaMalloc()*) in order to be used. In this way, we are forced to allocate all the GPU memory we need before starting the solver, therefore it becomes very complex the dynamic space management for each formula. As a first approach we tried to load the DIMACS files in RAM and then copy them to the GPU, next parse the formulas in parallel threads. In this way, half of the space of the GPU is occupied by the files containing the formulas halving the memory available to us. Afterwards, to try to reuse the space wasted by the parsing of the formulas, the space occupied by the loaded DIMACS files is freed and the same process is performed again. This approach did not bring great results in terms of performances, thus we decided to try another one. The next approach (the one we used at the end) is to parse all the formulas sequentially using the CPU, then allocate exactly the space needed in the GPU and copy the whole data structure which represents the formulas. This approach increases the time needed for parsing, but is the method that produced the best results overall. As we found a way to get the data on the GPU, we moved on to the translation of the resolution methods used by Microsat. We can not use libraries written in CPU in CUDA kernels and we need to set specific tags to specify where each function (kernel) must be executed (__host__, __device__, __global__). This led us to review and rewrite some procedures to allow the Microsat functions to run entirely on GPU. Once we got a working version, the next step was to find a way to get the results, as said before it is not possible to access external resources from GPU. To avoid taking up more space on GPU, when a formula produces a result it is printed on *stdout* along with various statistics (vars_number, clauses_number, file_number, db_max_mem, clause_learn_max_mem, initial_max_lemmas, time, parse_time, init_time, solve_time, mem_used - Mbyte, conflicts_number). In this way when we execute Microsat on CUDA, all the output will be redirected to a log file. When all the formulas have been processed we obtain a log file with all results and statistics. At the end a parser retrieves all the data from the log file and creates a more readable CSV file with the execution results.

3.1 MicroSAT-CUDA v1

After the conversion process, we obtained the algorithm described in (fig. 6). MicroSAT-CUDA simply reads and parses the input formulas on CPU, then initializes on the GPU the data structures used by the solving process for each formula. Next, it copies everything on GPU and starts the solving process. Moreover, the algorithm takes as input parameters the maximum memory available for by the solver to store assignments and clauses (*db_max_mem*), the maximum memory that can be used for clause learning (*learn_max_mem*) and the initial number of clauses that can be learned (*max_lemmas*), which defines how fast the db of clauses grows. For each kernel call, follows the *cudaDeviceSyncronize()* function which makes the CPU wait for the GPU to finish.

```
main() {
    cudaDeviceReset()

    parameters = getParams()
    files = getFiles()

    Malloc( size(files) )

    for each file:
        formula = Parse(file)
        cudaMalloc(formula)

        kernelInit<< < 1, 1 > >>(gpu_solver, formula, parameters)
        cudaDeviceSynchronize()

    kernelSolve<< < Count(files), 1 > >>(gpu_solver)
    cudaDeviceSynchronize()

    printMetrics()
}
```

Figure 6: MicroSAT-CUDA v1.

3.2 MicroSAT-CUDA v2 (optimized)

We made a second version of Microsat CUDA, where there in an enhancement in terms of memory used. In the first version (fig. 6) for each file parsed, there is the allocation of memory needed by the solver data structure and the db used to store and assign variables and clause learning. Then, in the optimized version (fig. 7), to reduce the number of *malloc()* operations, we use as *db* a single memory area allocated before formula parsing, where based on a defined offset each solver can access to perform operations. The main reason behind this enhancement is to avoid GPU memory fragmentation, so we can make the most of the available memory.

```
main() {
    cudaDeviceReset()

    parameters = getParams()
    files = getFiles()

    struct = cudaMalloc( size(files) )

    for each file:
        formula = Parse(file)
        offset = cudaMemCpy(formula, struct)

        kernelInit<< < 1, 1 > >>(gpu_solver, offset, parameters)

        cpu_solvers.add(gpu_solver)

    cudaMalloc(gpu_solvers)
    cudaMemCpy(cpu_solvers, gpu_solvers)
    kernelSolve<< < Count(files), 1 > >>(gpu_solvers)
    cudaDeviceSynchronize()

    printMetrics()
}
```

Figure 7: MicroSAT-CUDA v2.

3.3 MicroSAT-CUDA v3 (multi-GPU)

Since MicroSAT-CUDA works on a single GPU, we modified the optimized version (fig. 7) to run on multiple GPUs (fig. 8) to to have more memory available and therefore solve more formulas. To achieve this, we distributed the formulas equally among the available GPUs, defined as command line parameter. Next, for each GPU, there is the parsing of formulas, the initialization of data structures and the solving process.

```
main() {
    cudaDeviceReset()

    parameters = getParams()
    gpu_number = countCpu()
    files = getFiles()

    memory_on_gpu = files / gpu_number
    for each GPU:
         struct = cudaMalloc( size(files) )

    cudaSetDevice() for first GPU
    for each file:
        formula = Parse(file)
        offset = cudaMemCpy(formula, struct)

        kernelInit<< < 1, 1 > >>(gpu_solver, offset, parameters)

        cpu_solvers.add(gpu_solver)

    for each gpu:
        cudaMalloc(gpu_solvers)
        cudaMemCpy(cpu_solvers, gpu_solvers)
        kernelSolve<< < memory_on_gpu, 1 > >>(gpu_solvers)
        cudaDeviceSynchronize()

    printMetrics()
}
```

Figure 8: MicroSAT CUDA v3.

4 Experimentation and results

To perform the experimentation, we generated six different datasets (fig. 1) composed by satisfiable formulas having variables and clauses count of increasing size. For each formula size, we created 8 different instances with different sizes in terms of contained formulas, namely 1000, 2000, 4000, 8000, 16000, 32000, 64000 and 128000 formulas which is the maximum that our GPU could handle.

Name	Variables	Clauses
20_91	20	91
50_218	50	218
75_325	75	325
100_430	100	430
125_538	125	538
150_645	150	645

Table 1: Size of formulas used for experimentation.

We used two different environments to run MicroSAT-CUDA: the first is a desktop PC having a single NVIDIA GTX 1060 with 6GB of memory, the latter is the IRIDIS 5 cluster from University of Southampton[7] where we used a single node having four NVIDIA GTX 1080 Ti with 11GB of memory for a total of 44GB. The cluster also have nodes with Tesla V100 GPUs with more available memory, but after several tests the GTX 1080 Ti GPUs were more performing. Since MicroSAT-CUDA requires the setting of the parameters described in fig.2, we used a trial and error approach to find the ones that worked best for the smallest formula size (20 variables, 91 clauses), then we set a proportional value for the other formula sizes (fig.3). In some cases we had to manually tune the parameter due to execution failures. The value of input parameters directly affects the memory used and the running time.

Parameter name	Description
db_max_mem	Defines the dimension of db used by the solver mainly for storage of clauses and variable assignments.
learn_max_mem	Defines the maximum memory that can be used for clause learning.
max_lemmas	Defines the initial threshold for lemmas. It directly affects the growth of the clause database.

Table 2: Input parameters for MicroSAT-CUDA.

For example if *learn_max_mem* is too high there will be excessive memory usage, but if the value is too low the execution times of the solver will be

[7]https://www.southampton.ac.uk/isolutions/staff/iridis.page

longer. For example if *learn_max_mem* is too high there will be excessive memory usage, but if the value is too low the execution times of the solver will be longer.

Variables	Clauses	db_max_mem	learn_max_mem	max_lemmas
20	91	1000	1000	2000
50	218	3500	1000	2000
75	325	20000	1000	2000
100	430	35000	1000	2000
125	538	120000	1000	4000
150	645	412000	1000	4000

Table 3: Input parameters used in MicroSAT-CUDA for each formula size.

4.1 MicroSAT-CUDA on desktop PC

We executed and compared the performance between MiniSAT, MicroSAT, MicroSAT-CUDAv1 and MicroSAT-CUDAv2. In terms of execution time (fig. A.17), MicroSAT-CUDAv1 and MicroSAT-CUDAv2 perform better than MicroSAT and MiniSAT for small formulas (figs. A.18 and A.19) for up to 128000 formulas simultaneously, but for bigger formulas (figs. A.20, A.21, A.22 and A.23) they perform worse compared to MicroSAT and MiniSAT or simply the solver fails to run (ex. fig.A.23 the fields with zero value). Comparing the execution time of kernel functions (i.e. *init* and *solve*) between MicroSAT-CUDAv1 and MicroSAT-CUDAv2 (figs. A.24, A.25, A.26, A.27, A.28, A.29 and A.30), there are no improvements in MicroSAT-CUDAv2 which in some cases fails to execute. Comparing the memory used (figs. A.31, A.32, A.33, A.34, A.35, A.36 and A.36), MicroSAT uses less memory compared to MicroSAT-CUDAv1 and also for small instances performs better than MiniSAT. MicroSAT-CUDAv2 is always slightly better than MicroSAT-CUDAv1. In terms of number of conflicts (figs. A.10, A.11, A.12, A.13, A.14, A.15 and A.16), MicroSAT-CUDAv1 and MicroSAT-CUDAv2 have a lower number of conflicts compared to MicroSAT and MiniSAT with small instances (figs. A.11 and A.12). The difference between MicroSAT and MicroSAT-CUDA is given by the fact that with the first we used the default values, with the latter we used custom values based on a trial and error approach. The number of conflicts between MicroSAT-CUDAv1 and MicroSAT-CUDAv2 is nearly the same in all cases. As we could not explain the failures of MicroSAT-CUDAv2 on larger instances, we carefully double-checked the code and all the scripts used for the experiments. We found out

that they are due to an error on the parameters used for the launch scripts for instances *125_538* and *150_645*.

4.2 MicroSAT-CUDA on IRIDIS 5

IRIDIS 5 is a cluster for High Performance Computing of University of Southampton. It is designed as a batch service for multiple users which uses the SLURM batch job manager[8]. To execute our experimentation, we made some SLURM configuration files (fig. 9) where we set the needed resource (for example, the type of node to use - line 9) and the code to execute (lines 21-27).

```
1  #!/bin/bash
2
3  ####################################
4  #  Iridis 5 slurm script template #
5  #                                  #
6  #  Submit script: sbatch filename #
7  #                                  #
8  ####################################
9  #SBATCH -p gtx1080 --gres=gpu:1
10
11 #SBATCH --job-name=mcuda_opt # Job name
12 #SBATCH --output=/home/gp1y10/cuda_sat/microsat-cuda/cuda_jobs/gtx1080/mcuda_opt.log #
       Stdout (%j expands to jobId)
13 #SBATCH --error=/home/gp1y10/cuda_sat/microsat-cuda/cuda_jobs/gtx1080/mcuda_opt.err #
       Stderr (%j expands to jobId)
14 #SBATCH --ntasks-per-node=1  # Tasks per node
15 #SBATCH --ntasks=1   # Number of processor cores (i.e. tasks)
16 #SBATCH --nodes=1   # Number of nodes requested
17 #SBATCH --cpus-per-task=1  # Threads per task
18 #SBATCH --time=1440:00 # walltime
19 #SBATCH --begin=now
20
21 # load necessary modules
22 module load cuda
23
24 ## Running serial applications##
25 nvcc -o /scratch/gp1y10/dimacs/mcuda_opt
       /home/gp1y10/cuda_sat/microsat-cuda/microsat_cuda_malloc_opt.cu
26 cd /scratch/gp1y10/dimacs/
27 sh cuda_opt_job_cluster.sh
```

Figure 9: Example of SLURM configuration file for IRIDIS 5.

[8]https://slurm.schedmd.com/quickstart.html

The SLURM configuration file is in bash format and is executed using the command *sbatch slurm_file.slurm* . Due to disk quota limits, we had some problems with the generation of the instances. For each execution, we had to delete the previous instances and generate new ones using the 1000 formulas instances as starting point. We executed and compared the performance between MicroSAT-CUDAv2 and MicroSAT-CUDAv3 which can use the multiple GPUs available on the IRIDIS cluster. In terms of execution time (fig. B.45), MicroSAT-CUDAv3 is faster compared to MicroSAT-CUDAv2 in most instances (figs. B.46, B.47, B.48 and B.49) except in the larger formulas (figs. B.50 and B.51). On the other hand, MicroSAT-CUDAv3 it does not fail on larger instances because it has more memory available. Moreover, comparing in detail the execution time of *init*, *parse* and *solve* functions (figs. B.52, B.53, B.54, B.55, B.56 and B.57) MicroSAT-CUDAv3 is always faster in the *solve* function, while *init* and *parse* time is always equal or lower to MicroSAT-CUDAv2. In terms of memory used (figs. A.31, A.32, A.33, A.34, A.35 ,A.36 and A.37), MicroSAT-CUDAv3 uses slightly more memory than MicroSAT-CUDAv2 because all variables and data structures are allocated to each GPU. In terms of conflicts (figs. B.38, B.39, B.40, B.41, B.42, B.43 and B.44) in both cases are almost the same number.

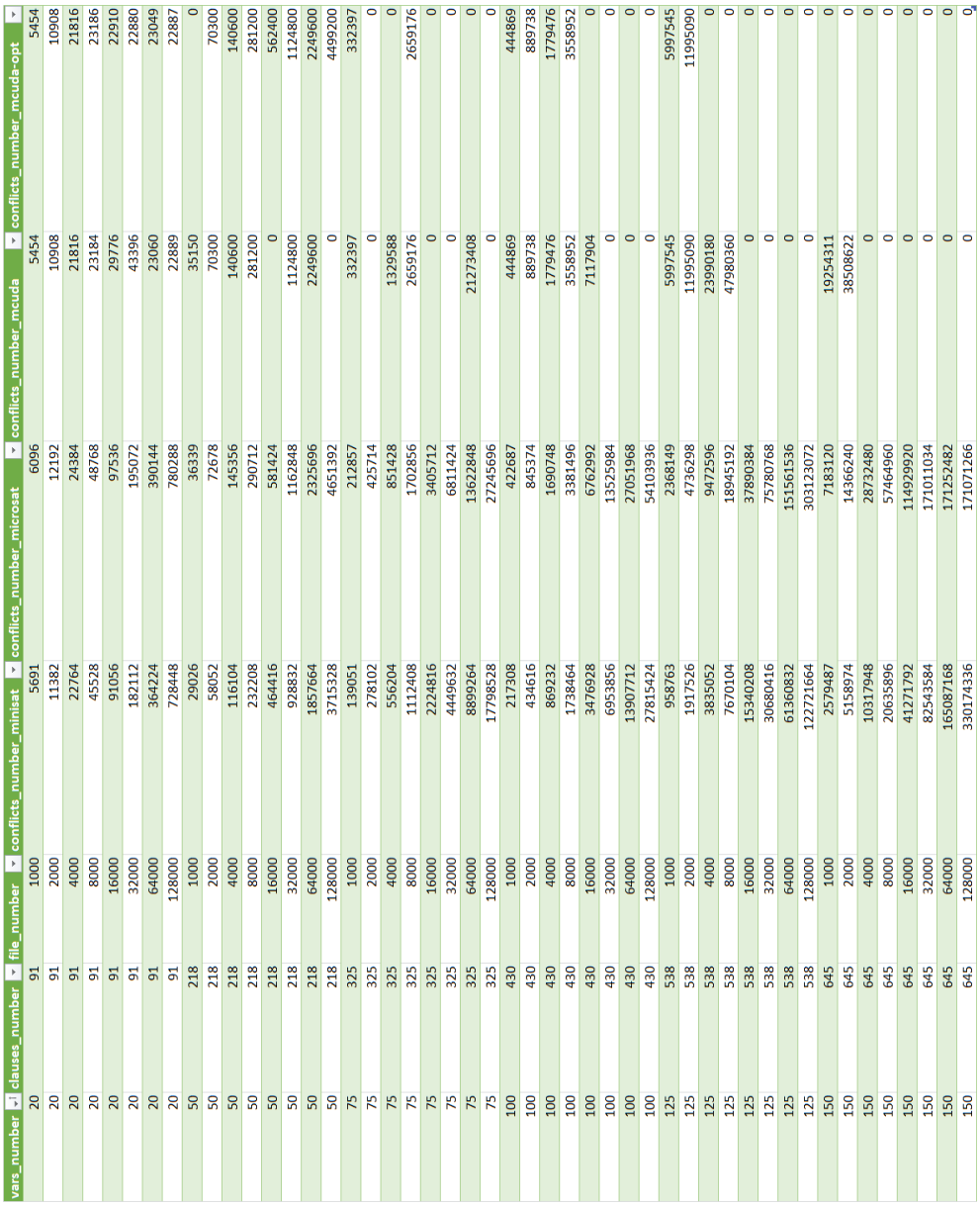
5 Conclusion

In this work we did an experimentation that consists in porting MicroSAT on GPU using CUDA and the parallel resolution of different instances of boolean formulas. Mapping SAT to a GPU platform is not a simple task. SAT problems require much less data and much more computation with respect to the traditional image and matrix processing for which GPUs are commonly employed. Moreover, the porting of a SAT solver designed to run on CPU can be a problem due to required libraries which are not present on CUDA. The biggest challenge to face for running a SAT solver on GPUs is the memory management, which is complex due to the fact that a GPU can not directly interact with the filesystem and has a much more complex memory model compared to traditional CPU programming. Anyhow, we have created three different versions of our porting of MicroSAT, namely MicroSAT-CUDAv1 which is the first version, MicroSAT-CUDAv2 which has some improvements in terms of memory management and MicroSAT-CUDAv3 which can run on multiple GPUs. We generated a dataset of six different formulas sizes 1 where each one has different instances, composed by 1000, 2000, 4000,

8000, 16000, 32000, 64000 and 128000 formulas. We used two different environments for our experimentation, where we have a desktop PC with a consumer GPU and IRIDIS 5, an HPC cluster from University of Southampton which can offer up to four high-end GPUs. In the first case, we compared MiniSAT, MicroSAT, MicroSAT-CUDAv1 and MicroSAT-CUDAv2 where MicroSAT-CUDA performs better on small instances, where on larger formulas MiniSAT and MicroSAT are performing better. In the second case, we compared MicroSAT-CUDAv2 with MicroSAT-CUDAv3 on the cluster using a single node with four GTX 1080Ti, where thanks to the higher amount of memory, MicroSAT-CUDAv3 could run also the larger instances. What we can get from our experimentation is that GPUs are powerful for solving parallel tasks, but in the case of a SAT solver, the best choice is not to run entirely the solver on GPU but to use the CPU for more complex tasks and use GPU for many simple tasks simultaneously like unit propagation, clause simplification and other techniques like two-watched literals.

A Desktop

A.1 Conflicts number comparison between MiniSAT, MicroSAT, mcuda (MicroSAT-CUDAv1) and mcuda-opt (MicroSAT-CUDAv2)



vars_number	clauses_number	file_number	conflicts_number_minisat	conflicts_number_microsat	conflicts_number_mcuda	conflicts_number_mcuda-opt
20	91	1000	5691	6096	5454	5454
20	91	2000	11382	12192	10908	10908
20	91	4000	22764	24384	21816	21816
20	91	8000	45528	48768	23184	23186
20	91	16000	91056	97536	29776	22910
20	91	32000	182112	195072	43396	22880
20	91	64000	364224	390144	23060	23049
20	91	128000	728448	780288	22889	22887
50	218	1000	29026	36339	35150	0
50	218	2000	58052	72678	70300	70300
50	218	4000	116104	145356	140600	140600
50	218	8000	232208	290712	281200	281200
50	218	16000	464416	581424	0	562400
50	218	32000	928832	1162848	1124800	1124800
50	218	64000	1857664	2325696	2249600	2249600
50	218	128000	3715328	4651392	0	4499200
75	325	1000	139051	212857	332397	332397
75	325	2000	278102	425714	0	0
75	325	4000	556204	851428	1329588	0
75	325	8000	1112408	1702856	2659176	2659176
75	325	16000	2224816	3405712	0	0
75	325	32000	4449632	6811424	0	0
75	325	64000	8899264	13622848	21273408	0
75	325	128000	17798528	27245696	0	0
100	430	1000	217308	422687	444869	444869
100	430	2000	434616	845374	889738	889738
100	430	4000	869232	1690748	1779476	1779476
100	430	8000	1738464	3381496	3558952	3558952
100	430	16000	3476928	6762992	7117904	0
100	430	32000	6953856	13525984	0	0
100	430	64000	13907712	27051968	0	0
100	430	128000	27815424	54103936	0	0
125	538	1000	958763	2368149	5997545	5997545
125	538	2000	1917526	4736298	11995090	11995090
125	538	4000	3835052	9472596	23990180	0
125	538	8000	7670104	18945192	47980360	0
125	538	16000	15340208	37890384	0	0
125	538	32000	30680416	75780768	0	0
125	538	64000	61360832	151561536	0	0
125	538	128000	122721664	303123072	0	0
150	645	1000	2579487	7183120	19254311	0
150	645	2000	5158974	14366240	38508622	0
150	645	4000	10317948	28732480	0	0
150	645	8000	20635896	57464960	0	0
150	645	16000	41271792	114929920	0	0
150	645	32000	82543584	171011034	0	0
150	645	64000	165087168	171252482	0	0
150	645	128000	330174336	171071266	0	0

Figure A.10

Figure A.11: 20 clauses - 91 literals

file number	conflitti minisat	conflitti microsat	confklitti mcuda	conflitti mcuda opt
1000	5691	6096	5454	5454
2000	11382	12192	10908	10908
4000	22764	24384	21816	21816
8000	45528	48768	23184	23186
16000	91056	97536	29776	22910
32000	182112	195072	43396	22880
64000	364224	390144	23060	23049
128000	728448	780288	22889	22887
Totale complessivo	**1451205**	**1554480**	**180483**	**153090**

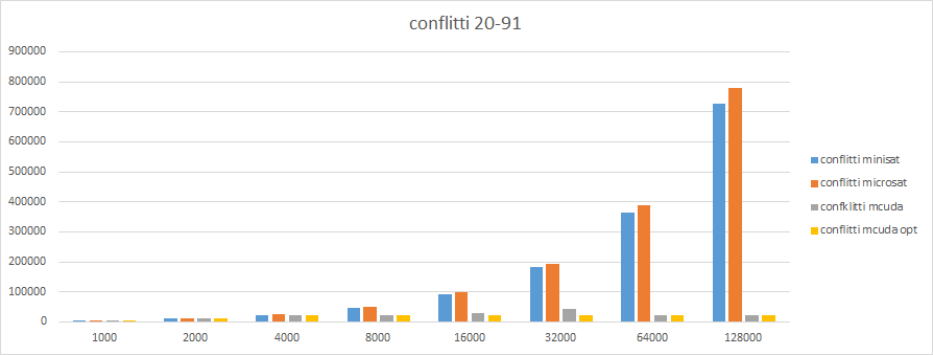


Figure A.12: 50 clauses - 218 literals

file number	conflitti minisat	conflitti microsat	confklitti mcuda	conflitti mcuda opt
1000	29026	36339	35150	0
2000	58052	72678	70300	70300
4000	116104	145356	140600	140600
8000	232208	290712	281200	281200
16000	464416	581424	0	562400
32000	928832	1162848	1124800	1124800
64000	1857664	2325696	2249600	2249600
128000	3715328	4651392	0	4499200
Totale complessivo	**7401630**	**9266445**	**3901650**	**8928100**

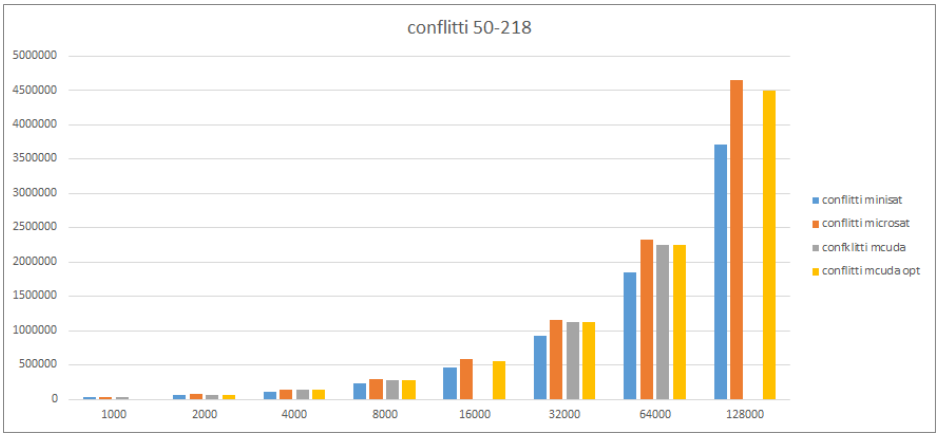


Figure A.13: 75 clauses - 325 literals

file number	conflitti minisat	conflitti microsat	confklitti mcuda	conflitti mcuda opt
1000	139051	212857	332397	332397
2000	278102	425714	0	0
4000	556204	851428	1329588	0
8000	1112408	1702856	2659176	2659176
16000	2224816	3405712	0	0
32000	4449632	6811424	0	0
64000	8899264	13622848	21273408	0
128000	17798528	27245696	0	0
Totale complessivo	**35458005**	**54278535**	**25594569**	**2991573**

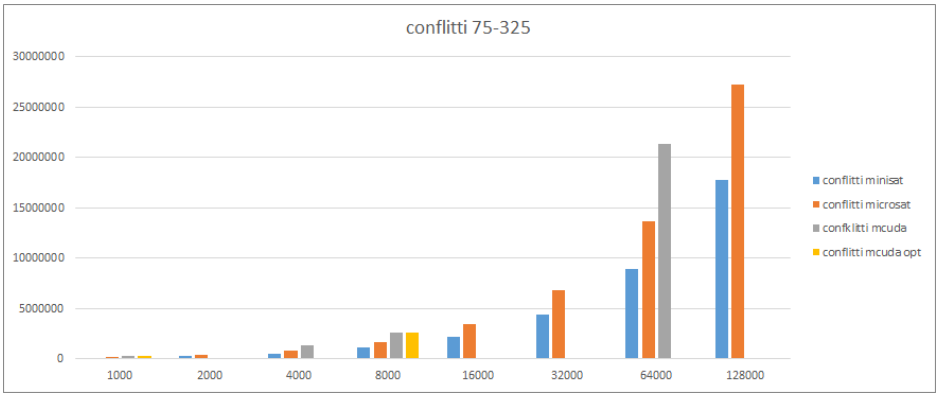


Figure A.14: 100 clauses - 430 literals

file number	conflitti minisat	conflitti microsat	confklitti mcuda	conflitti mcuda opt
1000	217308	422687	444869	444869
2000	434616	845374	889738	889738
4000	869232	1690748	1779476	1779476
8000	1738464	3381496	3558952	3558952
16000	3476928	6762992	7117904	0
32000	6953856	13525984	0	0
64000	13907712	27051968	0	0
128000	27815424	54103936	0	0
Totale complessivo	**55413540**	**107785185**	**13790939**	**6673035**

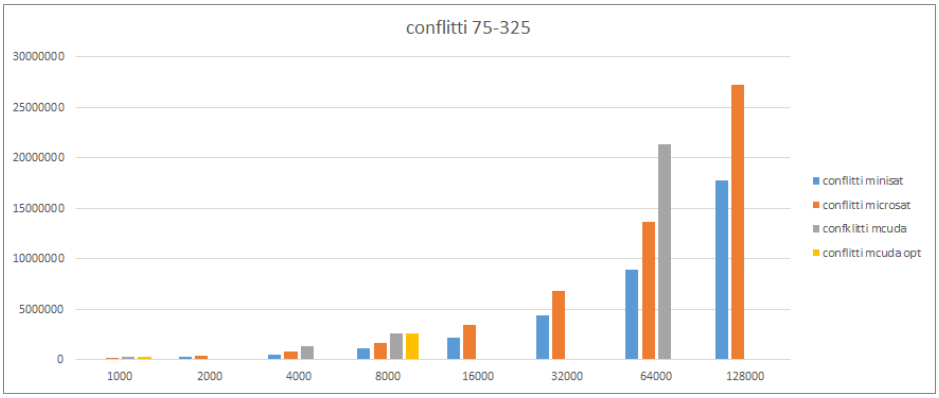


Figure A.15: 125 clauses - 538 literals

file number	conflitti minisat	conflitti microsat	confklitti mcuda	conflitti mcuda opt
1000	958763	2368149	5997545	5997545
2000	1917526	4736298	11995090	11995090
4000	3835052	9472596	23990180	0
8000	7670104	18945192	47980360	0
16000	15340208	37890384	0	0
32000	30680416	75780768	0	0
64000	61360832	151561536	0	0
128000	122721664	303123072	0	0
Totale complessivo	**244484565**	**603877995**	**89963175**	**17992635**

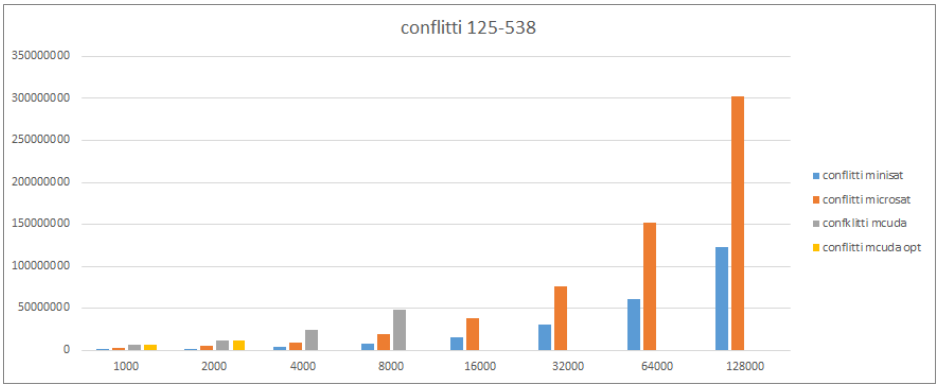
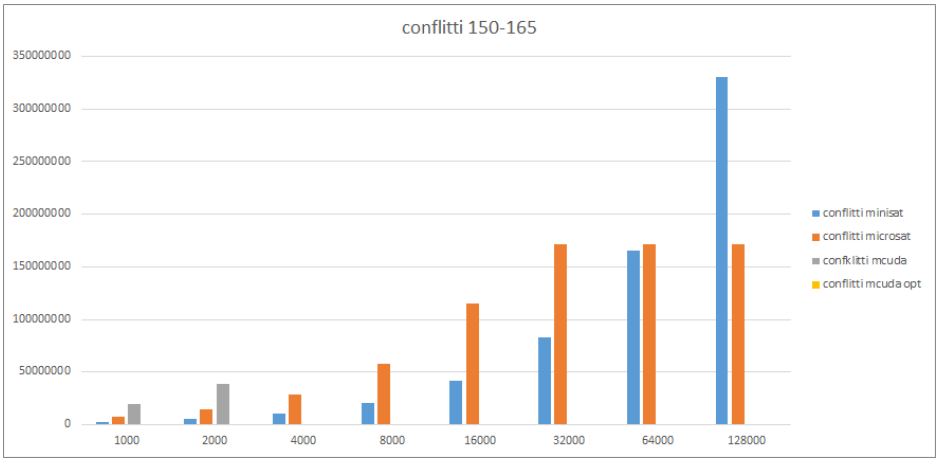


Figure A.16: 150 clauses - 645 literals

file number	conflitti minisat	conflitti microsat	confklitti mcuda	conflitti mcuda opt
1000	2579487	7183120	19254311	0
2000	5158974	14366240	38508622	0
4000	10317948	28732480	0	0
8000	20635896	57464960	0	0
16000	41271792	114929920	0	0
32000	82543584	171011034	0	0
64000	165087168	171252482	0	0
128000	330174336	171071266	0	0
Totale complessivo	**657769185**	**736011502**	**57762933**	**0**



A.2 Execution time comparison between MiniSAT, MicroSAT, mcuda (MicroSAT-CUDAv1) and mcuda-opt (MicroSAT-CUDAv2)

vars_number	clauses_number	file_number	minisat_tot_time	microsat_tot_time	mcuda_tot_time	mcuda_parse_time	mcuda_init_time	mcuda-opt_tot_time	mcuda-opt_parse_time	mcuda-opt_init_time
20	91	1000	1251	470	661	0,352294	0,271041	705	0,338607	0,226894
20	91	2000	2455	1243	1172	0,654349	0,452601	1239	0,663716	0,452266
20	91	4000	7355	2474	4371	1,446195	0,909737	1997	1,276033	0,903762
20	91	8000	11882	6680	4039	2,649591	1,809472	4151	2,647667	1,808137
20	91	16000	24770	11795	15464	5,854683	3,671909	15911	5,652341	3,677067
20	91	32000	50538	26373	22467	11,069157	7,249842	24894	10,960062	7,243697
20	91	64000	109166	55239	39666	22,239402	14,558036	39026	21,739653	14,558675
20	91	128000	235068	92237	88777	45,530241	29,092746	85661	43,884174	29,011738
50	218	1000	1318	1160	1175	0,685491	0,534895	0	0	0
50	218	2000	2626	1254	4892	1,516469	1,074272	1986	1,366017	1,069192
50	218	4000	7505	2507	3799	2,808707	2,141006	3633	2,726318	2,138849
50	218	8000	13051	5459	7212	5,566865	4,279609	7245	5,631199	4,278982
50	218	16000	26324	10773	0	0	0	22940	11,442196	8,567896
50	218	32000	54987	21508	39799	23,05754	17,246308	36418	22,737194	17,184673
50	218	64000	109313	49859	81252	46,700535	34,347283	79521	45,949758	34,314732
50	218	128000	233995	149291	0	0	0	157750	91,641967	68,703238
75	325	1000	1668	1543	2885	0,991874	0,795139	2931	1,038937	0,796031
75	325	2000	3337	3056	0	0	0	0	0	0
75	325	4000	8352	7672	14376	4,792898	3,264542	0	0	0
75	325	8000	15310	15433	22369	8,688811	6,514273	21933	8,478853	6,372871
75	325	16000	33001	29536	0	0	0	0	0	0
75	325	32000	66192	59867	0	0	0	0	0	0
75	325	64000	138246	125749	145602	67,387166	51,135237	0	0	0
75	325	128000	297053	271615	0	0	0	0	0	0
100	430	1000	2054	2896	5382	1,309537	1,051955	5419	1,294645	1,050044
100	430	2000	5996	5859	8157	2,588852	2,103403	13035	2,990311	2,231595
100	430	4000	9621	11586	17321	5,427034	4,295798	13048	5,135809	4,203807
100	430	8000	19269	23257	23925	10,440703	8,415079	23710	10,315528	8,405765
100	430	16000	39234	46569	53402	21,582934	16,931107	0	0	0
100	430	32000	79879	94712	0	0	0	0	0	0
100	430	64000	165561	193514	0	0	0	0	0	0
100	430	128000	356449	405144	0	0	0	0	0	0
125	538	1000	2678	2562	9504	1,619478	1,312536	9711	1,791014	1,318328
125	538	2000	11074	5246	14816	3,274156	2,632205	156372	3,203124	2,629467
125	538	4000	24992	10518	25745	7,384884	5,275355	0	0	0
125	538	8000	46486	20976	45402	13,563105	10,526588	0	0	0
125	538	16000	99577	42084	0	0	0	0	0	0
125	538	32000	192810	842593	0	0	0	0	0	0
125	538	64000	387644	1692766	0	0	0	0	0	0
125	538	128000	802912	3414459	0	0	0	0	0	0
150	645	1000	13850	151195	317615	2,004369	1,57627	0	0	0
150	645	2000	28230	301862	561513	4,058534	3,151605	0	0	0
150	645	4000	56827	606120	0	0	0	0	0	0
150	645	8000	112662	1209551	0	0	0	0	0	0
150	645	16000	226707	2421948	0	0	0	0	0	0
150	645	32000	454722	3600002	0	0	0	0	0	0
150	645	64000	915690	3600003	0	0	0	0	0	0
150	645	128000	1866429	3600002	0	0	0	0	0	0

Figure A.17

Figure A.18: 20 clauses - 91 literals

file number	time minisat	time microsat	time mcuda	time mcuda-opt
1000	1251	470	661	705
2000	2455	1243	1172	1239
4000	7355	2474	4371	1997
8000	11882	6680	4039	4151
16000	24770	11795	15464	15911
32000	50538	26373	22467	24894
64000	109166	55239	39666	39026
128000	235068	92237	88777	85661
Totale complessivo	**442485**	**196511**	**176617**	**173584**

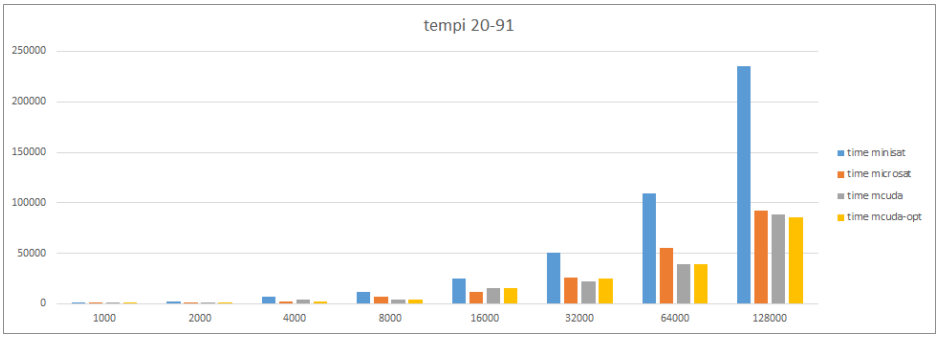


Figure A.19: 50 clauses - 218 literals

file number	time minisat	time microsat	time mcuda	time mcuda-opt
1000	1318	1160	1175	0
2000	2626	1254	4892	1986
4000	7505	2507	3799	3633
8000	13051	5459	7212	7245
16000	26324	10773	0	22940
32000	54987	21508	39799	36418
64000	109313	49859	81252	79521
128000	233995	149291	0	157750
Totale complessivo	**449119**	**241811**	**138129**	**309493**

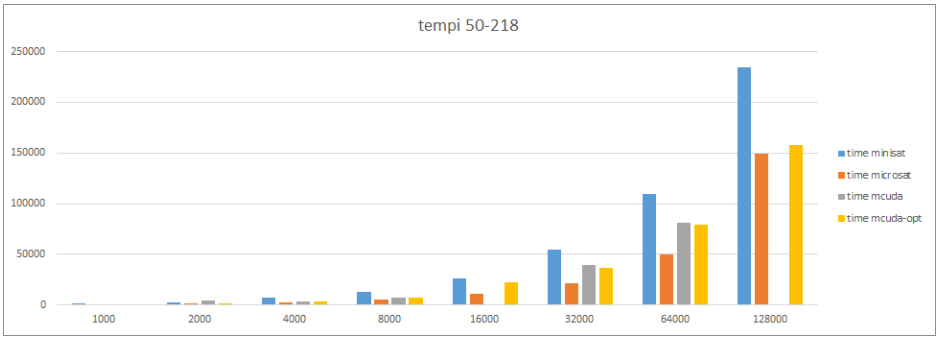


Figure A.20: 75 clauses - 325 literals

file number	time minisat	time microsat	time mcuda	time mcuda-opt
1000	1668	1543	2885	2931
2000	3337	3056	0	0
4000	8352	7672	14376	0
8000	15310	15433	22369	21933
16000	33001	29536	0	0
32000	66192	59867	0	0
64000	138246	125749	145602	0
128000	297053	271615	0	0
Totale complessivo	**563159**	**514471**	**185232**	**24864**

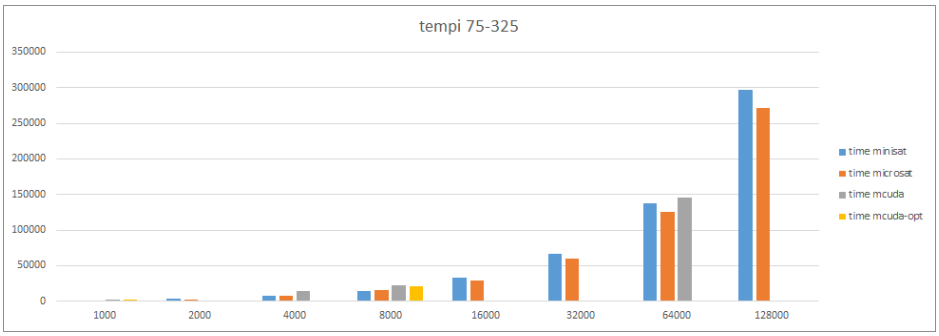


Figure A.21: 100 clauses - 430 literals

file number	time minisat	time microsat	time mcuda	time mcuda-opt
1000	2054	2896	5382	5419
2000	5996	5859	8157	13035
4000	9621	11586	17321	13048
8000	19269	23257	23925	23710
16000	39234	46509	53402	0
32000	79879	94712	0	0
64000	165561	193514	0	0
128000	356449	403144	0	0
Totale complessivo	**678063**	**781477**	**108187**	**55212**

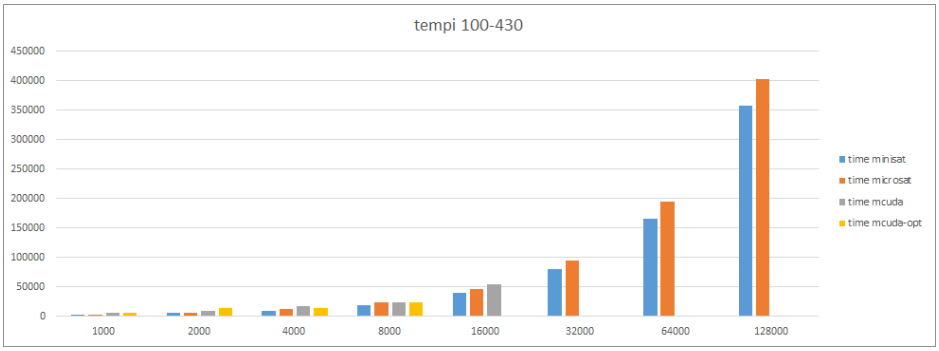


Figure A.22: 125 clauses - 538 literals

file number	time minisat	time microsat	time mcuda	time mcuda-opt
1000	4978	26262	95040	97111
2000	11074	52346	148165	150372
4000	24992	105128	257425	0
8000	46486	209763	454029	0
16000	93677	420841	0	0
32000	192810	842593	0	0
64000	387644	1692766	0	0
128000	802912	3414459	0	0
Totale complessivo	**1564573**	**6764158**	**954659**	**247483**

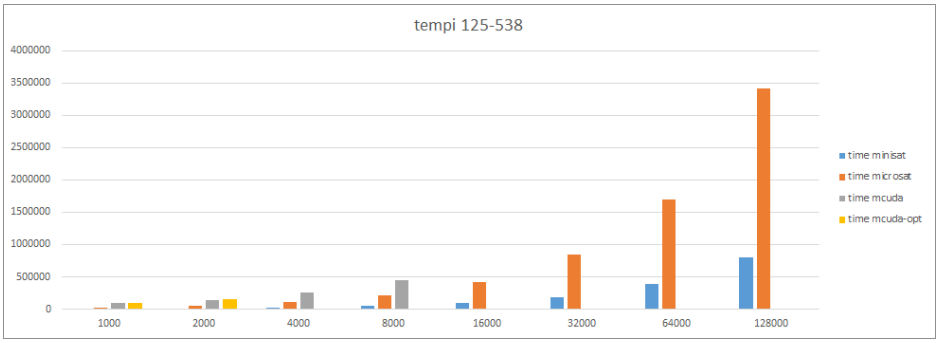
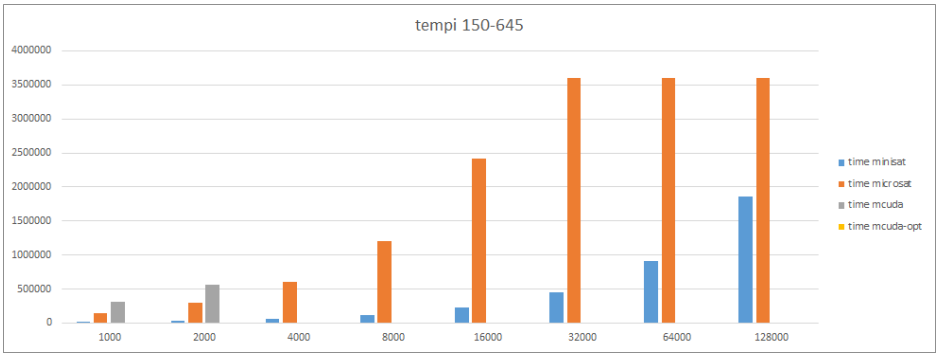


Figure A.23: 150 clauses - 645 literals

file number	time minisat	time microsat	time mcuda	time mcuda-opt
1000	13850	151195	317615	0
2000	28230	301862	561513	0
4000	56827	606120	0	0
8000	112662	1209551	0	0
16000	226707	2421948	0	0
32000	454722	3600002	0	0
64000	915690	3600003	0	0
128000	1866429	3600002	0	0
Totale complessivo	**3675117**	**15490683**	**879128**	**0**



A.3 Time comparison between mcuda (MicroSAT-CUDAv1) and mcuda-opt (MicroSAT-CUDAv2)

vars_number	clauses_number	file_number	mcuda_tot_time	mcuda_parse_time	mcuda_init_time	mcuda_solve_time	mcuda_opt_tot_time	mcuda-opt_parse_time	mcuda-opt_init_time	mcuda-opt_solve_time
20	91	1000	661	0,352294	0,271041	0,005248	705	0,338607	0,226894	0,004832
20	91	2000	1172	0,654349	0,452601	0,008237	1239	0,663716	0,452266	0,008197
20	91	4000	4371	1,446195	0,909737	0,014957	1997	1,276033	0,903762	0,014992
20	91	8000	4039	2,649591	1,809472	0,028654	4151	2,647667	1,808137	0,028723
20	91	16000	15464	5,854683	3,671909	0,055463	15911	5,652341	3,677067	0,055329
20	91	32000	22467	11,069157	7,249842	0,109258	24894	10,960062	7,243697	0,109244
20	91	64000	39666	22,239402	14,558036	0,217185	39026	21,739653	14,558675	0,217051
20	91	128000	88777	45,530241	29,092746	0,433377	85661	43,884174	29,011738	0,431698
50	218	1000	1175	0,685491	0,534895	0,075959	0	0	0	0
50	218	2000	4892	1,516469	1,074272	0,110425	1986	1,366017	1,069192	0,111591
50	218	4000	3799	2,808707	2,141006	0,182492	3633	2,726318	2,138849	0,185451
50	218	8000	7212	5,566865	4,279609	0,329087	7245	5,631199	4,278982	0,347327
50	218	16000	0	0	0	0	22940	11,442196	8,567896	0,629064
50	218	32000	39799	23,05754	17,246308	1,215323	36418	22,737194	17,184673	1,219129
50	218	64000	81252	46,700535	34,347283	2,39324	79521	45,949758	34,314732	2,468226
50	218	128000	0	0	0	0	157750	91,641967	68,703238	4,775583
75	325	1000	2885	0,991874	0,795139	1,479003	2931	1,038937	0,796631	1,499962
75	325	2000	0	0	0	0	0	0	0	0
75	325	4000	14376	4,792898	3,264542	2,992761	0	0	0	0
75	325	8000	22369	8,686811	6,514273	5,220209	21933	8,478853	6,372871	5,355791
75	325	16000	0	0	0	0	0	0	0	0
75	325	32000	0	0	0	0	0	0	0	0
75	325	64000	145602	67,387166	51,135237	39,713508	0	0	0	0
75	325	128000	0	0	0	0	0	0	0	0
100	430	1000	5382	1,309537	1,051955	3,71679	5419	1,294645	1,050044	3,788062
100	430	2000	5382	1,309537	1,051955	3,71679	7330	1,561357	1,094036	3,430012
100	430	4000	14376	4,792898	3,264542	2,992761	5419	1,294645	1,050044	3,788062
100	430	8000	22369	8,686811	6,514273	5,220209	21933	8,478853	6,372871	5,355791
100	430	16000	0	0	0	0	0	0	0	0
100	430	32000	0	0	0	0	0	0	0	0
100	430	64000	145602	67,387166	51,135237	39,713508	0	0	0	0
100	430	128000	0	0	0	0	0	0	0	0
100	430	1000	5382	1,309537	1,051955	3,71679	7330	1,561357	1,094036	3,430012
100	430	2000	5404	0	0	0	7330	1,561357	1,094036	3,430012
100	430	4000	8157	2,588852	2,103403	4,991945	13035	2,990311	2,231595	4,979914
100	430	8000	8157	2,588852	2,103403	4,991945	7719	2,57815	2,102814	4,5678
100	430	16000	6757	0	0	0	13035	2,990311	2,231595	4,979914
100	430	32000	6757	0	0	0	7719	2,57815	2,102814	4,5678
100	430	64000	17321	5,427034	4,295798	7,21049	13048	5,135809	4,203807	7,104288
100	430	128000	17321	5,427034	4,295798	7,21049	11721	5,175608	4,208282	6,681668
100	430	4000	15376	0	0	0	13048	5,135809	4,203807	7,104288
100	430	4000	15376	0	0	0	11721	5,175608	4,208282	6,681668
100	430	8000	23925	10,440703	8,419079	12,067734	23710	10,313528	8,405765	11,988385
100	430	8000	23925	10,440703	8,419079	12,067734	29672	10,906697	8,507031	12,017287
100	430	8000	20552	0	0	0	23710	10,313528	8,405765	11,988385
100	430	8000	20552	0	0	0	29672	10,906697	8,507031	12,017287
100	430	16000	53402	21,582934	16,931107	22,499521	0	0	0	0
100	430	16000	53402	21,582934	16,931107	22,499521	0	0	0	0
100	430	16000	35469	0	0	0	0	0	0	0
100	430	32000	0	0	0	0	0	0	0	0
100	430	32000	0	0	0	0	0	0	0	0
100	430	32000	0	0	0	0	0	0	0	0
100	430	64000	0	0	0	0	0	0	0	0
100	430	64000	0	0	0	0	0	0	0	0
100	430	64000	0	0	0	0	0	0	0	0
100	430	128000	0	0	0	0	0	0	0	0
100	430	128000	0	0	0	0	0	0	0	0
100	430	128000	0	0	0	0	0	0	0	0
125	538	1000	95040	1,619478	1,312536	93,055188	97111	1,791014	1,318328	92,679797
125	538	2000	148165	3,274156	2,63205	144,278656	150372	3,203124	2,629467	146,566188
125	538	4000	257425	7,384884	5,275355	240,812703	0	0	0	0
125	538	8000	454029	13,563105	10,526588	435,432	0	0	0	0
125	538	16000	0	0	0	0	0	0	0	0
125	538	32000	0	0	0	0	0	0	0	0
125	538	64000	0	0	0	0	0	0	0	0
125	538	128000	0	0	0	0	0	0	0	0
150	645	1000	317615	2,004369	1,57627	315,142781	0	0	0	0
150	645	2000	561513	4,058534	3,151605	556,687125	0	0	0	0
150	645	4000	0	0	0	0	0	0	0	0
150	645	8000	0	0	0	0	0	0	0	0
150	645	16000	0	0	0	0	0	0	0	0
150	645	32000	0	0	0	0	0	0	0	0
150	645	64000	0	0	0	0	0	0	0	0
150	645	128000	0	0	0	0	0	0	0	0

Figure A.24

Figure A.25: 20 clauses - 91 literals

file number	parse time cuda	parse time cuda opt	init time cuda	init time cuda opt	solve time cuda	solve time cuda opt
1000	0,352294	0,338607	0,271041	0,226894	0,005248	0,004832
2000	0,654349	0,663716	0,452601	0,452266	0,008237	0,008197
4000	1,446195	1,276033	0,909737	0,903762	0,014957	0,014992
8000	2,649591	2,647667	1,809472	1,808137	0,028654	0,028723
16000	5,854683	5,652341	3,671909	3,677067	0,055463	0,055329
32000	11,069157	10,960062	7,249842	7,243697	0,109258	0,109244
64000	22,239402	21,739653	14,558036	14,558675	0,217185	0,217051
128000	45,530241	43,884174	29,092746	29,011738	0,433377	0,431698
Totale complessivo	**89,795912**	**87,162253**	**58,015384**	**57,882236**	**0,872379**	**0,870066**

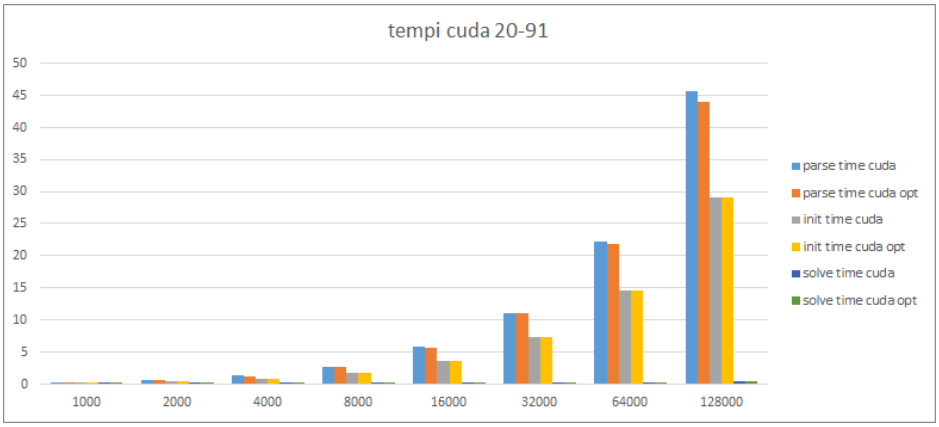


Figure A.26: 50 clauses - 218 literals

file number	parse time cuda	parse time cuda opt	init time cuda	init time cuda opt	solve time cuda	solve time cuda opt
1000	0,685491	0	0,534895	0	0,075959	0
2000	1,516469	1,366017	1,074272	1,069192	0,110425	0,111591
4000	2,808707	2,726318	2,141006	2,138849	0,182492	0,185451
8000	5,566865	5,631199	4,279609	4,278982	0,329087	0,347327
16000	0	11,442196	0	8,567896	0	0,629064
32000	23,05754	22,737194	17,246308	17,184673	1,215323	1,219129
64000	46,700535	45,949758	34,347283	34,314732	2,39324	2,468226
128000	0	91,641967	0	68,703238	0	4,775583
Totale complessivo	**80,335607**	**181,494649**	**59,623373**	**136,257562**	**4,306526**	**9,736371**

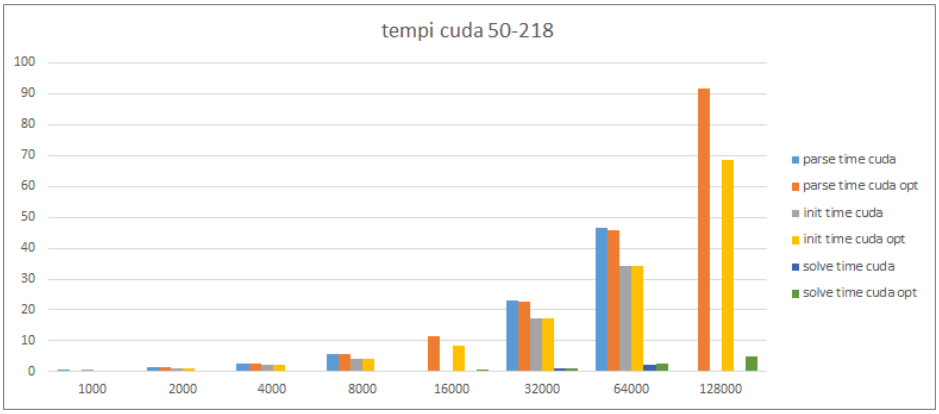


Figure A.27: 75 clauses - 325 literals

file number	parse time cuda	parse time cuda opt	init time cuda	init time cuda opt	solve time cuda	solve time cuda opt
1000	0,991874	1,038937	0,795139	0,796031	1,479003	1,499962
2000	0	0	0	0	0	0
4000	4,792898	0	3,264542	0	2,992761	0
8000	8,686811	8,478853	6,514273	6,372871	5,220209	5,255791
16000	0	0	0	0	0	0
32000	0	0	0	0	0	0
64000	67,387166	0	51,135237	0	39,713508	0
128000	0	0	0	0	0	0
Totale complessivo	**81,858749**	**9,51779**	**61,709191**	**7,168902**	**49,405481**	**6,755753**

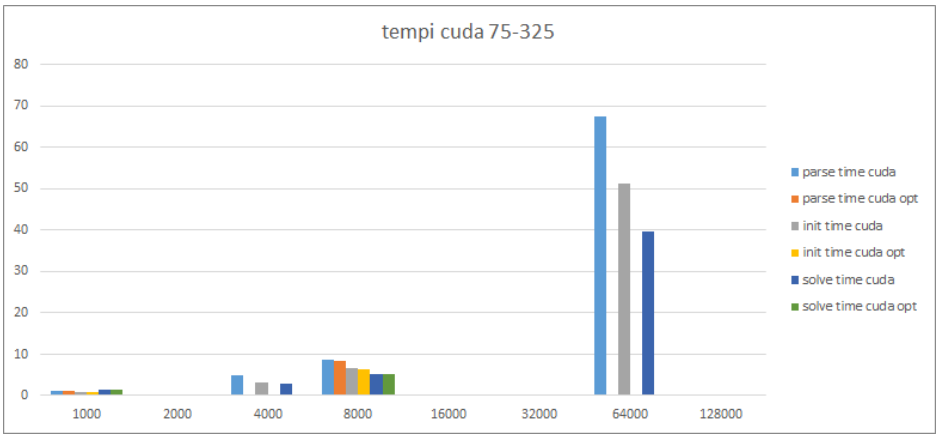


Figure A.28: 100 clauses - 430 literals

file number	parse time cuda	parse time cuda opt	init time cuda	init time cuda opt	solve time cuda	solve time cuda opt
1000	2,619074	5,712004	2,10391	4,28816	7,45358	14,436148
2000	5,177704	11,136922	4,206806	8,668818	9,98389	19,095428
4000	10,854068	20,622834	8,591596	16,824178	14,42098	27,571912
8000	20,881406	42,44045	16,838158	33,825592	24,135468	48,011344
16000	43,165868	0	33,862214	0	44,999042	0
32000	0	0	0	0	0	0
64000	0	0	0	0	0	0
128000	0	0	0	0	0	0
Totale complessivo	**82,69812**	**79,91221**	**65,602684**	**63,606748**	**100,99296**	**109,114832**

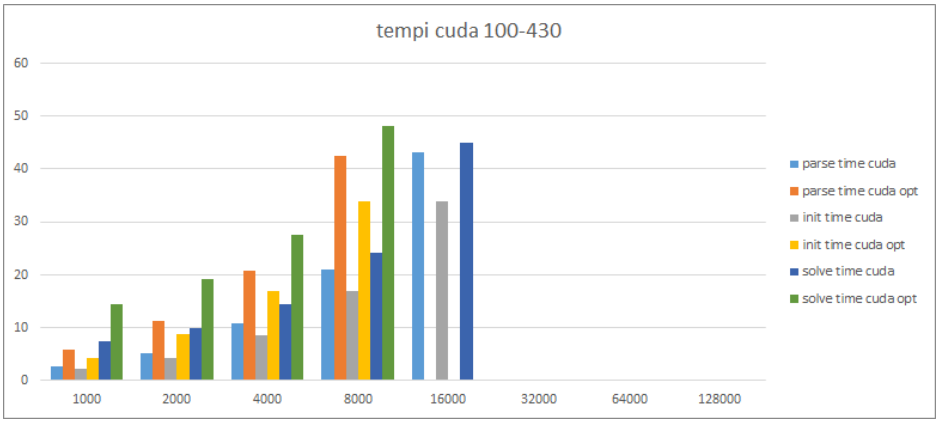


Figure A.29: 125 clauses - 538 literals

file number	parse time cuda	parse time cuda opt	init time cuda	init time cuda opt	solve time cuda	solve time cuda opt
1000	1,619478	1,791014	1,312536	1,318328	93,055188	92,679797
2000	3,274156	3,203124	2,63205	2,629467	144,278656	146,566188
4000	7,384884	0	5,275355	0	240,812703	0
8000	13,563105	0	10,526588	0	435,432	0
16000	0	0	0	0	0	0
32000	0	0	0	0	0	0
64000	0	0	0	0	0	0
128000	0	0	0	0	0	0
Totale complessivo	**25,841623**	**4,994138**	**19,746529**	**3,947795**	**913,578547**	**239,245985**

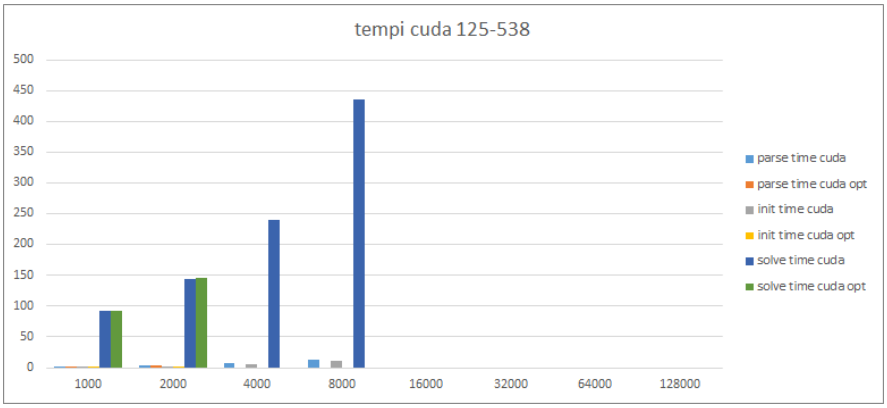
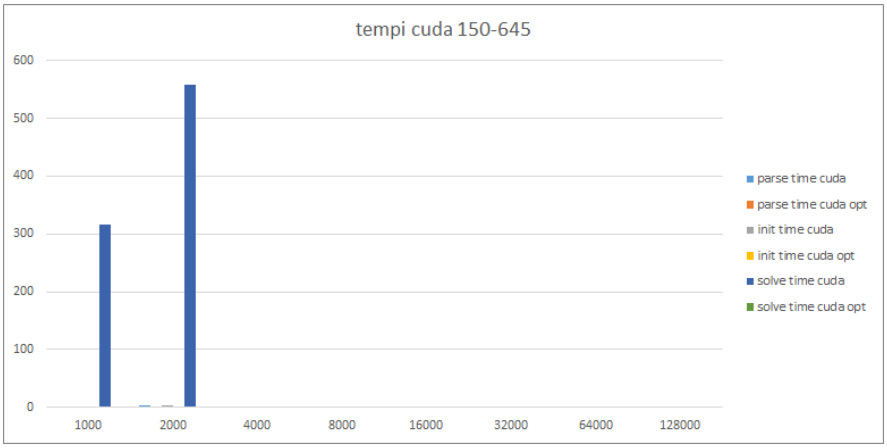


Figure A.30: 150 clauses - 645 literals

file number	parse time cuda	parse time cuda opt	init time cuda	init time cuda opt	solve time cuda	solve time cuda opt
1000	2,004369	0	1,57627	0	315,142781	0
2000	4,058534	0	3,151605	0	556,687125	0
4000	0	0	0	0	0	0
8000	0	0	0	0	0	0
16000	0	0	0	0	0	0
32000	0	0	0	0	0	0
64000	0	0	0	0	0	0
128000	0	0	0	0	0	0
Totale complessivo	**6,062903**	**0**	**4,727875**	**0**	**871,829906**	**0**



A.4 Memory used comparison betweenMiniSAT, MicroSAT, mcuda (MicroSAT-CUDAv1) and mcuda-opt (MicroSAT-CUDAv2)

vars_number	clauses_number	file_number	mem_minisat(Mbyte)	mem_microsat(Mbyte)	mem_mcuda(Mbyte)	mem_mcuda_opt(Mbyte)
20	91	1000	10000	0,788604	8	8
20	91	2000	20000	1,577208	14	14
20	91	4000	40000	3,154416	26	26
20	91	8000	80000	6,308832	52	52
20	91	16000	160000	12,617664	102	102
20	91	32000	320000	25,235328	204	204
20	91	64000	640000	50,470656	408	404
20	91	128000	1280000	100,941312	816,9375	804
50	218	1000	10000	2,087051	20	0
50	218	2000	20000	4,174102	38	38
50	218	4000	40000	8,348204	74	72
50	218	8000	80000	16,696408	146	146,9375
50	218	16000	160000	33,392816	0	286
50	218	32000	320000	66,785632	580	570
50	218	64000	640000	133,571264	1158	1140,1875
50	218	128000	1280000	267,142528	0	2274
75	325	1000	10000	4,498206	84	84,4375
75	325	2000	20000	8,996412	0	0
75	325	4000	40000	17,992824	334	0
75	325	8000	80000	35,985648	666	664
75	325	16000	160000	71,971296	0	0
75	325	32000	320000	143,942592	0	0
75	325	64000	640000	287,885184	5314	0
75	325	128000	1280000	575,770368	0	0
100	430	1000	10000	7,989825	144	142
100	430	2000	20000	15,97965	286	284
100	430	4000	40000	31,9593	572	568
100	430	8000	80000	63,9186	1144	1134
100	430	16000	160000	127,8372	2286	0
100	430	32000	320000	255,6744	0	0
100	430	64000	640000	511,3488	0	0
100	430	128000	1280000	1022,6976	0	0
125	538	1000	10000	31,223805	500	468
125	538	2000	20000	62,44761	1000	925,9375
125	538	4000	40000	124,89522	1989,9375	0
125	538	8000	80000	249,79044	4000	0
125	538	16000	160000	499,58088	0	0
125	538	32000	320000	999,16176	0	0
125	538	64000	640000	1998,32352	0	0
125	538	128000	1280000	3996,64704	0	0
150	645	1000	10000	97,421593	2002,1875	0
150	645	2000	20000	194,843186	4006,3125	0
150	645	4000	40000	389,686372	0	0
150	645	8000	80000	779,372744	0	0
150	645	16000	160000	1558,745488	0	0
150	645	32000	320000	2319,51664	0	0
150	645	64000	640000	2322,768107	0	0
150	645	128000	1280000	2320,332881	0	0

Figure A.31

Figure A.32: 20 clauses - 91 literals

file number	minisat(Mbyte)	microsat(Mbyte)	mcuda(Mbyte)	mcuda_opt(Mbyte)
1000	10000	0,788604	8	8
2000	20000	1,577208	14	14
4000	40000	3,154416	26	26
8000	80000	6,308832	52	52
16000	160000	12,617664	102	102
32000	320000	25,235328	204	204
64000	640000	50,470656	408	404
128000	1280000	100,941312	816,9375	804
Totale complessivo	**2550000**	**201,09402**	**1630,9375**	**1614**

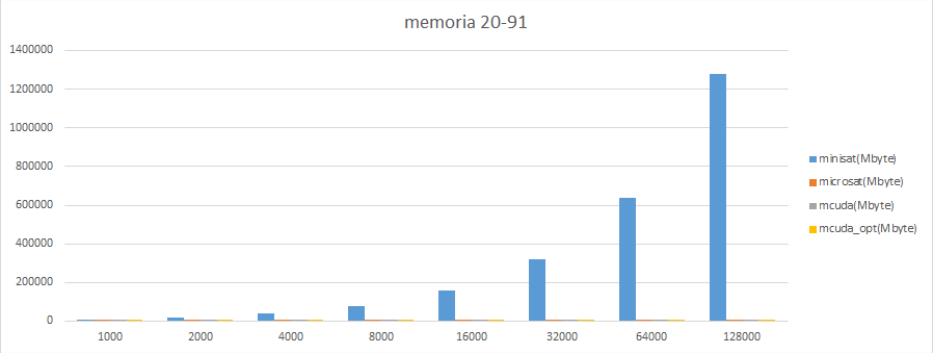


Figure A.33: 50 clauses - 218 literals

file number	minisat(Mbyte)	microsat(Mbyte)	mcuda(Mbyte)	mcuda_opt(Mbyte)
1000	10000	2,087051	20	0
2000	20000	4,174102	38	38
4000	40000	8,348204	74	72
8000	80000	16,696408	146	146,9375
16000	160000	33,392816	0	286
32000	320000	66,785632	580	570
64000	640000	133,571264	1158	1140,1875
128000	1280000	267,142528	0	2274
Totale complessivo	**2550000**	**532,198005**	**2016**	**4527,125**

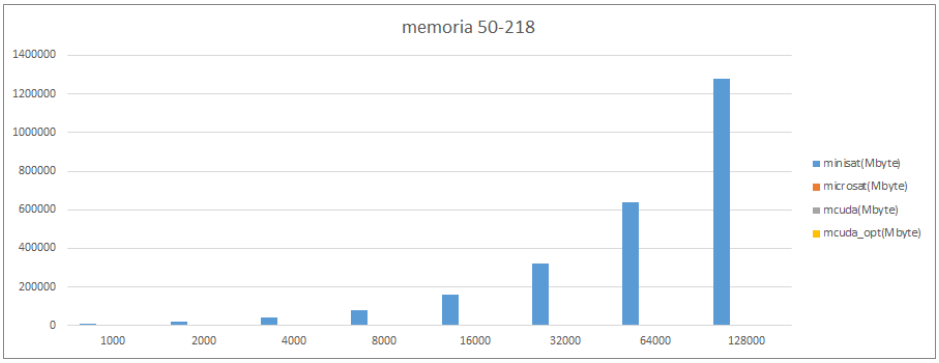


Figure A.34: 75 clauses - 325 literals

file number	minisat(Mbyte)	microsat(Mbyte)	mcuda(Mbyte)	mcuda_opt(Mbyte)
1000	10000	4,498206	84	84,4375
2000	20000	8,996412	0	0
4000	40000	17,992824	334	0
8000	80000	35,985648	666	664
16000	160000	71,971296	0	0
32000	320000	143,942592	0	0
64000	640000	287,885184	5314	0
128000	1280000	575,770368	0	0
Totale complessivo	**2550000**	**1147,04253**	**6398**	**748,4375**

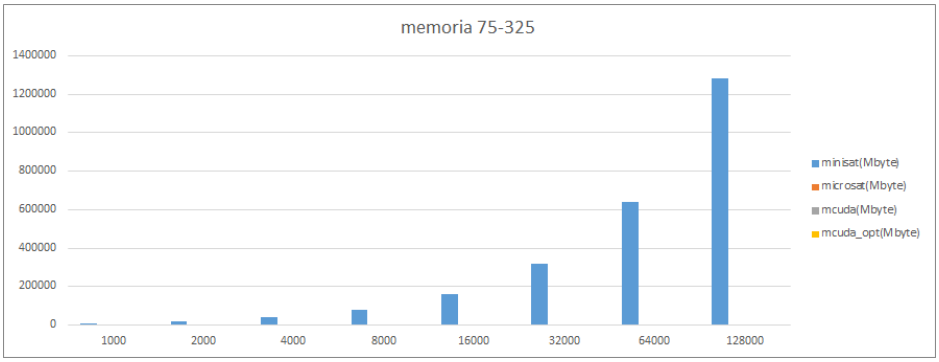


Figure A.35: 100 clauses - 430 literals

file number	minisat(Mbyte)	microsat(Mbyte)	mcuda(Mbyte)	mcuda_opt(Mbyte)
1000	10000	7,989825	144	142
2000	20000	15,97965	286	284
4000	40000	31,9593	572	568
8000	80000	63,9186	1144	1134
16000	160000	127,8372	2286	0
32000	320000	255,6744	0	0
64000	640000	511,3488	0	0
128000	1280000	1022,6976	0	0
Totale complessivo	**2550000**	**2037,405375**	**4432**	**2128**

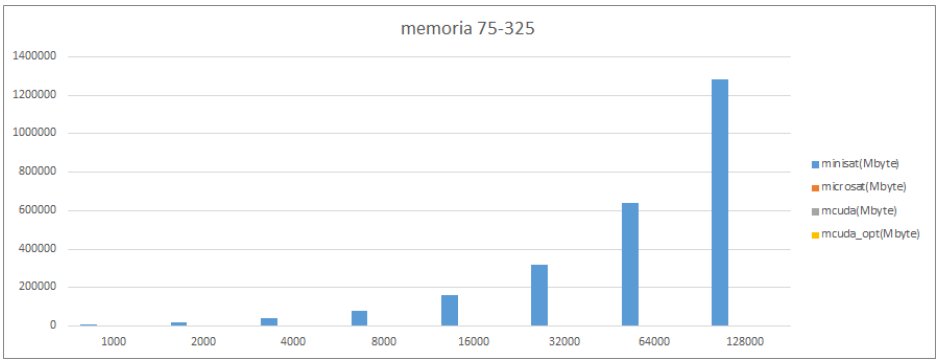


Figure A.36: 125 clauses - 538 literals

file number	minisat(Mbyte)	microsat(Mbyte)	mcuda(Mbyte)	mcuda_opt(Mbyte)
1000	10000	31,223805	500	468
2000	20000	62,44761	1000	925,9375
4000	40000	124,89522	1989,9375	0
8000	80000	249,79044	4000	0
16000	160000	499,58088	0	0
32000	320000	999,16176	0	0
64000	640000	1998,32352	0	0
128000	1280000	3996,64704	0	0
Totale complessivo	**2550000**	**7962,070275**	**7489,9375**	**1393,9375**

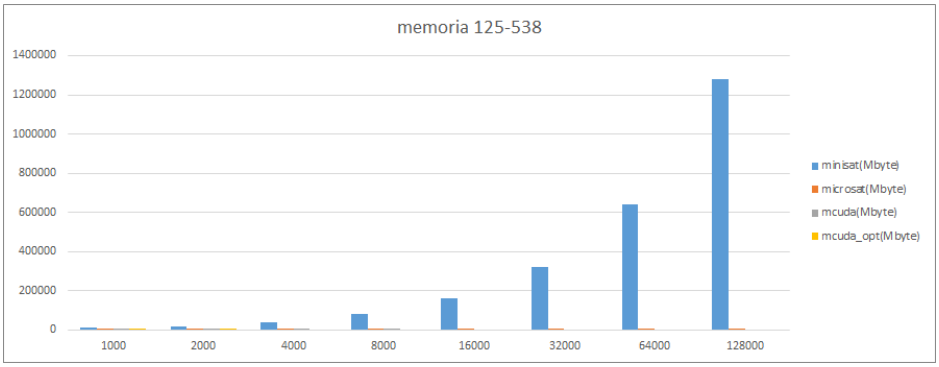
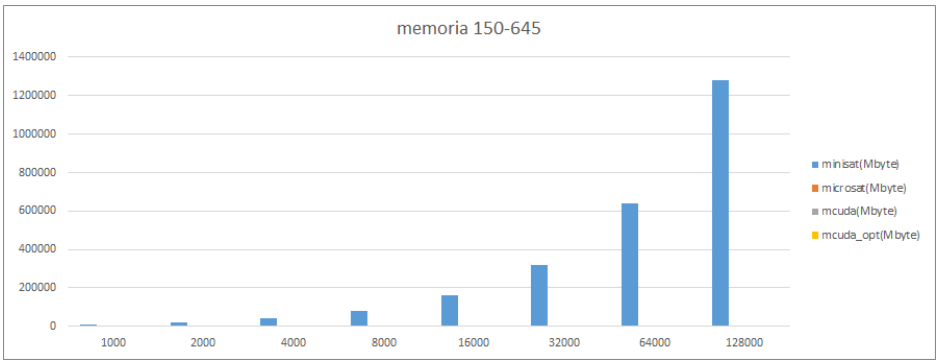


Figure A.37: 150 clauses - 645 literals

file number	minisat(Mbyte)	microsat(Mbyte)	mcuda(Mbyte)	mcuda_opt(Mbyte)
1000	10000	97,421593	2002,1875	0
2000	20000	194,843186	4006,3125	0
4000	40000	389,686372	0	0
8000	80000	779,372744	0	0
16000	160000	1558,745488	0	0
32000	320000	2319,51664	0	0
64000	640000	2322,768107	0	0
128000	1280000	2320,332881	0	0
Totale complessivo	**2550000**	**9982,687011**	**6008,5**	**0**



B Cluster

B.1 Conflicts number comparison between mcuda-opt with single GPU (MicroSAT-CUDAv2) and multi-gpu (MicroSAT-CUDAv3)

vars_number	clauses_number	file_number	db_max_mem	clause_learn_max_mem	initial_max_lemmas	mcuda-opt_tot_conflicts	multigpu_conflicts_gpu_0	multigpu_conflicts_gpu_1	multigpu_conflicts_gpu_2	multigpu_conflicts_gpu_3	multigpu_tot_conflicts
20	91	1000	1000	1000	2000	5454	1399	1382	1352	1321	5454
20	91	2000	1000	1000	2000	10908	2677	2786	2682	2763	10908
20	91	4000	1000	1000	2000	21816	5300	5634	5370	5512	21816
20	91	8000	1000	1000	2000	23994	10994	11055	10805	10778	43632
20	91	16000	1000	1000	2000	24192	21930	21898	21636	21800	87264
20	91	32000	1000	1000	2000	35750	23889	24442	24405	29714	102450
20	91	64000	1000	1000	2000	24157	24143	24179	24504	24036	96862
20	91	128000	1000	1000	2000	24590	24250	24367	24470	29985	103072
50	218	1000	3500	1000	2000	35150	9259	8908	8790	8193	35150
50	218	2000	3500	1000	2000	70300	18180	17535	17324	17261	70300
50	218	4000	3500	1000	2000	140600	35253	35843	34275	35229	140600
50	218	8000	3500	1000	2000	220680	70456	69694	70811	70239	281200
50	218	16000	3500	1000	2000	404569	139324	141360	142377	139339	562400
50	218	32000	3500	1000	2000	798090	242753	278131	228201	255883	1004968
50	218	64000	3500	1000	2000	1526781	173232	467593	320611	286841	1248277
50	218	128000	3500	1000	2000	2672265	690045	422016	825642	395483	2333186
75	325	1000	25000	2000	4000	332397	84366	79740	87393	80898	332397
75	325	2000	25000	2000	4000	664794	167010	163466	167182	167136	664794
75	325	4000	25000	2000	4000	1329588	336072	318638	339654	335224	1329588
75	325	8000	25000	2000	4000	2659176	677458	645338	675092	661288	2659176
75	325	16000	25000	2000	4000	5318352	1350665	1311455	1338298	1317934	5318352
75	325	32000	25000	2000	4000	0	2662812	2638390	2672715	2662787	10636704
75	325	64000	25000	2000	4000	0	5294178	5294120	5375489	5309621	21273408
75	325	128000	25000	2000	4000	0	0	0	0	0	0
100	430	1000	40000	2000	4000	444802	98769	96821	118617	130595	444802
100	430	2000	40000	2000	4000	889604	215709	224944	217113	231838	889604
100	430	4000	40000	2000	4000	1779208	415623	437412	454438	471735	1779208
100	430	8000	40000	2000	4000	3558416	859682	876220	887908	934606	3558416
100	430	16000	40000	2000	4000	0	1731970	1765611	1792438	1826813	7116832
100	430	32000	40000	2000	4000	0	3567250	3579818	3496648	3589948	14233664
100	430	64000	40000	2000	4000	0	0	0	0	0	0
100	430	128000	40000	2000	4000	0	0	0	0	0	0
125	538	1000	125000	3000	5000	5943643	1568190	1585393	1305228	1484832	5943643
125	538	2000	125000	3000	5000	11887286	3015565	3050814	2787546	3033361	11887286
125	538	4000	125000	3000	5000	23774572	5975121	6020820	5656691	6121940	23774572
125	538	8000	125000	3000	5000	0	11934583	12181421	11637710	11795430	47549144
125	538	16000	125000	3000	5000	0	23983362	23996843	23503241	23614842	95098288
125	538	32000	125000	3000	5000	0	0	0	0	0	0
125	538	64000	125000	3000	5000	0	0	0	0	0	0
125	538	128000	125000	3000	5000	0	0	0	0	0	0
150	645	1000	420000	4000	6000	18930483	5089155	4543082	4942985	4355261	18930483
150	645	2000	420000	4000	6000	0	9916926	9448475	9184081	9311484	37860966
150	645	4000	420000	4000	6000	0	19160139	18902254	18629835	19029704	75721932
150	645	8000	420000	4000	6000	0	0	0	0	0	0
150	645	16000	420000	4000	6000	0	0	0	0	0	0
150	645	32000	420000	4000	6000	0	0	0	0	0	0
150	645	64000	420000	4000	6000	0	0	0	0	0	0
150	645	128000	420000	4000	6000	0	0	0	0	0	0

Figure B.38

Figure B.39: 20 clauses - 91 literals

file number	multi-gpu	mcudaopt
1000	5454	5454
2000	10908	10908
4000	21816	21816
8000	43632	23994
16000	87264	24192
32000	102450	35750
64000	96862	24157
128000	103072	24590
Totale complessivo	**471458**	**170861**

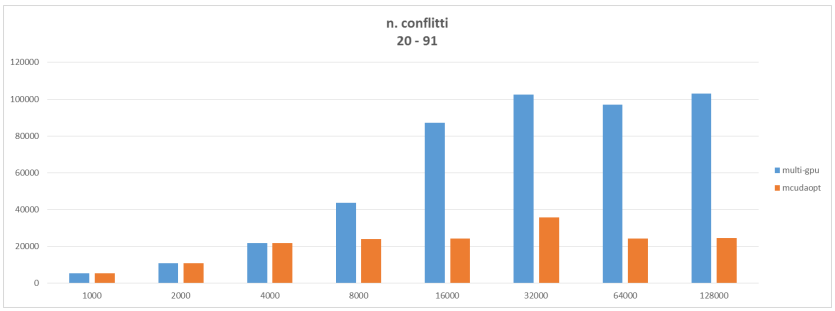


Figure B.40: 50 clauses - 218 literals

file number	multi-gpu	mcudaopt
1000	35150	35150
2000	70300	70300
4000	140600	140600
8000	281200	220680
16000	562400	404569
32000	1004968	798090
64000	1248277	1526781
128000	2333186	2672265
Totale complessivo	**5676081**	**5868435**

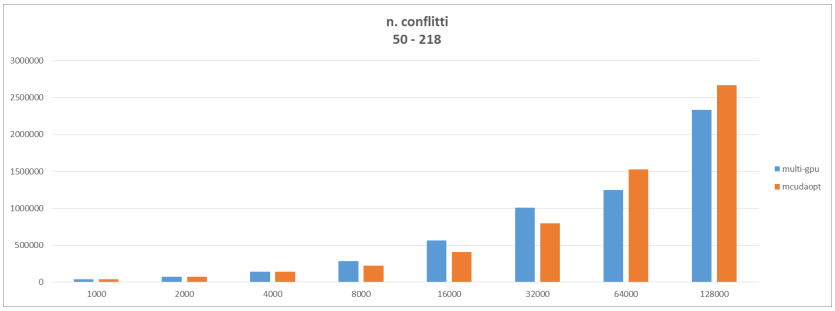


Figure B.41: 75 clauses - 325 literals

file number	multi-gpu	mcudaopt
1000	332397	332397
2000	664794	664794
4000	1329588	1329588
8000	2659176	2659176
16000	5318352	5318352
32000	10636704	0
64000	21273408	0
128000	0	0
Totale complessivo	**42214419**	**10304307**

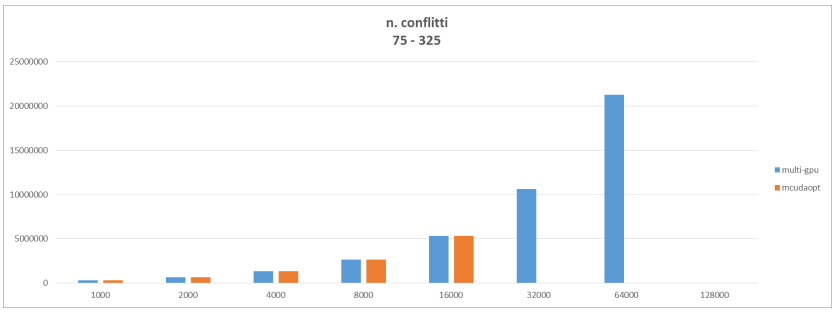


Figure B.42: 100 clauses - 430 literals

file number	multi-gpu	mcudaopt
1000	444802	444802
2000	889604	889604
4000	1779208	1779208
8000	3558416	3558416
16000	7116832	0
32000	14233664	0
64000	0	0
128000	0	0
Totale complessivo	**28022526**	**6672030**

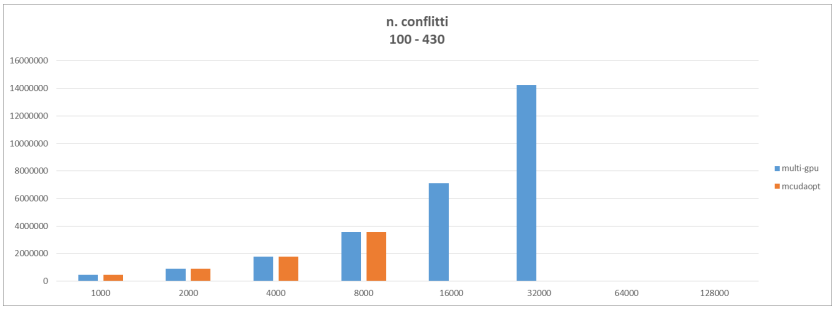


Figure B.43: 125 clauses - 538 literals

file number	multi-gpu	mcuda-opt
1000	18930483	18930483
2000	37860966	0
4000	75721932	0
8000	0	0
16000	0	0
32000	0	0
64000	0	0
128000	0	0
Totale complessivo	**132513381**	**18930483**

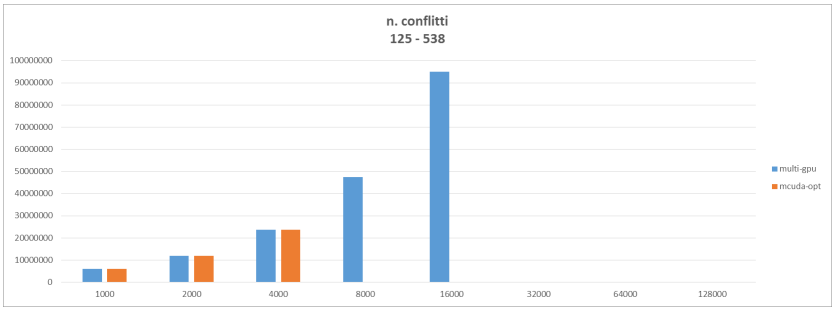
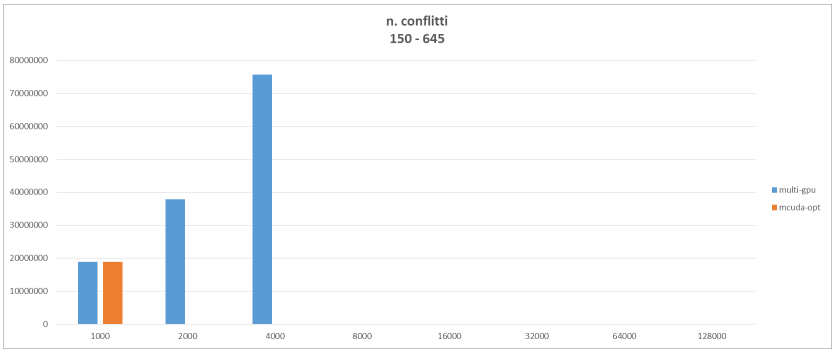


Figure B.44: 150 clauses - 645 literals

file number	multi-gpu	mcuda-opt
1000	18930483	18930483
2000	37860966	0
4000	75721932	0
8000	0	0
16000	0	0
32000	0	0
64000	0	0
128000	0	0
Totale complessivo	**132513381**	**18930483**



B.2 Execution time comparison between mcuda-opt with single GPU (MicroSAT-CUDAv2) and multi-gpu (MicroSAT-CUDAv3)

vars_number	clauses_number	file_number	db_max_mem	clause_learn_max_mem	initial_max_lemmas	mcuda-opt_time_ms	mcuda-opt_parse_time	mcuda-opt_init_time	mcuda-opt_solve_time	multigpu_time_ms	multigpu_parse_time	multigpu_init_time	multigpu_solve_time	mcuda-opt_time_s	multigpu_time_s
20	91	1000	1000	1000	2000	1397	0,47	0,310866	0,003498	2703	0,49	0,320122	0	1,397	2,703
20	91	2000	1000	1000	2000	2170	0,88	0,544185	0,005052	3619	0,96	0,626214	0	2,17	3,619
20	91	4000	1000	1000	2000	3872	1,78	1,077295	0,008236	4804	1,85	1,156762	0	3,872	4,804
20	91	8000	1000	1000	2000	7130	3,59	2,096776	0,015037	8468	3,73	2,222438	0	7,13	8,468
20	91	16000	1000	1000	2000	14474	7,4	4,216495	0,028322	16150	7,88	4,620902	0	14,474	16,15
20	91	32000	1000	1000	2000	29242	15,87	8,993356	0,057921	29372	15,38	8,990496	0	29,242	29,372
20	91	64000	1000	1000	2000	73809	35	20,081333	0,108832	57605	31,85	18,434579	0	73,809	57,605
20	91	128000	1000	1000	2000	177511	66,9	38,151363	0,215661	113032	64,78	37,248942	0	177,511	113,032
50	218	1000	3500	1000	2000	1910	0,89	0,647137	0,060167	3400	0,96	0,728751	0	1,91	3,4
50	218	2000	3500	1000	2000	3064	1,71	1,235624	0,076136	4630	1,87	1,388625	0	3,064	4,63
50	218	4000	3500	1000	2000	5580	3,47	2,474027	0,111218	7056	3,58	2,608784	0	5,58	7,056
50	218	8000	3500	1000	2000	10516	6,92	4,889787	0,176802	11932	7,02	5,002743	0	10,516	11,932
50	218	16000	3500	1000	2000	21362	13,96	9,721976	0,313517	22340	13,95	9,773952	0	21,362	22,34
50	218	32000	3500	1000	2000	44134	28,14	19,400131	0,575636	43020	27,86	19,400213	0	44,134	43,02
50	218	64000	3500	1000	2000	92173	56,84	38,714149	1,112295	82536	55,81	38,633369	1,41508	92,173	82,536
50	218	128000	3500	1000	2000	284757	114,14	78,614767	2,172444	162142	111,87	77,079298	0	284,757	162,142
75	325	1000	25000	2000	4000	11448	1,55	1,123396	1,319355	15029	1,55	1,123297	0	11,448	15,029
75	325	2000	25000	2000	4000	37984	3,18	2,251844	1,570365	37891	3,24	2,259859	0	37,984	37,891
75	325	4000	25000	2000	4000	78031	6,39	4,500943	2,023666	78765	6,61	4,514232	0	78,031	78,765
75	325	8000	25000	2000	4000	159745	13,09	9,015232	2,98533	152025	13,19	8,988582	0	159,745	152,025
75	325	16000	25000	2000	4000	324741	26,01	18,026563	5,069058	300609	26,08	17,994874	3,671056	324,741	300,609
75	325	32000	25000	2000	4000	0	0	0	0	601239	52,39	36,057693	6,191923	0	601,239
75	325	64000	25000	2000	4000	0	0	0	0	1220941	105,29	72,018021	0	0	1220,941
75	325	128000	25000	2000	4000	0	0	0	0	0	0	0	0	0	0
100	430	1000	40000	2000	4000	15522	1,99	1,480383	3,249065	21996	1,98	1,481376	0	15,522	21,996
100	430	2000	40000	2000	4000	40205	4,05	2,973848	3,646556	47447	4,1	2,966267	0	40,205	47,447
100	430	4000	40000	2000	4000	85040	8,09	5,949496	4,682041	83650	8,27	5,935744	9,795187	85,04	83,65
100	430	8000	40000	2000	4000	165635	15,89	11,89134	7,064789	157981	16,34	11,874358	0	165,635	157,981
100	430	16000	40000	2000	4000	0	0	0	0	312807	32,91	23,790636	13,505071	0	312,807
100	430	32000	40000	2000	4000	0	0	0	0	629181	65,87	47,490248	20,53672	0	629,181
100	430	64000	40000	2000	4000	0	0	0	0	0	0	0	0	0	0
100	430	128000	40000	2000	4000	0	0	0	0	0	0	0	0	0	0
125	538	1000	125000	3000	5000	98122	2,38	1,847306	87,012039	279892	2,37	1,848966	251,59166	98,122	279,892
125	538	2000	125000	3000	5000	140544	4,86	3,715041	103,904453	325715	4,99	3,710803	203,886477	140,544	325,715
125	538	4000	125000	3000	5000	228331	9,82	7,43259	151,926406	388239	9,89	7,419957	230,791336	228,331	388,239
125	538	8000	125000	3000	5000	0	0	0	0	536157	20,07	14,838651	384,912156	0	536,157
125	538	16000	125000	3000	5000	0	0	0	0	894957	39,87	29,667501	587,275125	0	894,957
125	538	32000	125000	3000	5000	0	0	0	0	0	0	0	0	0	0
125	538	64000	125000	3000	5000	0	0	0	0	0	0	0	0	0	0
125	538	128000	125000	3000	5000	0	0	0	0	0	0	0	0	0	0
150	645	1000	420000	4000	6000	230500	2,76	2,212578	218,123312	661618	2,82	2,214723	648,858641	230,5	661,618
150	645	2000	420000	4000	6000	0	0	0	0	723418	5,75	4,439476	505,731078	0	723,418
150	645	4000	420000	4000	6000	0	0	0	0	907130	11,59	8,881305	830,307141	0	907,13
150	645	8000	420000	4000	6000	0	0	0	0	0	0	0	0	0	0
150	645	16000	420000	4000	6000	0	0	0	0	0	0	0	0	0	0
150	645	32000	420000	4000	6000	0	0	0	0	0	0	0	0	0	0
150	645	64000	420000	4000	6000	0	0	0	0	0	0	0	0	0	0
150	645	128000	420000	4000	6000	0	0	0	0	0	0	0	0	0	0

Figure B.45

Figure B.46: 20 clauses - 91 literals

file number	multi-gpu	mcudaopt
1000	2,703	1,397
2000	3,619	2,17
4000	4,804	3,872
8000	8,468	7,13
16000	16,15	14,474
32000	29,372	29,242
64000	57,605	73,809
128000	113,032	177,511
Totale complessivo	**235,753**	**309,605**

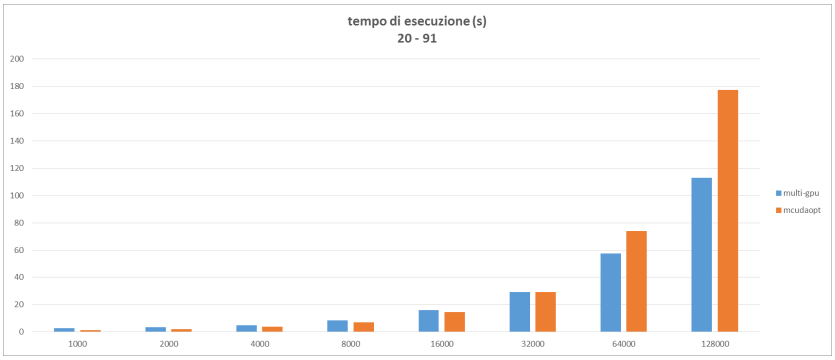


Figure B.47: 50 clauses - 218 literals

file number	multi-gpu	mcudaopt
1000	3,4	1,91
2000	4,63	3,064
4000	7,056	5,58
8000	11,932	10,516
16000	22,34	21,362
32000	43,02	44,134
64000	82,536	92,173
128000	162,142	284,757
Totale complessivo	**337,056**	**463,496**

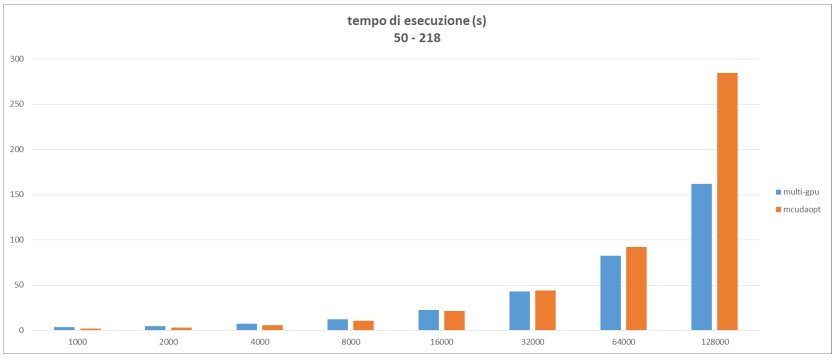


Figure B.48: 75 clauses - 325 literals

file number	multi-gpu	mcudaopt
1000	15,029	11,448
2000	37,891	37,984
4000	78,765	78,031
8000	152,025	159,745
16000	300,609	324,741
32000	601,239	0
64000	1220,941	0
128000	0	0
Totale complessivo	**2406,499**	**611,949**

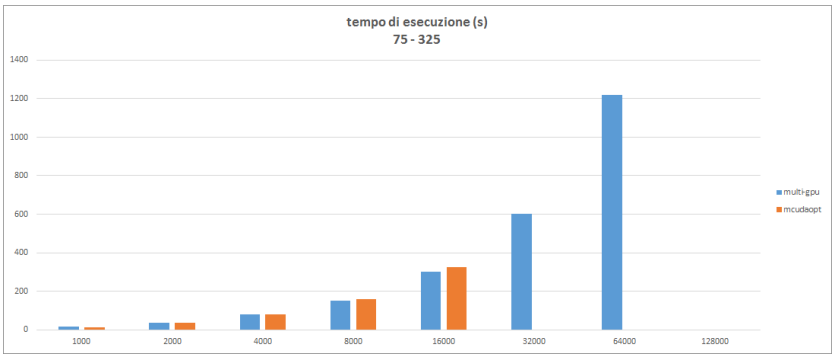


Figure B.49: 100 clauses - 430 literals

file number	multi-gpu	mcudaopt
1000	21,996	15,522
2000	47,447	40,205
4000	83,65	85,04
8000	157,981	165,635
16000	312,807	0
32000	629,181	0
64000	0	0
128000	0	0
Totale complessivo	**1253,062**	**306,402**

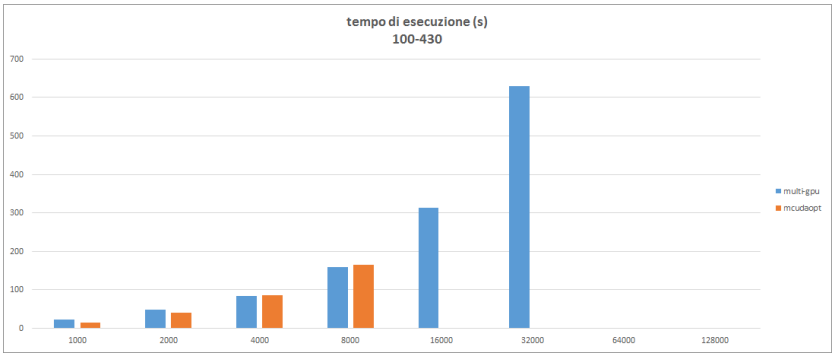


Figure B.50: 125 clauses - 538 literals

file number	multi-gpu	mcudaopt
1000	279,892	98,122
2000	325,715	140,544
4000	388,239	228,331
8000	536,157	0
16000	894,957	0
32000	0	0
64000	0	0
128000	0	0
Totale complessivo	**2424,96**	**466,997**

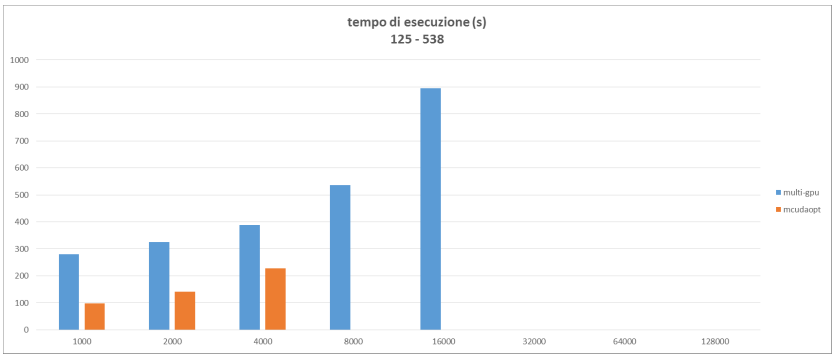
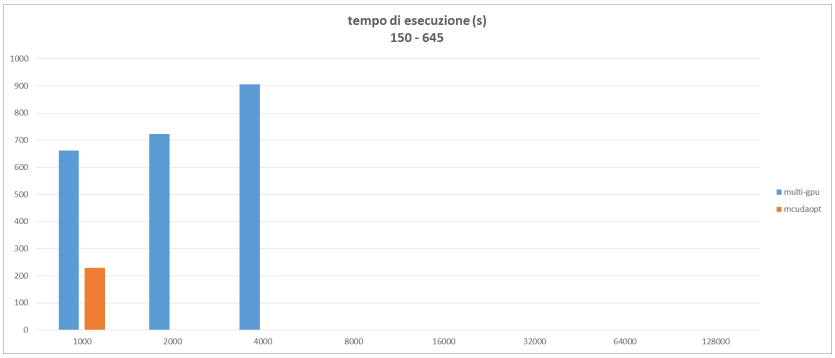


Figure B.51: 150 clauses - 645 literals

file number	multi-gpu	mcudaopt
1000	661,618	230,5
2000	723,418	0
4000	907,13	0
8000	0	0
16000	0	0
32000	0	0
64000	0	0
128000	0	0
Totale complessivo	**2292,166**	**230,5**



B.3 Time comparison between mcuda-opt with single GPU (MicroSAT-CUDAv2) and multi-gpu (MicroSAT-CUDAv3)

Figure B.52: 20 clauses - 91 literals

file number	multigpu solve time	mcudaopt solve time	mcudaopt parse time	mcudaopt init time	multigpu parse time	multigpu init time
1000	0	0,003498	0,47	0,310866	0,49	0,320122
2000	0	0,005052	0,88	0,544185	0,96	0,626214
4000	0	0,008236	1,78	1,077295	1,85	1,156762
8000	0	0,015037	3,59	2,096776	3,73	2,222438
16000	0	0,028322	7,4	4,216495	7,88	4,620902
32000	0	0,057921	15,87	8,993356	15,38	8,990496
64000	0	0,108832	35	20,081333	31,85	18,434579
128000	0	0,215661	66,9	38,151363	64,78	37,248942
Totale complessivo	**0**	**0,442559**	**131,89**	**75,471669**	**126,92**	**73,620455**

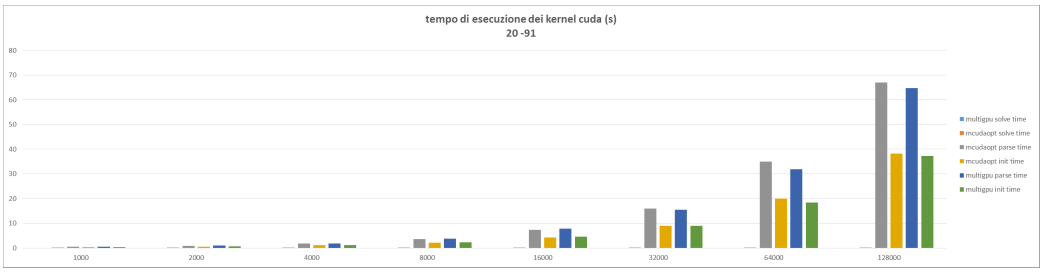


Figure B.53: 50 clauses - 218 literals

file number	multigpu solve time	mcudaopt solve time	mcudaopt parse time	mcudaopt init time	multigpu parse time	multigpu init time
1000	0	0,060167	0,89	0,647137	0,96	0,728751
2000	0	0,076136	1,71	1,235624	1,87	1,388625
4000	0	0,111218	3,47	2,474027	3,58	2,608784
8000	0	0,176802	6,92	4,889787	7,02	5,002743
16000	0	0,313517	13,96	9,721976	13,95	9,773952
32000	0	0,575636	28,14	19,400131	27,86	19,400213
64000	1,41508	1,112295	56,84	38,714149	55,81	38,633369
128000	0	2,172444	114,14	78,614767	111,87	77,079298
Totale complessivo	**1,41508**	**4,598215**	**226,07**	**155,697598**	**222,92**	**154,615735**

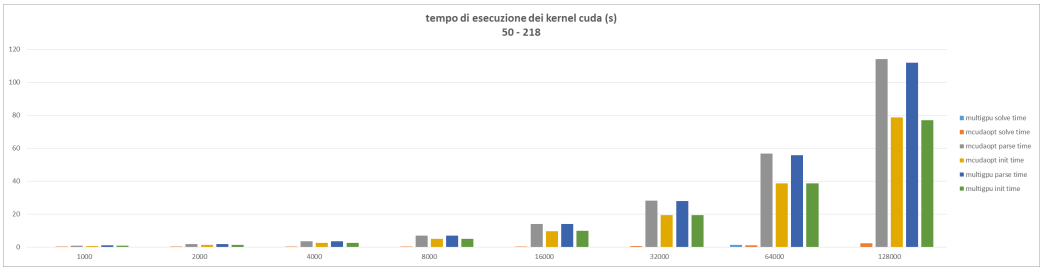


Figure B.54: 75 clauses - 325 literals

file number	multigpu solve time	mcudaopt solve time	mcudaopt parse time	mcudaopt init time	multigpu parse time	multigpu init time
1000	0	1,319355	1,55	1,123396	1,55	1,123297
2000	0	1,570365	3,18	2,251844	3,24	2,259859
4000	0	2,023666	6,39	4,500943	6,61	4,514232
8000	0	2,98533	13,09	9,015232	13,19	8,988582
16000	3,671056	5,069058	26,01	18,026563	26,08	17,994874
32000	6,191923	0	0	0	52,39	36,057693
64000	0	0	0	0	105,29	72,018021
128000	0	0	0	0	0	0
Totale complessivo	**9,862979**	**12,967774**	**50,22**	**34,917978**	**208,35**	**142,956558**

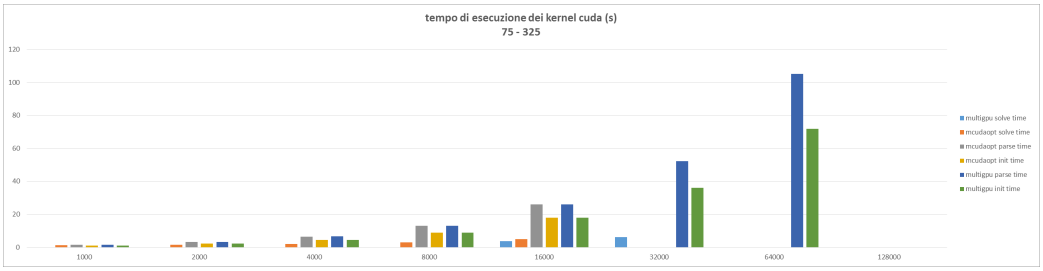


Figure B.55: 100 clauses - 430 literals

file number	multigpu solve time	mcudaopt solve time	mcudaopt parse time	mcudaopt init time	multigpu parse time	multigpu init time
1000	0	3,249065	1,99	1,480383	1,98	1,481376
2000	0	3,646556	4,05	2,973848	4,1	2,966267
4000	9,795187	4,682041	8,09	5,949496	8,27	5,935744
8000	0	7,064789	15,89	11,89134	16,34	11,874358
16000	13,505071	0	0	0	32,91	23,790636
32000	20,53672	0	0	0	65,87	47,490248
64000	0	0	0	0	0	0
128000	0	0	0	0	0	0
Totale complessivo	**43,836978**	**18,642451**	**30,02**	**22,295067**	**129,47**	**93,538629**

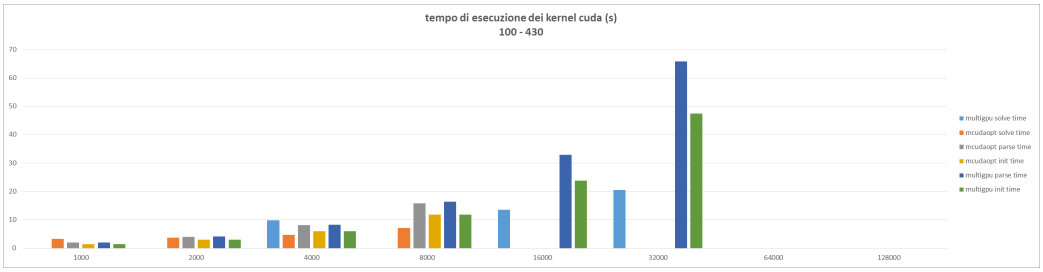


Figure B.56: 125 clauses - 538 literals

file number	multigpu solve time	mcudaopt solve time	mcudaopt parse time	mcudaopt init time	multigpu parse time	multigpu init time
1000	251,59166	87,012039	2,38	1,847306	2,37	1,848966
2000	203,886477	103,904453	4,86	3,715041	4,99	3,710803
4000	230,791336	151,926406	9,82	7,43259	9,89	7,419957
8000	384,912156	0	0	0	20,07	14,838651
16000	587,275125	0	0	0	39,87	29,667501
32000	0	0	0	0	0	0
64000	0	0	0	0	0	0
128000	0	0	0	0	0	0
Totale complessivo	**1658,456754**	**342,842898**	**17,06**	**12,994937**	**77,19**	**57,485878**

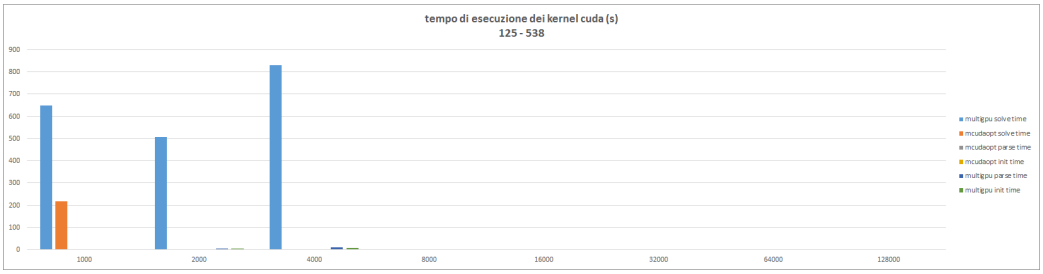
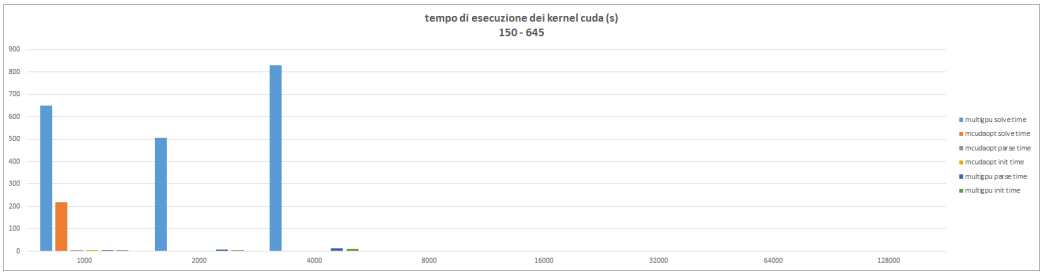


Figure B.57: 150 clauses - 645 literals

file number	multigpu solve time	mcudaopt solve time	mcudaopt parse time	mcudaopt init time	multigpu parse time	multigpu init time
1000	648,858641	218,123312	2,76	2,212578	2,82	2,214723
2000	505,731078	0	0	0	5,75	4,439476
4000	830,307141	0	0	0	11,59	8,881305
8000	0	0	0	0	0	0
16000	0	0	0	0	0	0
32000	0	0	0	0	0	0
64000	0	0	0	0	0	0
128000	0	0	0	0	0	0
Totale complessivo	**1984,89686**	**218,123312**	**2,76**	**2,212578**	**20,16**	**15,535504**



B.4 Comparison of memory usage between mcuda-opt with single GPU (MicroSAT-CUDAv2) and multi-gpu (MicroSAT-CUDAv3)

vars_number	clauses_number	file_number	db_meta_mem	clause_learn_max_mem	mtbk_max_lemmas	mcudaopt_mem(Mbyte)	multigpu_mem_gpu_0(Mbyte)	multigpu_mem_gpu_1(Mbyte)	multigpu_mem_gpu_2(Mbyte)	multigpu_mem_gpu_3(Mbyte)	multigpu_tot_mem(Mbyte)
20	91	1000	1000	1000	2000	8	2	2	2	2	8
20	91	2000	1000	1000	2000	14	4	4	4	4	16
20	91	4000	1000	1000	2000	26	8	8	8	8	32
20	91	8000	1000	1000	2000	52	14	14	14	14	56
20	91	16000	1000	1000	2000	102	26	26	26	26	104
20	91	32000	1000	1000	2000	204	52	52	52	52	208
20	91	64000	1000	1000	2000	404	102	102	102	102	408
20	91	128000	1000	1000	2000	804	204	204	204	204	816
50	218	1000	3500	1000	2000	20	6	6	6	6	24
50	218	2000	3500	1000	2000	38	12	12	12	12	48
50	218	4000	3500	1000	2000	72	20	20	20	20	80
50	218	8000	3500	1000	2000	144	38	38	38	38	152
50	218	16000	3500	1000	2000	286	72	72	72	72	288
50	218	32000	3500	1000	2000	570	144	144	144	144	576
50	218	64000	3500	1000	2000	1138	286	286	286	286	1144
50	218	128000	3500	1000	2000	2274	570	570	570	570	2280
75	325	1000	25000	2000	4000	104	26	26	26	26	104
75	325	2000	25000	2000	4000	206	52	52	52	52	208
75	325	4000	25000	2000	4000	408	104	104	104	104	416
75	325	8000	25000	2000	4000	816	206	206	206	206	824
75	325	16000	25000	2000	4000	1628	408	408	408	408	1632
75	325	32000	25000	2000	4000	0	816	816	816	816	3264
75	325	64000	25000	2000	4000	0	1628	1628	1628	1628	6512
75	325	128000	25000	2000	4000	0	0	0	0	0	0
100	430	1000	40000	2000	4000	162	42	42	42	42	168
100	430	2000	40000	2000	4000	322	82	82	82	82	328
100	430	4000	40000	2000	4000	644	162	162	162	162	648
100	430	8000	40000	2000	4000	1286	322	322	322	322	1288
100	430	16000	40000	2000	4000	0	644	644	644	644	2576
100	430	32000	40000	2000	4000	0	1286	1286	1286	1286	5144
100	430	64000	40000	2000	4000	0	0	0	0	0	0
100	430	128000	40000	2000	4000	0	0	0	0	0	0
125	538	1000	125000	3000	5000	488	124	124	124	124	496
125	538	2000	125000	3000	5000	974	246	246	246	246	984
125	538	4000	125000	3000	5000	1946	488	488	488	488	1952
125	538	8000	125000	3000	5000	0	974	974	974	974	3896
125	538	16000	125000	3000	5000	0	1946	1946	1946	1946	7784
125	538	32000	125000	3000	5000	0	0	0	0	0	0
125	538	64000	125000	3000	5000	0	0	0	0	0	0
125	538	128000	125000	3000	5000	0	0	0	0	0	0
150	645	1000	420000	4000	6000	1616	406	406	406	406	1624
150	645	2000	420000	4000	6000	0	808	808	808	808	3232
150	645	4000	420000	4000	6000	0	1616	1616	1616	1616	6464
150	645	8000	420000	4000	6000	0	0	0	0	0	0
150	645	16000	420000	4000	6000	0	0	0	0	0	0
150	645	32000	420000	4000	6000	0	0	0	0	0	0
150	645	64000	420000	4000	6000	0	0	0	0	0	0
150	645	128000	420000	4000	6000	0	0	0	0	0	0

Figure B.58

Figure B.59: 20 clauses - 91 literals

file number	multigpu (Mbyte)	mcudaopt (Mbyte)
1000	8	8
2000	16	14
4000	32	26
8000	56	52
16000	104	102
32000	208	204
64000	408	404
128000	816	804
Totale complessivo	**1648**	**1614**

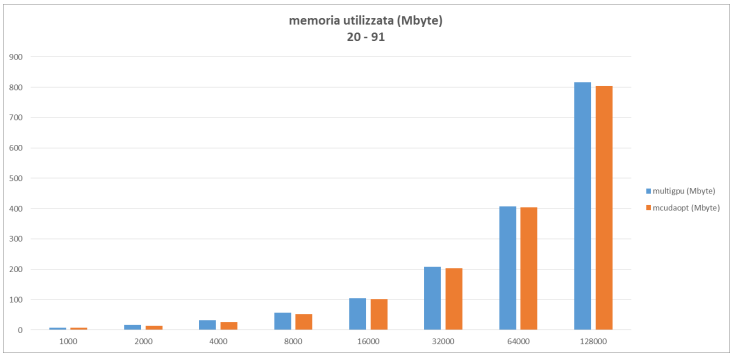


Figure B.60: 50 clauses - 218 literals

file number	multigpu (Mbyte)	mcudaopt (Mbyte)
1000	24	20
2000	48	38
4000	80	72
8000	152	144
16000	288	286
32000	576	570
64000	1144	1138
128000	2280	2274
Totale complessivo	**4592**	**4542**

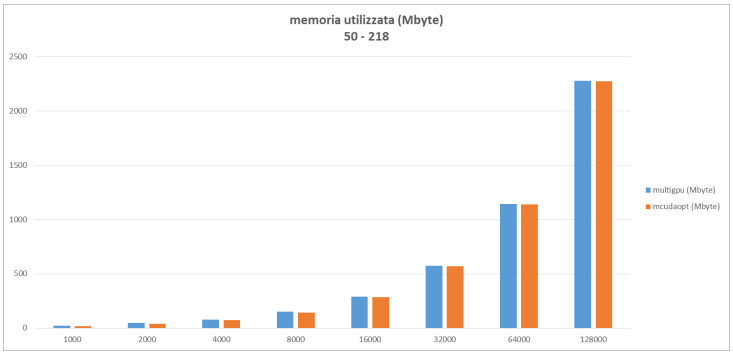


Figure B.61: 75 clauses - 325 literals

file number	multigpu (Mbyte)	mcudaopt (Mbyte)
1000	104	104
2000	208	206
4000	416	408
8000	824	816
16000	1632	1628
32000	3264	0
64000	6512	0
128000	0	0
Totale complessivo	**12960**	**3162**

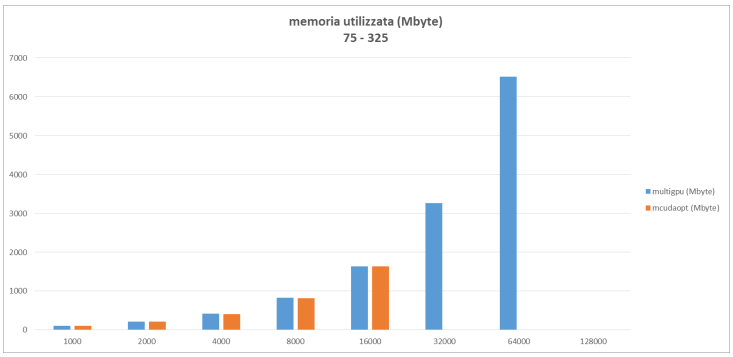


Figure B.62: 100 clauses - 430 literals

file number	multigpu (Mbyte)	mcudaopt (Mbyte)
1000	168	162
2000	328	322
4000	648	644
8000	1288	1286
16000	2576	0
32000	5144	0
64000	0	0
128000	0	0
Totale complessivo	**10152**	**2414**

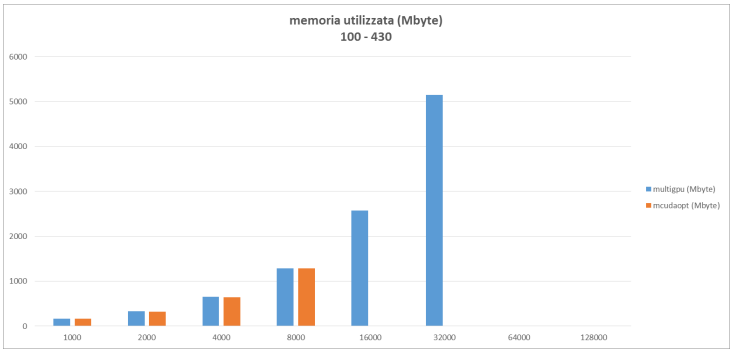


Figure B.63: 125 clauses - 538 literals

file number	multigpu (Mbyte)	mcudaopt (Mbyte)
1000	496	488
2000	984	974
4000	1952	1946
8000	3896	0
16000	7784	0
32000	0	0
64000	0	0
128000	0	0
Totale complessivo	**15112**	**3408**

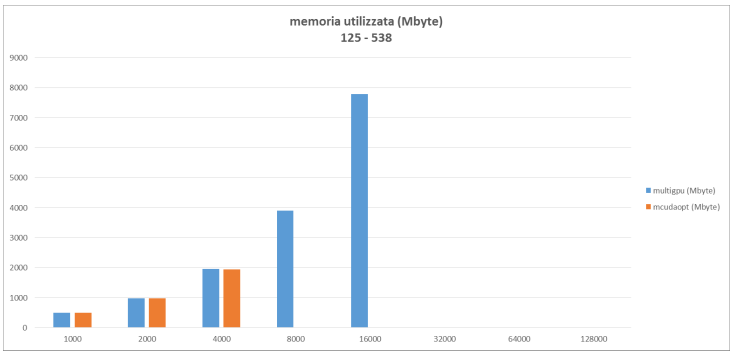
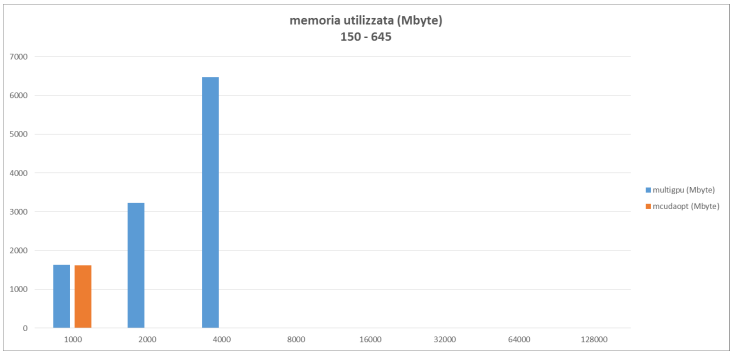


Figure B.64: 150 clauses - 645 literals

file number	multigpu (Mbyte)	mcudaopt (Mbyte)
1000	1624	1616
2000	3232	0
4000	6464	0
8000	0	0
16000	0	0
32000	0	0
64000	0	0
128000	0	0
Totale complessivo	**11320**	**1616**



References

[1] Wikipedia contributors, Boolean satisfiability problem — Wikipedia, the free encyclopedia, 2020. [Online; accessed 28-June-2020].

[2] M. Davis, H. Putnam, A computing procedure for quantification theory, Journal of the ACM (JACM) 7 (1960) 201–215.

[3] M. Davis, G. Logemann, D. Loveland, A machine program for theorem-proving, Communications of the ACM 5 (1962) 394–397.

[4] A. Biere, M. Heule, H. van Maaren, T. Walsh, Conflict-driven clause learning sat solvers, Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications (2009) 131–153.

[5] C. Costa, Parallelization of SAT Algorithms on GPUs, Technical Report, Technical report, INESC-ID, Technical University of Lisbon, 2013.

[6] S. Beckers, G. De Samblanx, F. De Smedt, T. Goedemé, L. Struyf, J. Vennekens, Parallel hybrid sat solving using opencl, Proceedings BNAIC 2012 (2012) 11–18.

[7] A. Dal Palù, A. Dovier, A. Formisano, E. Pontelli, Cud@ sat: Sat solving on gpus, Journal of Experimental & Theoretical Artificial Intelligence 27 (2015) 293–316.

[8] NVIDIA, The programming guide to the cuda model and interface, 2020. [Online; accessed 28-June-2020].

[9] NVIDIA, Unified memory for cuda beginners, 2020. [Online; accessed 28-June-2020].