AUTOMATYKA • 2010 • Tom 14 • Zeszyt 3/2

Paweł Kapusta*, Michał Majchrowicz*, Robert Banasiak*

# Applying Parallel and Distributed Computing for Image Reconstruction in 3D Electrical Capacitance Tomography

## 1. Introduction

### 1.1. Tomography and challenges

In the three-dimensional Electrical Capacitance Tomography (3D ECT) an approximation for the internal spatial permittivity distribution is computed using the knowledge about the capacitance measurement data acquired from the spatially located set of electrodes on the surface of the investigated object [1, 2]. It is often assumed that the electrostatic fields are simplified to the two-dimensional electrode plane and the image reconstruction is based on 2D assumptions or non-true 3D called as "2.5D" interpolation [1]. However, the actual electrostatic field is spreading out in three dimensions and, therefore, all the volumetric structures inside an industrial process have significant effect on the reconstructed images. Since year 2003 there has been great deal of interest in using true three-dimensional volumetric [3] ECT both in 3D static [1, 2] and in 4D dynamic imaging mode [4]. Electrodes at the surface of the volume apply electrical potential into the surrounding area and emergent charge is measured at the boundary. The measured data is then used in an electrostatic model to determine the dielectric permittivity properties of the object. Nowadays three dimensional ECT imaging technique is to become an important tool in industrial imaging for process monitoring [5].

Electrical capacitance tomography is a relatively mature imaging method in industrial process tomography [6]. The ECT is performing a task of imaging of materials with a contrast in dielectric permittivity by measuring capacitance from a set of electrodes. Applications of ECT include the monitoring of oil-gas flows in pipelines, gas-solids flows in pneumatic conveying and imaging flames in combustion, gravitational flows in silo [5].

3D ECT can be important tool for imaging the volumetric distribution of electrical permittivity. 3D ECT image reconstruction presents a similar inverse problem to electrical

* Politechnika Łódzka, Łódź, Poland

impedance tomography (EIT), which has been extensively studied, so 3D ECT will naturally benefit from progress in 3D EIT image reconstruction. Similarly to EIT, ECT has potential to generate images with high temporal resolution and relatively poor spatial resolution. The spatial resolution is limited as a result of the inherent ill-posedness of the inverse problem and the existence of modeling and measurement errors and limited number of independent measurements in comparison to number of image elements. Among other non-invasive imaging techniques, ECT characterizes much higher temporal resolution than MRI, CT etc. This makes ECT a good candidate for real-time imaging technique which is capable of long term monitoring on fast-varying industrial process applications.

To reach this goal 3D ECT computational subsystem should be able to transform capacitance data into image in fractions of seconds, which is really hard to achieve since typically 3D ECT tomography image can be composed of large number of elements. 3D ECT provides few challenging computational issues that have been reported in the past by many researchers [1, 2]. In comes from the fact most of direct and iterative 3D ECT reconstruction techniques uses matrices processing and manipulation which is too complex to be processed in a real-time using classic (even multi-core chips) CPU power. In this paper a lately emerged GP GPU approach has been considered as an efficient way to obtain a significant speed-up of 3D ECT reconstruction process.

### 1.2. GPGPU computing and distributed systems

GPGPU (*General-Purpose computing on Graphics Processing Units*) is a technique of using graphic cards (GPU – *Graphics Processing Unit*), which normally handles graphics rendering, for computations that are usually handled by processors (CPU – *Central Processing Unit*).

Growing interest in GPU computations started with the inability to clock CPU above certain level, because of the limitations of silicon based transistor technology and constant demand for improvements. Because of this interest in multi-core technology and parallel computing started to emerge, even though sequential programs are more natural. There is a limit though of how fast such program can run. Any change in speed of sequential programs execution is now based on architecture improvements of the CPU rather than higher clocks, but even this approach has limitations.

Parallel programming is not a new idea, though till only recently it was reserved for high performance clusters with many processors, the cost of such solution was extremely high. This changed with the introduction of many-core processors to the mainstream market. GPUs fit well in that trend, even take it to another level. Compared to CPUs, which today have maximum of 2 to 12 cores, GPUs consist, of dozens and even hundreds of smaller, simpler cores designed for high-performance calculations.

CPUs are built and designed to execute single thread no matter how unpredictable, diverse or complicated it may be as fast as possible. For that they require additional resources such as: complicated mechanisms for predicting branches, cache memory and data

prefetch. On the other hand GPUs mostly take care of data computations that are much simpler in their nature and for that reason their execution units, or cores, can be much simpler, which also mean smaller. Thanks to that there can be much more of them on a single chip with numbers reaching dozens or even hundreds. Also, because graphics computations can be easily parallelizable cores in GPU work based on a SIMD (*Single Instruction, Multiple Data*), or more precisely SIMT (*Single Instruction, Multiple Thread*) architecture, where one instruction is applied to a big portion of independent data. This translates into much higher number of operations per second than can be achieved on CPU. The main difference between SIMD and SIMT is, that in former threads can communicate with each other whereas in latter the communication is only done by shared memory and none of them has access to registers of another one. Thanks to this GPUs can run hundreds even thousands of threads at once, compared to only few on CPU.

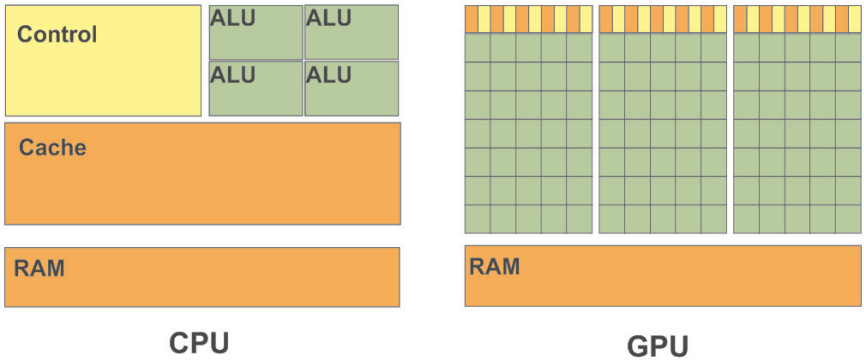


**Fig. 1.** Difference in internal construction between CPU and GPU

Raw power of such solutions is much higher than that of CPUs, by a factor of 10 in double precision (Intel Core i7 965 XE-70 GFLOPS compared to AMD Hemlock XT-928 GFLOPS) and even more in single precision computations. It is possible because of heavy design differences between CPUs and GPUs (Fig. 1).

The reason why GPUs need so much computing power is the dynamically growing game industry – new games become more and more demanding and benefit highly from floating performance,

All this made development of new algorithms possible, using higher computing power of the GPUs. Many computations that were compute heavy and time consuming can now be made in close to real time, with small investments in hardware compared to the cost of achieving the same results using established methods

GPU, although have many advantages, have also many limitations. One of the biggest is that they were designed for parallel performance and high computing power and as such will not do well on sequential tasks, or ones that involve lot of branching. This is still domain of CPUs as they were designed to cope well with such operations.

## 2. Overview of OpenCL and Xgrid technologies

OpenCL (*Open Computing Language*) is the first open, royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and handheld/embedded devices [11]. It was initially developed by Apple Inc., which holds trademark rights, and refined into an initial proposal in collaboration with technical teams at AMD, IBM, Intel, and Nvidia (Fig. 2).

It is a framework for writing programs that can execute across heterogeneous platforms (consisting of CPUs, GPUs, and other processors). It is composed of a programming language (based on C99 with specific extensions) for writing kernels (functions that can be executed on OpenCL capable devices) and APIs for defining and controlling the platform.

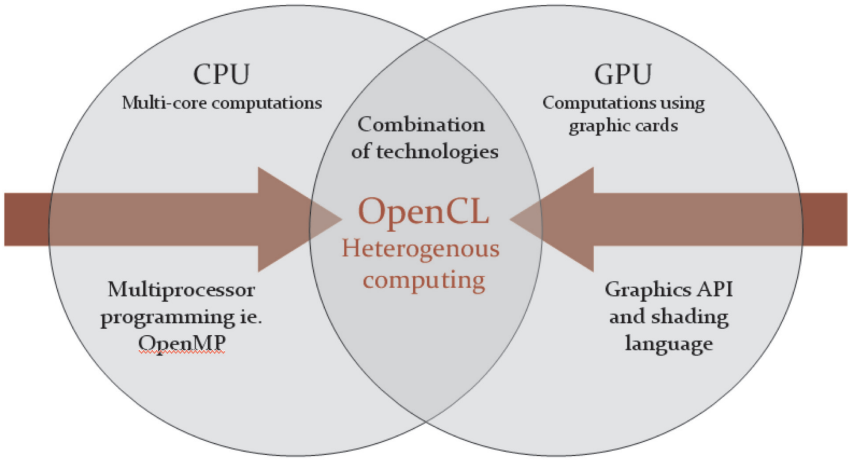


**Fig. 2.** OpenCL as an example of a heterogeneous computing solution

OpenCL can be used in wide range of applications, including embedded, consumer software and even HPC solutions. It is possible because trough low-level, high-performance, portable abstraction. It's also important to notice that thanks to creation of efficient, close-to-the-metal programming interface. OpenCL will be the foundation of a parallel computing ecosystem of platform-independent tools and applications [11].

Thanks to OpenCL the application is able to use Graphical Processing Units for non-graphical computing, as a result OpenCL extends the power of the Graphical Processing Units beyond graphics. OpenCL is analogous to the open industry standards OpenGL and OpenAL, for 3D graphics and computer audio, respectively. It's managed by the non-profit technology consortium – Khronos Group with the participation of many industry-leading companies and institutions including AMD, Apple, IBM, Intel NVIDIA, QNX, Texas Instruments and many others.

Developed solution makes use of OpenCL capable cards in a distributed environment. To simplify the construction of such system Xgrid was used as sample distributed platform to illustrate obstacles and challenges

Xgrid is a program and distributed computing protocol developed by Advanced Computation Group, a subdivision of Apple Inc. It provides administration tools for easy creation of computing clusters that tap into previously unused computational power for calculations that are easily parallelizable. Creating a cluster does not require any additional spending since Xgrid is preinstalled on all Mac OS X 10.4 or later machines.

Xgrid technology (Fig. 3) consists of three components – „client", „controller" and „agent". Process of submitting new tasks is performed in following order:

- The "client" sends jobs to the "controller,
- The "controller" queues the jobs and sends them to the "agent",
- The "agent" runs the job.

The point of the technology is to have several agents working in parallel, each on a different job. It is also possible to have several clients.

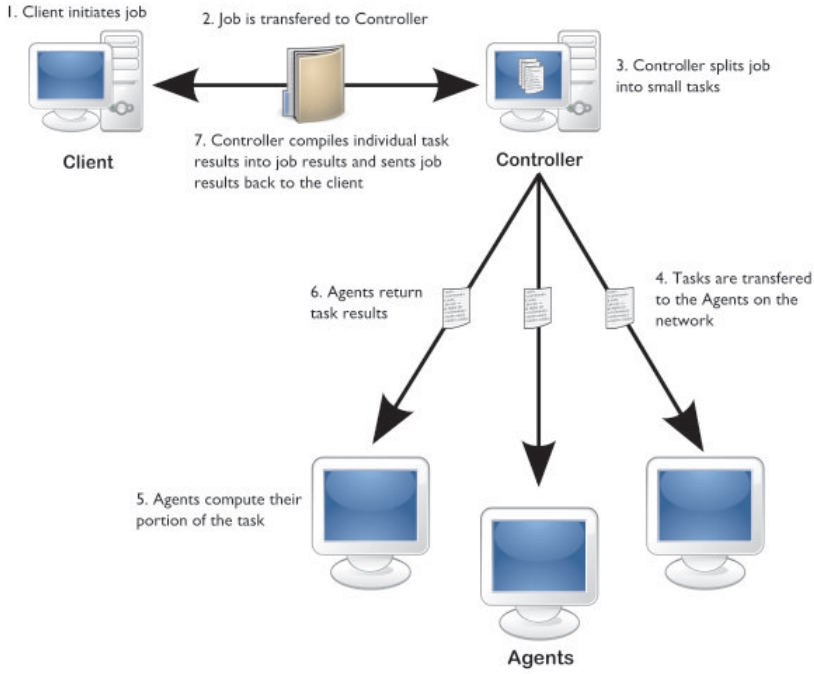


**Fig. 3.** Xgrid network working scheme

All nodes communicate with each other using propriety protocol based on BEEP framework. Every machine that has Mac OS X with Xgrid installed is automatically detected by controller and added to the list of available computers to use for processing tasks.

Every job send to controller is split into smaller packets called tasks. These small packets are transferred to all Xgrid-enabled computers on the network. Every node executes instructions provided by the controller and after finishing the calculations returns the results. After that the controller assembles all task results into job results and returns them to the client.

The protocol consists of three types of messages that are sent between computers combined in a cluster: requests, notifications and replies. Every request requires a reply, notification does not require a reply, and replies are responses to previous requests. All messages are encapsulated into BEEP message (BEEP MSG) and acknowledge by sending an empty reply (RPY) by the recipient. Xgrid messages are encoded into dictionaries of key/value pairs and converted into XML format before being sent using BEEP network.

The architecture of Xgrid is designed around job based system. Every job consists of all instructions and data (files) necessary to perform a calculation. In order to fully exploit Xgrid potential all instructions provided by the controller must be able to be executed simultaneously or asynchronously.

## 3. Developed solution

The mechanism of image creation in electrical capacitance tomography is called the image reconstruction. It boils down to calculating the so-called inverse problem. Solving the inverse problem for capacitive tomography means approximating the spatial distribution of electric permittivity Í from the measured values of C.

Image reconstruction using deterministic methods requires execution of many basic operations of linear algebra, such as transposition, multiplication, addition and subtraction. Matrix calculations are characterized by the fact that their computational complexity is high, especially for a large number of elements. Moreover, the time needed to perform them normally does not grow linearly with the increase in the number of elements. For example, in the case of multiplication, which in itself is an algorithm of O ($n^3$) class it can be simplified to assume that n-fold increase in the multiplied matrix dimensions will $n^3$ fold increase the execution time.

In ECT tomography images are often reconstructed using FEM (*Finite Elements Method*) which involves the use of meshes and usually they consist of a large number of elements. In this case, the process of image reconstruction cannot be achievedon-line with the currently available CPU capacity.

With a large number of elements calculation time necessary to display an image may take up to several hours, which not only prevents the application currently in tomography but also limits the development and testing processes of reconstruction algorithms and hardware.

Matrix operations are characterized by the fact, that they can be parallelized, which means that computations can be performed simultaneously on different parts of matrices. Thanks to this, with the application of proper hardware and algorithms, a significant speed up can be achieved.

OpenCL platform is natively parallel and can fully utilize the multi-core architecture of graphics cards, which in turn allows for even cheapest hardware to outperform CPUs in tomography specific computations.

Every hardware solution has limitations, single graphic cards are faster than single CPUs, nevertheless using multiple GPUs is a major leap in computations and distributing calculation to multiple computers equipped with OpenCL capable cards allows almost unlimited parallelization. Image reconstruction in ECT tomography, in many cases, requires multiplying matrices that (in binary format) are 200 MB or more in size, with small matrices that are the output from capacity sensors. Main problem of real-time image reconstruction are delays caused by computation, nevertheless data sending is also a very important issue since it introduces additional delays and hinders the whole process. In many situations transferring large amounts of data over a LAN may have huge impact on performance and slow down the calculation process and as a result it may be impossible to achieve on-line (many frames per second) image reconstruction despite using high computing power. Moreover these big matrices (sensitivity matrices) do not change, the only thing that is being replaced are small matrices provided by the ECT system (measurements). Furthermore, for performing calculation instant matrices exchange is required in each iteration of current algorithm. It should also be noted that Xgrid uses base64 format for transferring binary files which introduces overhead and increases the size by 30%. Moreover TCP/IP randomness makes it very hard to predict how long it will take to transfer all information needed to perform computation.

Time to load matrices, after successful transfer, form the disk has to be also taken into consideration since it might take as long as up to 6–7 seconds with matrices of size of 200 MB. Unfortunately all data transferred to the clients is lost after calculation is finished. If computations are performed in a relatively short time then the amount of data that has to be transferred can have a significant impact on performance. As it was stated before Xgrid system does not provide any way of storing data for later usage. In order to achieve such functionality we have created a small program that copies all files from local directory to "/tmp/xgridagent_data" on remote machines. It is also capable of creating this directory and also listing, computing md5 check-sum and deleting all files from it. Our research also revealed that Xgrid has problems with transferring data that are bigger than 50 MB but thanks to presented solution it is possible to split the file into many parts and transfer every part to target machine separately. We have also modified source code of xgridagent-java, a program that can be run on machines with Windows or Linux operating systems installed, that is capable of connecting to Xgrid network, accepting and running tasks. As a result every computer running one of mentioned operating system can be used to accelerate the calculations. The combination of tools require creating tasks in form of C programs that can be compiled on any of mentioned systems and providing three binary files one without the extension (for Mac OS X) with „.exe" extension (for Windows) and with „.lin" extension (for Linux). After creating such environment we were able to easily distribute OpenCL executables that performed LBP (*Linear Back Projection*) algorithm and, as a result, achieve significant acceleration of image reconstruction process.

## 4. Tests

For testing purposes a simple LBP image reconstruction algorithm was used, which is given by following equation:

$$\boldsymbol{\varepsilon} = \mathbf{S} * \mathbf{C} \tag{1}$$

where:

$\boldsymbol{\varepsilon}$ – Image vector,
$\mathbf{S}$ – Sensitivity matrix,
$\mathbf{C}$ – Capacitance measurements vector.

More compute-heavy algorithms were simulated by invoking LBP many times and treating it as one full iteration of more complicated algorithm. This was done to simplify the process of parallelization and distributing data and computations between systems and to remove many variables that are currently beyond the scope of this article.

- Tests were conducted using following configurations:
- Computations on one processor core (Intel Core 2 Duo 2.53 GHz)
- Computations on two processor cores, inside one physical processor (Intel Core 2 Duo 2.53 GHz)
- Computations on four cores (two physical processors) connected by Xgrid (2x Intel Core 2 Duo 2.53 GHz)
- Computations on Nvidia GeForce 9400M graphic card with 256 MB RAM
- Computations on two, not identical, GPUs simultaneously, inside one computer (Nvidia GeForce 9400M and 9600M GT cards, both with 256 MB RAM)
- Computations on three graphic cards, two in one system (9400M and 9600M GT) connected with other one using Xgrid (GeForce 9400 M)

Results are presented in Table 1 and Table 2, as well as using graphical representation on Figure 4 and 5.

**Table 1**
Reconstruction time using LBP algorithm on many test systems

| Problem Size | CPU (single core) [s] | CPU (dual core) [s] | Xgrid CPU (four cores) [s] | GPU [s] | Dual GPU [s] | Dual GPU (optimized) [s] | Xgrid 3xGPU [s] |
|---|---|---|---|---|---|---|---|
| 4992 | 0.747 | 0.403 | 1.001 | 0.020 | 0.057 | 0.015 | 0.811 |
| 9984 | 1.312 | 0.604 | 1.156 | 0.040 | 0.029 | 0.02 | 0.870 |
| 19968 | 2.490 | 1.195 | 1.498 | 0.051 | 0.058 | 0.04 | 0.930 |
| 39936 | 5.036 | 2.538 | 2.297 | 0.102 | 0.116 | 0.081 | 1.089 |
| 60000 | 7.330 | 3.023 | 2.983 | 0.154 | 0.175 | 0.089 | 1.265 |

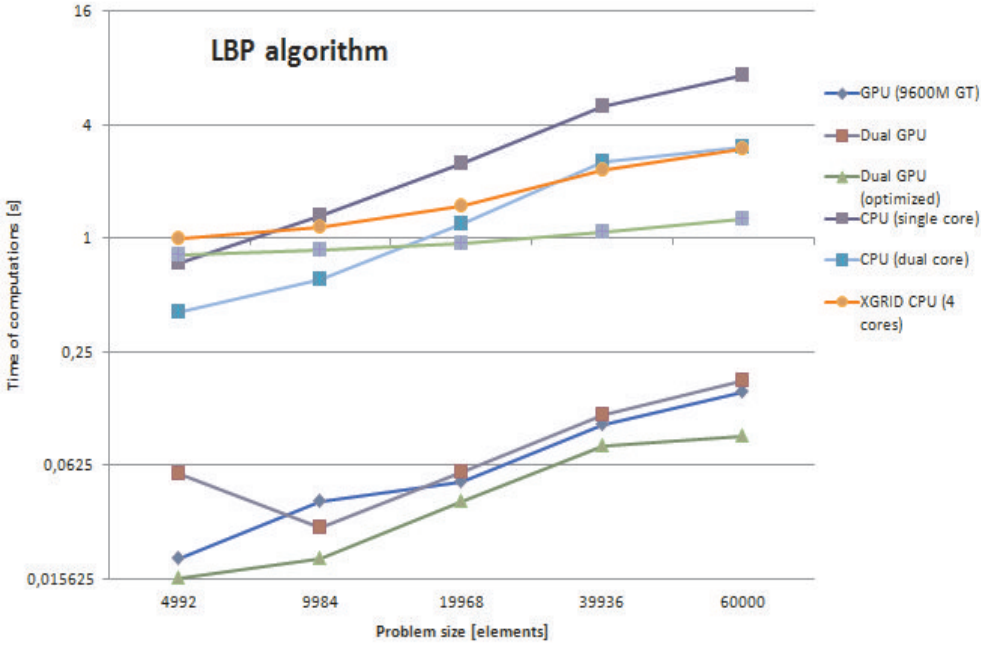


**Fig. 4.** Graphical representation of LBP algorithm's execution time

Figure 4 shows the time of computing LBP algorithm for different problem sizes. As can be seen form it, GPU computations, even not optimized are much faster than ones performed on CPU. Even when using dual core processor GPU implementation, using only one graphics card, is more than 19 times faster, for the same problem size. With the optimized dual GPU computations, the results are even better. This is not the case though for algorithms which are not optimized, as can be seen from the results of computations using two graphics processors, but without proper load balancing and data distribution between them. Figure 4 clearly shows that in such case computations are performed even slower than using single graphics card, even though the combined theoretical power is higher.

Interesting conclusions can be drawn from analyzing the distributed computation of LBP algorithm. For small problem size local execution on a dual core CPU is faster than both Xgrid computations, either using two processors or even 3 GPUs and optimized load balancing algorithms. This is all due to overhead that Xgrid introduces, both in data transfer and remote execution of programs that are needed for the distributed system to work correctly. Only using bigger problem sizes, when the computations time becomes the dominant factor a speed up can be seen. This shows that distributed computing is usually not a good approach for small problem cases.

**Table 2**
Execution time of a simulated, complex reconstruction algorithm on many test systems

| Problem Size | CPU (single core) [s] | CPU (dual core) [s] | Xgrid CPU (four cores) [s] | GPU [s] | Dual GPU [s] | Dual GPU (optimized) [s] | Xgrid 3xGPU [s] |
|---|---|---|---|---|---|---|---|
| 4992 | 9.387 | 5.069 | 3.353 | 0.255 | 0.333 | 0.2 | 0.972 |
| 9984 | 18.820 | 8.657 | 5.196 | 0.509 | 0.667 | 0.398 | 1.125 |
| 19968 | 37.584 | 18.040 | 9.918 | 1.019 | 1.330 | 0.796 | 1.431 |
| 39936 | 70.998 | 35.783 | 18.912 | 2.041 | 2.610 | 1.348 | 2.042 |
| 60000 | 103.52 | 51.139 | 26.756 | 3.429 | 3.667 | 2.251 | 2.851 |

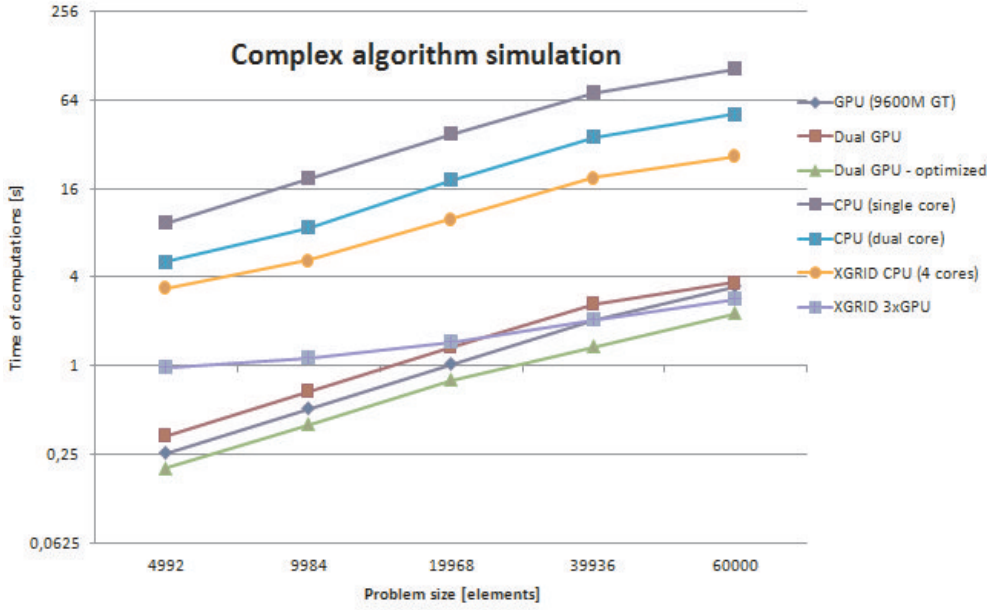


**Fig. 5.** Graphical representation of execution time using complex reconstruction algorithm

Figure 5 shows the time of computations of more complex image reconstruction algorithm, simulated by invoking LBP twenty times. As can be seen on Figure 5 not optimized version of algorithm is not only slower than optimized but also slower than computations performed only on a single 9600M GT graphics processor. These results show clearly how optimization and good load balancing algorithms are important in case of OpenCL. Moreover optimized version is almost 50 times faster than the same computation on CPU. Using

2 cores speeds up the process, although it can still 8 times slower than simple Dual GPU version and nearly 26 times slower than optimized one. Interestingly Xgrid computation using four CPU cores is always faster than single and dual core CPU versions. It seems that transport and Xgrid system itself doesn't cause enough overhead to slow down the process enough to have noticeable impact on computations time. Nevertheless in case of OpenCL Xgrid version we can clearly see that such overhead exists and can really slow down the whole process if the calculations weren't complex enough. Furthermore data shows that this overhead does not change dramatically with the increase in problem size. In case of computations using 3 GPUs and for maximum problem size reconstruction time is approaching Dual GPU (optimized) solution. It can be seen, that the Xgrid 3 GPU solution will probably gain advantage over local execution. This approach can be used even with larger numbers of devices and therefore achieve higher computation power than any other solution.

## 5. Conclusion

For many years people got used to performing computations on single core CPU's. Nowadays multicore CPUs and GPUs calculations are becoming more and more popular. Developing OpenCL standard was a leap that allowed everyone to create algorithms and applications that are platform and device independent. Unfortunately everything has its limits and although OpenCL supports in some way performing calculations on multiple devices it doesn't support performing it on multiple machines. As was outlined in this article such computational environment can achieve very high computational power. Since Xgrid isn't optimized for performing calculations at high speeds new solution has to be developed in the future to allow for future improvements.

## References

[1] Wajman R., Banasiak R., Mazurkiewicz Ł., Dyakowski T., Sankowski D., *Spatial imaging with 3D capacitance measurements*. Measurement Science and Technology, vol. 17, No. 8, August 2006, 2006, 2113–2118.

[2] Soleimani M., *Three-dimensional electrical capacitance tomography imaging*. Insight, Non-Destructive Testing and Condition Monitoring, vol. 48, No. 10, 2006, 613–617.

[3] Warsito W., Fan L-S., *Development of 3-Dimensional Electrical Capacitance Tomography Based on Neural Network Multi-criterion Optimization Image Reconstruction*. Proc. of 3rd World Congress on Industrial Process Tomography (Banff), 2003, 942–947.

[4] Soleimani M., Mitchell C.N., Banasiak R., Wajman R., Adler A., *Four-dimensional electrical capacitance tomography imaging using experimental data*. Progress in Electromagnetics Research-PIER, 90, 2009, 171–186.

[5] Romanowski A., Grudzien K., Banasiak R., Williams R.A., Sankowski D., *Hopper Flow Measurement Data Visualization: Developments Towards 3D*. Proc. of 5th World Congress on Industrial Process Tomography, Bergen, Norway, 2006.

[6] Yang WQ., Peng L., *Image reconstruction algorithms for electrical capacitance tomography*. Institute of Physics Publishing, 2002.
[7] White M.K., *Apple Training Series: Mac OS X Support Essentials*. Berkley, Peachpit Press, 2008.
[8] Kshemkalyani D.A., *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, Cambridge, 2008.
[9] Kirk B.D., *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, San Francisco, 2010.
[10] Herlihy M., *The Art of Multiprocessor Programming*. Morgan Kaufmann, San Francisco, 2008.
[11] http://www.khronos.com/opencl/.