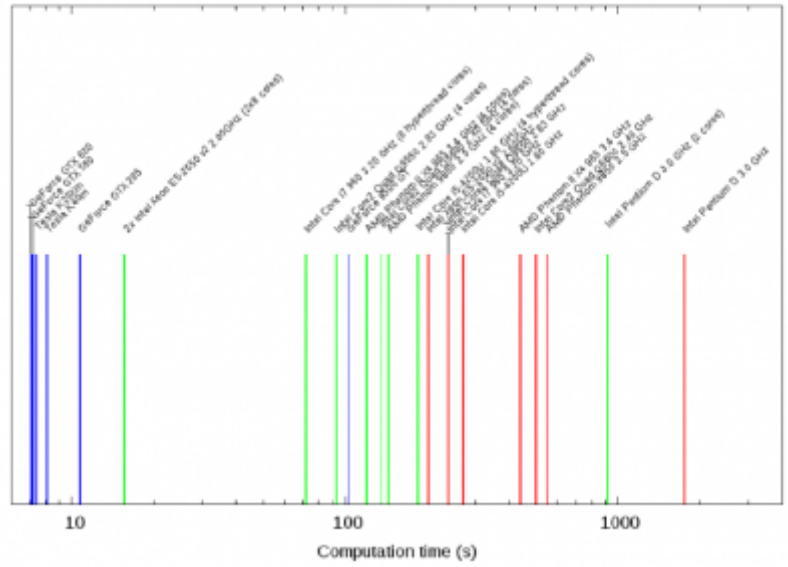
Table of Contents

Graphics cards use

FDTD method is computationally limited by dense discretization that is necessary for its proper operation. For the standard Yee's algorithm (that is a core of FDTD) the stable discretization requires a spatial minimum of $\Delta x \leq \lambda/10$, where λ is propagating electromagnetic wave wavelength. There are some approaches in the literature to overcome these limitations, which however lead to much more complicated algorithms that cannot be simply adapted to all of the large spectrum of FDTD calculations.

As a result, both the capacity of computer memory and the processing speed is a limiting factor for large and long calculations in FDTD. This fact decreases the applicability of FDTD for many purposes. An alternative approach, that is used in Gsvit, is based on using a graphics processing unit (graphics card) to move the calculation from the computer processor (CPU) to the graphics processing unit (GPU) and to speed it up significantly, as shown in the following graph where the time necessary for the same calculation on different cards and processors is listed (note that the x axis is logarithmc).



The capabilities of graphics cards for numerical modeling have increased dramatically over the last few years. Two main producers NVIDIA and ATI have developed drivers for running calculations on their cards, and both memory and processing power of commonly used graphical cards have increased by a large factor. In Gsvit we focus on Nvidia CUDA products (from historical reasons when this was the only graphics card that could be coded in C). Generally, the available memory and computing power of GPUs increases much more rapidly than for personal computers themselves, which is also promising. Moreover, even special supercomputers based on graphical cards can be purchased now, like Nvidia Tesla systems. In these computers there are multiple graphics cards and we want to address also computing on multiple cards therefore.

To use GPU for a calculation is not straightforward, unfortunataly. We cannot simply take a conventional PC executable and run it on GPU. Both data processing and memory model is completely different for GPU and for CPU and the part of the code that should be run on GPU (called kernel) must be written to fulfill these conditions. GPU is equipped by several multiprocessors, consisting of a large number of processors. Many hundreds of threads (kernel calls) grouped in thread blocks can be processed simultaneously on GPU, which is the basis of tremendous speedup that we can achieve. Memory available on GPU can be divided into a global memory - accessible by all the multiprocessors, a shared memory - accessible by processors within one multiprocessor, and a local memory -

accessible by single processor. All the memories are hardware limited (for each type of GPU differently). We refer to Nvidia CUDA developer zone for further details.

To check if the graphics card installed on your computer is suitable for GSvit calculations and if GSvit was installed with graphics card support at all, you can run the solver with parameter “test 0”, e.g. on Linux system (see all the tests for more details):

```
klapetek@pejsek:~/rungsvit/bin> ./gsvit test 0
Running GSvit tests at level 0
System has 32 cores
Gsvit is installed at /home/klapetek/rungsvit/bin
Program is compiled with GPU support
Searching for available GPUs...
Found 5 GPUs
The Properties of the Device with ID 0 are
Device Name             : Tesla K40m
Device Memory Size      : 3489202176
Block Shared memory size: 49152
Max grid size           : 2147483647x65535x65535
Max threads dim         : 1024x1024x64
The Properties of the Device with ID 1 are
Device Name             : Tesla K20Xm
Device Memory Size      : 1744371712
Block Shared memory size: 49152
Max grid size           : 2147483647x65535x65535
Max threads dim         : 1024x1024x64
The Properties of the Device with ID 2 are
Device Name             : Tesla K20Xm
Device Memory Size      : 1744371712
Block Shared memory size: 49152
Max grid size           : 2147483647x65535x65535
Max threads dim         : 1024x1024x64
The Properties of the Device with ID 3 are
Device Name             : Tesla K20Xm
Device Memory Size      : 1744371712
Block Shared memory size: 49152
Max grid size           : 2147483647x65535x65535
Max threads dim         : 1024x1024x64
The Properties of the Device with ID 4 are
Device Name             : Tesla K20Xm
Device Memory Size      : 1744371712
Block Shared memory size: 49152
Max grid size           : 2147483647x65535x65535
Max threads dim         : 1024x1024x64
```

This means that you can use five different GPUs on this system (which does not usually happen on a standard computer). GSvit does not support use of all of them at once, but you can still run up to four different instances on different cards. The basic settings for using the GPU are therefore whether to use it at all (GPU directive in the parameter file) and eventually which one to use (UGPU directive in

the parameter file). GPUs are numbered from 0, exactly the same way as what <tt>gsvit test 0</tt> outputs, so these settings in the parameter file:

```
GPU
1

UGPU
0
```

will run the calculation on the Tesla K40m card on our system.

Even if we try to do our best to have one to one correspondence for everything on CPU and GPU, it can happen that some of the recent algorithms are not yet implemented on GPU. The reference should always be the CPU implementation. We therefore strongly recommend, namely for setting up the new task, to test the task first on CPU, crosscheck it with GPU and then run the various repetitive calculations on GPU to speedup the solution. Some known issues with GPU are listed also later on this page.

As an example you can test the following parameter files, comparing performance of GSvit on different number of CPU cores and on a single GPU on the above mentioned system. The calculation took 8 minutes 37 seconds on a single core, 1 minute 43 seconds on eight cores, and 54 seconds on the first GPU.

Sample parameter file: gpu use.
A 200x200x200 computational domain with a plane wave and dielectric sphere, run on CPU, CPU with multithreading and on GPU. Three different parameter files are provided, one for each case.

Generally, the bigger is the calculation size and problem complexity, the better GPU works comparing to CPU as the effect of administration (data transfers) is smaller. Also, too many unnecessary transfers, e.g. for image outputs at every few steps, degrade the GPU performance.

Known issues and missing algorithms

At present the following algorithms are known to be not yet implemented on GPU or known to have some other troubles and not recommended for use in GPU:

- focused source (key TSFF)
- layered focused source (key LTSFF)

Note that this list might be not complete and we always recommend a cross-check with CPU results when starting a new type of simulation.