Efficient Usage of CPU and GPU Hardware Resources

This application note presents how is the efficiency of CPU and GPU parallelization in WIPL-D software affected with the electrical size of a problem. In all of the simulations carried out to complete this application note, CPU matrix fill-in and GPU matrix inversion were used.

Introduction

In WIPL-D software, a simulation has two main phases, the **matrix fill-in** followed with the **matrix inversion**. For electrically small problems, where the MoM matrix has up to a few thousands of unknown coefficients (unknowns), the fill-in time is dominant. For electrically larger problems, the matrix inversion time becomes a more significant portion of the total simulation time. The matrix fill-in time raises as O(N^2), where N is the number of unknowns. In general, matrix inversion time raises as O(N^3).

Since the simulation time for electrically small EM problems is usually not critical, we consider WIPL-D capabilities on CPU and GPU platforms for electrically moderate or large solutions.

The matrix fill-in time depends on several factors, such as shape of the mesh plates, the relative position of the plates, orders of current approximation used, etc. For this reason, two different problems are simulated: generic model of an airplane (with a varying shape of the mesh elements) and metallic cubes (where all mesh elements are the identical squares).

Unlike matrix fill-in time, the matrix inversion time depends only on the number of unknown coefficients.

In the first section of the application note, we address the efficiency of WIPL-D matrix fill-in for modern multicore CPUs. In the second part, we focus on the importance of using GPUs for speeding up the matrix inversion phase. In the last part, it will be shown that the matrix inversion phase becomes dominant as number of unknowns increases, and the best strategy for solving electrically large problems is suggested.

Efficient Matrix Fill-in on Multi-Core CPUs

The most important aspect of the matrix fill-in is to exploit the advantage of modern CPUs, which all have **multi-core capabilities**. Having a quad core CPU has become a standard for the desktop and laptop PCs, and is even emerging as the standard for various mobile devices. WIPL-D has followed this trend acknowledging that the modern desktop CPUs could be available with 6, 8, 10, 12, 16 or more cores. The modern motherboards support more than one multi core CPUs coming with an affordable desktop PC or a small workstation (with mostly 2 CPUs, but also 3, 4…). In that sense, it is not uncommon to encounter a desktop PCs with 20, 24 or 32 cores. With the multi-threading applied, it turns into 40, 48 or 64 threads, respectively.

WIPL-D kernel fully exploits such a huge advantage of modern PCs with **highly efficient matrix fill-in**. In the following figures, we will illustrate the speed-up and parallelization efficiency achieved by using more than standard 4 cores, when generic model of an airplane is simulated. On the PC with 24 physical cores, we run WIPL-D simulation with WIPL-D kernel set to use 4, 8 and all 48 threads.

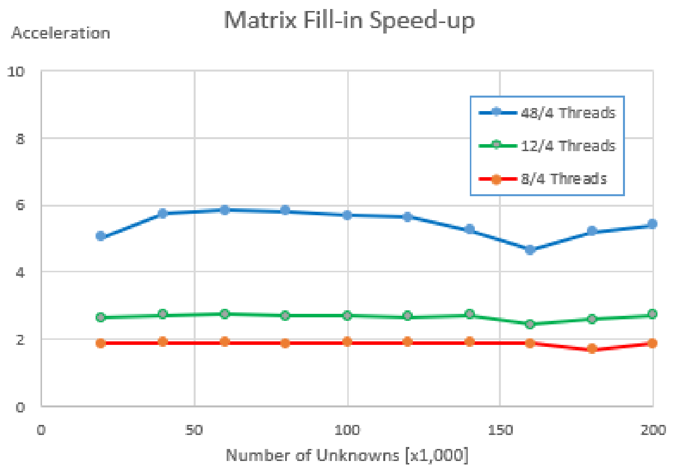

Figure 1. Improving the simulation time for matrix fill-in on multicore PC when the matrix fill-in is done by using 4, 8 and 48 threads

It can be seen, in the diagram shown in Figure 1, that matrix fill-in time is practically halved when 8 threads is used instead of 4, regardless the number of unknowns. When all of the threads are used (24 cores multiplied by 2 with Intel multi-threading), the simulation time is reduced almost 6 times.

Machine used for simulations is 2xCPU Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20 GHz.

Multi-Core CPUs for Multifrequency Problems

As the previous section indicates, for electrically small problems the efficiency of the parallelization on multicore (or multithread) CPUs decreases as the number of threads increases. Typically, such a problem is simulated in a frequency band i.e. at several frequencies. For such a case, a specific feature called “frequency parallel run” can be used with WIPL-D. The underlying idea is very simple – in fact a multi-frequency problem is automatically subdivided into a set of single frequency problems, which are simulated in parallel using a number of processes, as defined by a user. Each simulation uses its own set of CPU threads, which is equal to a total number of threads divided by a number of concurrent simulations. After the simulations of all subprojects are finished, output files are automatically merged into a single output file.

We can illustrate this on a simple example. Let us imagine we need to simulate an antenna over a wide frequency band at 48 points. If an affordable workstation with two 12-core CPUs (Intel hyperthreading enabled) is available for simulation, this actually means that all 48 threads are available for simulation. If one frequency at a time is executed on all 48 threads, the efficiency will not reach a maximum of 100%. However, the frequency parallel run allows us to run 2 frequency points in parallel, each at 24 thread, or 4 frequency points each at 12 threads etc.

As an illustrative example a relatively simple but quite demanding from the simulation perspective direction-finding antenna can be considered. This antenna requires a low frequency simulation, which typically requires computations in double precision.

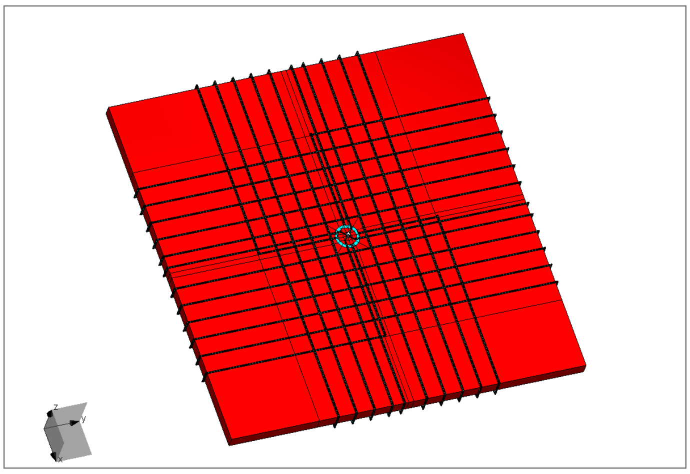
Figure 2. ADF antenna – WIPL-D model

The antenna is next attached to a CAD realistic model of a 19m long helicopter, in the frequency range 190 - 560 kHz. The DF algorithm often requires larger number of frequency points. The simulation involves 10,500 quad mesh elements and only 22,000 unknowns. The matrix fill-in time is dominant, but the multithreading is also important for the matrix inversion.

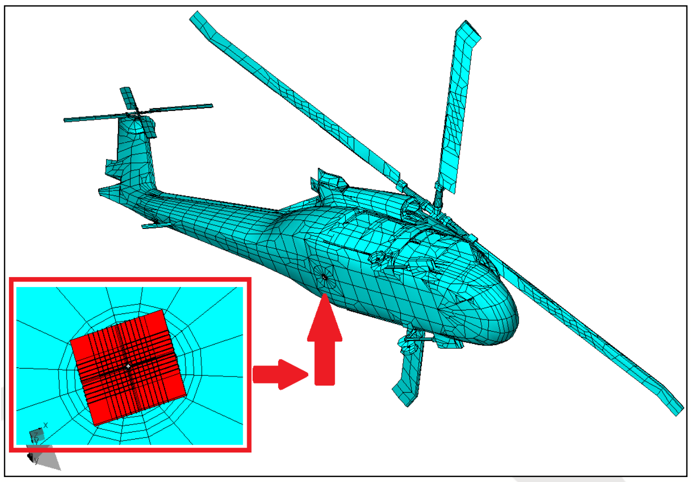
Figure 3. ADF antenna mounted to the helicopter

The simulation is carried out at 48 frequency points, split in different configuration (one frequency at time at 48 thread, two frequencies at time each at 24 threads, etc.). The simulation times per frequency are listed in Table 1. The configuration used is:

Intel® Xeon® Gold 5118 CPU @ 2.30 GHz (2 processors) with 192 GB RAM.

Table 1. Simulation times for the different frequency parallel run settings.

Number of frequency points in parallel	**t [s]**
1	161
2	136.5
4	122.5
8	119.5
16	119.5
24	117.5
48	No sufficient RAM

We can conclude that the multithreading efficiency is not 100% when a single frequency project is executed at such a large number of threads (48). However, even if the project needs to be run at as little as two frequency points, a much better efficiency is achieved (parallel run of two frequency point, each at 24 threads). The efficiency practically reaches 100% even with 4 frequency points and remains perfect if the number of frequency points is further increased. The maximum efficiency should be reached when 48 frequency points are executed in parallel, each at single thread. However, such simulation could not be performed due to the RAM limitation.

Efficiency of GPU Solver

WIPL-D GPU Solver is an add-on tool that exploits high computation power of nVIDIA CUDA™-enabled GPUs to significantly **decrease EM simulation time**. Acceleration by an order of magnitude can be achieved when compared to a CPU multi-threaded solution.

The sophisticated parallelization algorithm results in a highly efficient utilization of an arbitrary number of GPUs. Supported GPUs are nVIDIA's CUDA-enabled GPUs (GeForce, Tesla or Quadro series) with compute capability 2.0 and higher.

The greatest speed-up is achieved in the matrix inversion phase of simulation.

GPU acceleration versus number of unknown coefficients is given for two hardware configurations: a desktop PC equipped with single GPU, and a server equipped with 4 GPUs. In all the simulations used for the following figures, CPU matrix fill-in is used.

The graphs below illustrate the reduction of total simulation time, when GPU inversion replaces CPU inversion, on the configuration with 1 GPU.

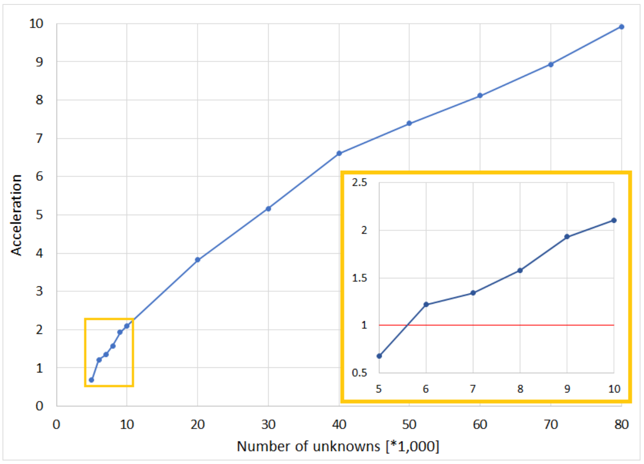

Figure 4. Improving the simulation times on standard desktop PC when the matrix inversion is done at single GPU card (nVIDIA GeForce GTX 1080)

The desktop machine configuration is as follows: Intel® Core™ i7-7700 @ 3.60 GHz, 64 GB of RAM, one nVIDIA GeForce GTX 1080 GPU. GPU acceleration increases with increasing number of unknown coefficients. For 80,000 unknowns, acceleration is ~10 times. Number of unknowns where the GPU solution becomes faster than the CPU solution is ~5,500. Nevertheless, significant acceleration is not noticeable for less than 10,000 unknowns.

Simulation time (when CPU fill-in and GPU inversion are used) for a problem with 10,000 unknowns is 5 seconds, and for a problem with 80,000 unknowns it is 6.8 min.

The simulation times for problems requiring over 80,000 unknowns are less practical at standard desktop PC, even when equipped with a GPU card. Electrically larger problems are more efficiently solved at advanced hardware configurations, involving more CPU cores and multiple GPU cards. The server configuration is as follows: Intel Xeon CPU E5-2660 v2 @2.2 GHz (2 processors), 256 GB of RAM, four nVIDIA GeForce GTX 1080 Ti GPUs and 7 SATA HDDs configured in RAID-0 configuration.

GPU acceleration of total simulation time, for the problems with 80,000 to 200,000 unknowns is shown in Figure 5. Acceleration increases with increase of the number of unknowns.

An abrupt change in acceleration is achieved when the number of unknowns increases from 180,000 to 200,000. The reason for such a behavior is the switching of the matrix inversion to **out-of-core**. In the case of GPU Solver HDDs I/O operations are performed in parallel with GPU calculations, which provides additional acceleration when compared to CPU calculations.

A problem with 100,000 unknowns is solved in 6.4 min, while the simulation time for 200,000 unknowns is 46.5 min.

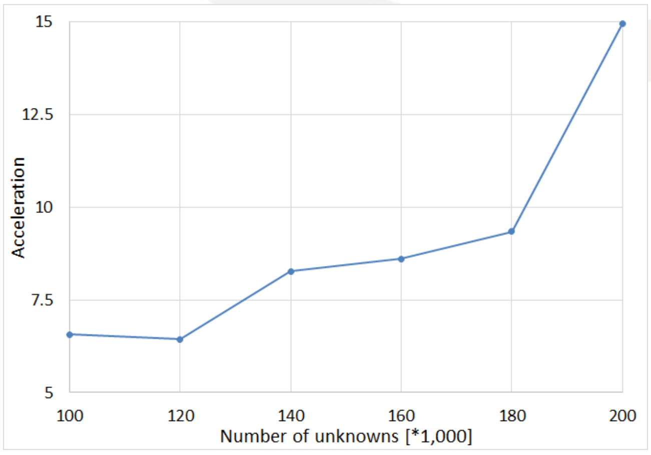

Figure 5. Improving the simulation times on the advanced hardware configuration when the matrix inversion is done at four GPU cards (nVIDIA GeForce GTX 1080)

The next sections will focus on the application of highly efficient GPU solver for the two typical EM problems.

Airplane Model

The model of generic airplane (Fig. 6), is meshed in **WIPL-D Pro CAD** tool. The airplane was scaled in such a way to have approximately 20, 50, 100, 150 and 200 thousand of unknowns.

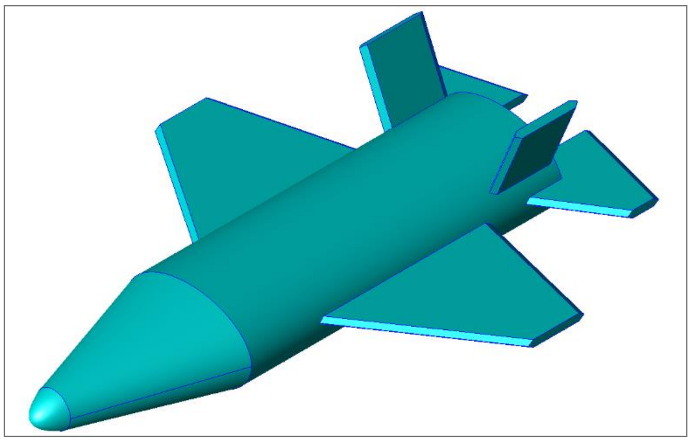

Figure 6. Generic airplane model in WIPL-D Pro CAD.

Fig 7 displays mesh of the generic airplane for 150,000 unknowns.

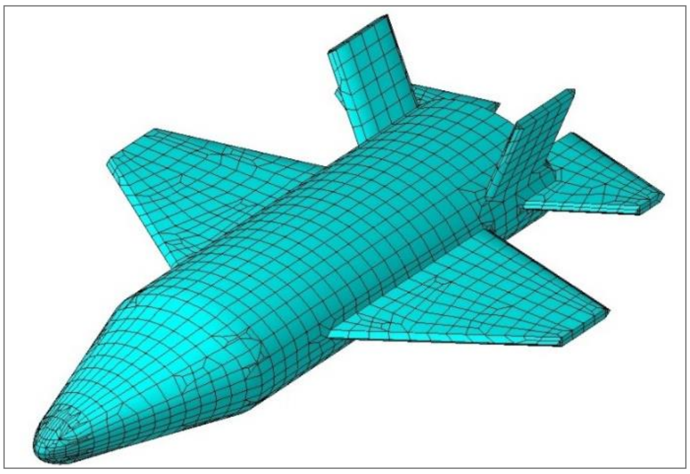
Figure 7. Meshed generic airplane for 150,000 unknowns.

Regular Cubes Model

Cubes presents one of the simplest and widely used canonical structures for EM simulation. All mesh elements are of square shape.

Fig. 8 shows cubes for 20,000 unknowns. Models of 50, 100, 150 and 200 thousand are obtained using copy manipulation.

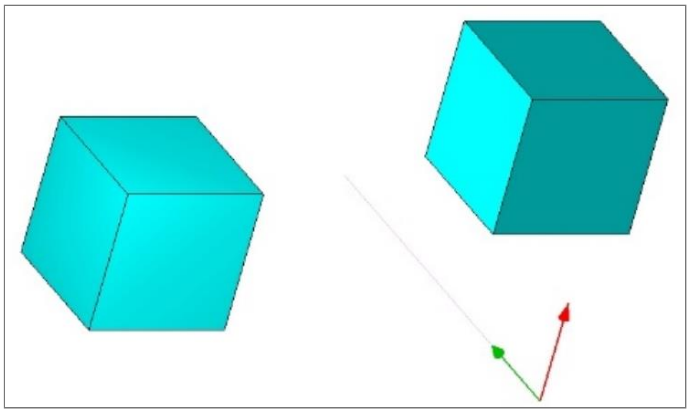
Figure 8. Cubes for 20,000 unknowns.

Airplane and Cubes Simulations

Machine used for simulations is 2xCPU Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20 GHz, 256 GB of RAM, 7 SATA hard disks connected in RAID-0 configuration, and 4xGPU GTX1080 TI.

In order to take full advantage of WIPL-D **high order basis functions (HOBFs)**, CPU parallelization is used for matrix fill-in and GPU solver for the matrix inversion.

Simulation times of generic airplane and cube problem are listed in Table 2 and Table 3, respectively. The data from the tables were used to graphically present the ratio of the matrix fill-in and the matrix inversion time versus a number of unknowns. For the generic airplane, the ratios are presented in Fig. 9, and for the cubes, in Fig. 10.

Table 2. Simulation times of generic airplane model.

Number of Unknowns	Fill-In Time on CPU [s]	Inversion Time on GPU [s]	Total Simulation Time [s]
~20,000	9.4 (35%)	16.8 (63%)	26.9
~50,000	25.6 (23%)	83.6 (75%)	111.1
~100,000	92.0 (22%)	324.2 (77%)	421.6
~150,000	199.2 (18%)	870.9 (81%)	1078.4
~200,000	**296.1 (10%)**	**2,526.0 (83%)**	**3,039.4**

Table 3. Simulation times of cube model.

Number of Unknowns	Fill-In Time on CPU [s]	Inversion Time on GPU [s]	Total Simulation Time [s]
~20,000	4.0 (20%)	15.7 (78 %)	20.2
~50,000	20.6 (17 %)	99.8 (82%)	122.0
~100,000	69.5 (18%)	322.4 (81 %)	396.4
~150,000	**196.2 (14 %)**	**1,026.2 (75 %)**	**1,362.4**
~200,000	**320.0 (10 %)**	**2,553.1 (83 %)**	**3,080.4**

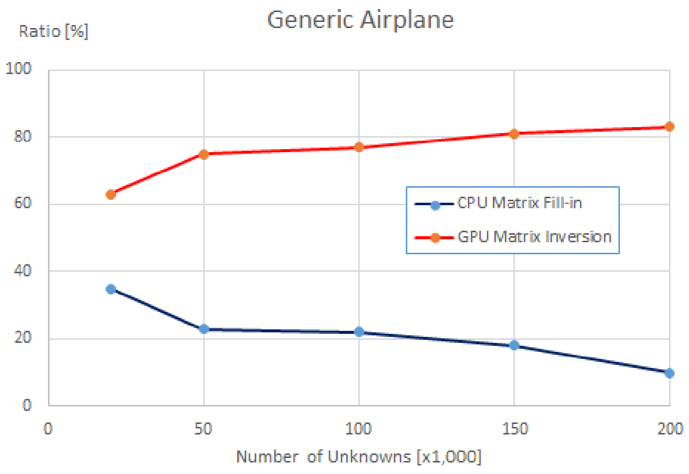

Figure 9. Matrix fill-in – total simulation time ratio (red line) and matrix inversion – total simulation time ratio (blue line) for generic airplane.

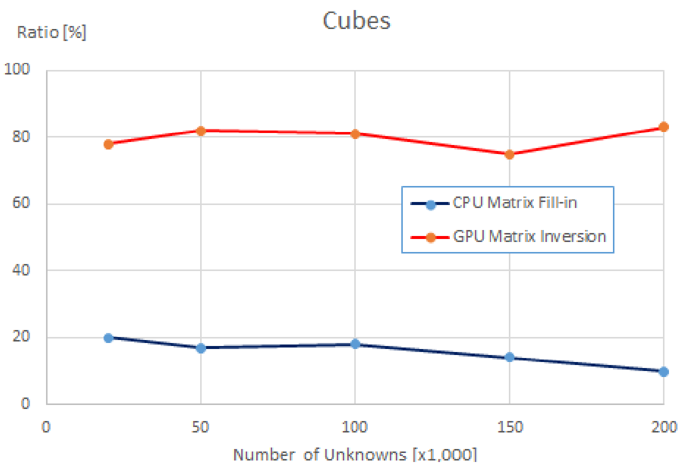

Figure 10. Matrix fill-in – total simulation time ratio (red line) and matrix inversion – total simulation time ratio (blue line) for cube models

It can be concluded from the figures that for electrically large problems, above 50,000 of unknowns, the matrix inversion time requires 80-90% of the total simulation time. The total simulation time is most efficiently reduced by decreasing the most dominant component: the matrix inversion time. That is achieved by using an arbitrary number of GPU cards in the inversion process.

For the simulation of generic airplane with 200,000 unknowns, as well as for the cubes with 150,000 and 200,000 unknowns, the solution is obtained by using the out-of-core solver. The simulation times are bolded in the tables.

It is important to notice that the time required for a matrix inversion for the problem with ~150,000 unknowns, when it is performed in the in-core mode, and out-of-core mode are rather similar. The in-core computations are referred to the airplane model, requiring slightly less than 150,000 unknowns, while the out-of-core computations are referred to the cube model, which requires slightly more than 150,000 unknowns. In the case of out-of-core simulation the total simulation time has been increased mostly by the time used to write the matrix to a hard disk.

While performing the matrix fill-in, the CPU usage is just under the 100%. However, during the matrix inversion stage CPU is used only to synchronize all data transfers between CPU RAM and GPUs VRAM, and CPU RAM and HDDs, and CPU usage is accordingly very low.

Fig. 11 shows CPU usage during the simulation process, on the example of generic airplane model which requires 100,000 unknowns.

Conclusion

Highly efficient CPU and GPU parallelization of WIPL-D numerical kernel is presented in the application note. The matrix fill-in process is fully CPU parallelized, while matrix inversion is fully GPU parallelized. Total simulation time decreases so that a problem with 200,000 unknowns can be simulated in less than one hour.

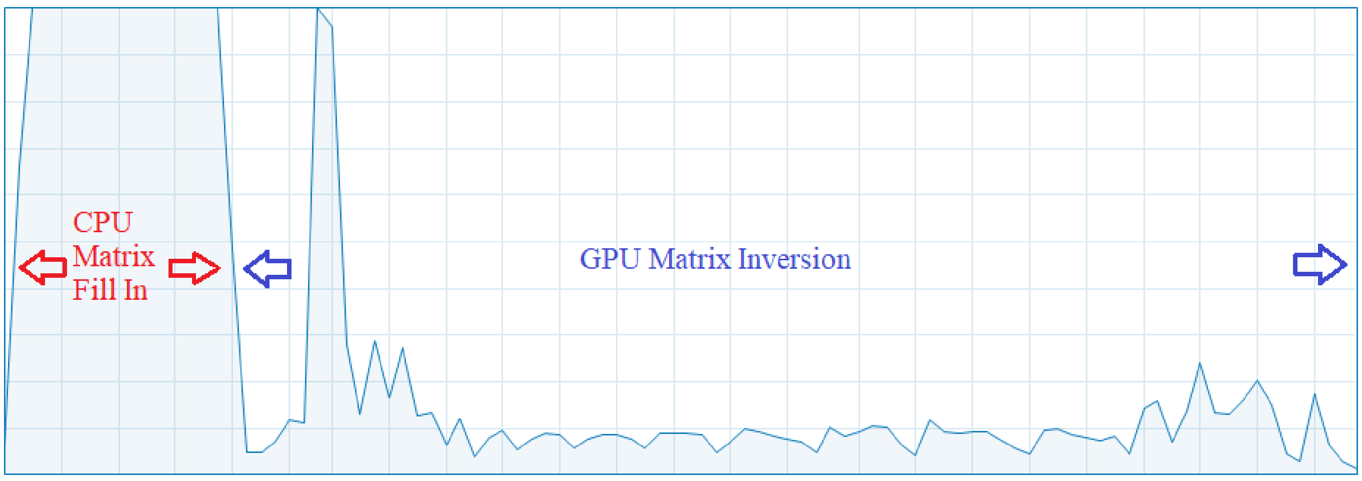

Figure 11. CPU usage during the simulation of generic airplane problem with 100,000 unknowns.