Journal of Multidisciplinary Engineering Science and Technology (JMEST)
ISSN: 2458-9403
Vol. 7 Issue 12, December - 2020

KYAMOS Software – CUDA aware MPI Solver for Poisson equation

Antonis P. Papadakis*, Aimilios Ioannou and Wasif Almady
KYAMOS LTD, 37 Polyneikis Street, Strovolos, 2047, Nicosia, Cyprus
*ceo@kyamosmultiphysics.com

***Abstract*—KYAMOS LTD is a newly founded startup operating in the Computer Aided Engineering industry and its mission is to provide cloud computing software for scientists, engineers and non-engineers using cloud-based, InfiniBand GPU computing. KYAMOS utilizes built-in, state-of-the-art software in an attempt to attract new users, or existing users of either open source or proprietary commercial software which realize that, much cheaper and faster with better accuracy simulations are possible, using state of the art algorithms. In this paper, we first conduct a brief introduction to computational fluid dynamics simulations, and discuss the different types of differential equations used in the macroscopic world. Then we discuss the theory behind adaptive mesh techniques, its advantages and disadvantages and present an algorithm that produces nearly ideal mesh element qualities. Then we describe the finite element Galerkin formulation of the Poisson equation using an iterative Conjugate Gradient method and its parallelization into CUDA aware MPI. Finally, we conduct validation results for the Poisson equation in uniform geometries and test the scaling of our computer software to multiple threads and GPUs.**

Keywords—KYAMOS software, High Performance Computing, CUDA, Multiphysics, GPU;

I. INTRODUCTION

Multiphysics simulations deal with the usage of computer simulations to mimic the real behavior of physics phenomena and their application to engineering systems that aid in the design process. Thus, save costs by avoiding expensive experimental testing and by providing detailed insight into the design process. Multiphysics simulations are used by engineers, non-engineers, scientists, researchers and consultancy firms.

The multiphysics simulations industry exists for more than 25 years and a number of both free and proprietary software have been released through the years. It is a mature industry which is highly competitive and very important industry in making the world a better place by designing cheaper systems, more environmentally friendly and optimized for security and aesthetic purposes using the available software.

Generally, to simulate multiphysics phenomena in the macroscopic world, in most of the cases, we revert to the solution of different types of partial differential equations of 2nd order, which generally fall within three main categories, the hyperbolic, elliptic and parabolic types. Even though there are some analytical solutions of the above equations, many times their solution is not linear and can only be solved using computer simulations.

The hyperbolic equations of order n are partial differential equations that have a well-posed, initial-value problem for the first n-1 derivatives. Many of the differential equations of mechanics are hyperbolic. The hyperbolic equation solutions behave like a wave and an initial perturbation, needs time to be propagated within the domain. Examples of hyperbolic application include fluid flows, aerodynamic flows, contaminants through a porous media and atmospheric flows [1]. When compared to parabolic and elliptic equations, the solutions to hyperbolic equations tend to be more complex due to the fact that the phenomena are more complex.

A parabolic partial differential equation is widely used in the simulation of heat conduction and other diffusion equations such as the Burger's equation. It is an initial-boundary condition problem and it is transient.

Examples of elliptic equations of 2nd order which are non-trivial are the Laplace and Poisson equations. It describes phenomena that do not change with time, where the solution is defined by the boundary conditions, such as Dirichlet and Neumann boundary conditions. The solution of elliptic equations involves the solution of a system of linear equations which can be solved using direct solution methods such as Gaussian elimination, and indirect simulations such as the Conjugate Gradient and the Gauss-Seidel methods.

For the accurate simulation of computational fluid simulations, a mesh decomposition of the geometrical domain is necessary. The element quality of the decomposed individual elements has a direct impact on the quality, as well as the stability of the results. Since we are tackling to solve the most challenging and complicated problems in the industry, which usually involves non-uniform geometries, we have chosen as basis elements, triangles in two and tetrahedrals in three-dimensions, that can capture non-uniform geometries, efficiently. However, most of the times, the quality of the mesh, which is a measure of the geometrical uniformity is very bad and needs to be corrected. On top of that, the mesh sizing has a major impact on the computational times and generally a

finer resolution on the mesh will result in more accurate, converged results, however; at the same time, it will result in increased computational times, most of the times, too much too handle, with nowadays computer capabilities. Hence, a new innovative method of adaptive meshing is proposed, where mesh resolution changes dynamically and automatically, by meshing at the appropriate places where the solution changes, and coarsens the mesh at places that high resolution is not necessary. In the next section, an adaptive mesh algorithm, which also includes element quality improvement algorithm, is presented.

II. ADAPTIVE MESHING

A. Introduction

In the majority of the methods used for the solution of partial differential equations, a mesh exists which is used to define the geometry of the problem and the necessary resolution, varying from uniform to non-uniform and in many dimensions. One-dimensional space is separated by nodes and edges, two-dimensional space is separated by equilaterals, triangles or polysurfaces, and three-dimensional space is separated by blocks and polyhedral such as tetrahedral, hexahedral and so on. Adaptive meshing utilizes an error indicator to instruct the simulation to automatically pose extra refinement in specific places and coarsen other areas, such that one is able to capture the physics with the correct resolution where it matters, and avoid tedious simulations of no significance. Even though as a principle sounds an excellent idea, it has some major pitfalls which make its usage not always so efficient and fundamental. These drawbacks emanate from the extra computing time and effort to generate the various meshes and to interpolate the results from one mesh to the other. Specifically, you will need error indicators that will dictate the level of adaptation and this is not always trivial to catch, since it depends on the physics involved and can be different for each specific problem, and also a lot of time is spent on the interpolation between meshes. This causes two issues with the first being that it is very time consuming to interpolate from on mesh to the other, since you need to identify each element of one mesh in which element of the other mesh it resides and apply interpolation. This is usually done more efficiently with a reference to a regular grid, where both meshes can refer to, such that to make this recognition easier. Secondly, during the interpolation process, numerical diffusion is introduced in the results, if the resolution between the meshes is any different. This introduces a major drawback in the accuracy of the results, which can only be avoided, if there is overlapping between the interest regions of the two meshes, hence increasing the mesh necessary because of the overlapping necessity. Finally, every time the mesh is changed, all the metrics that have already being setup and calculated for a specific problem such as centroids, midfaces, normal vectors, any already setup sparse structures, need to be calculated from the beginning, which adds another extra and significant burden to the simulations. If one decides to re-mesh as late as possible, this could help things out. Also, one needs to decide when to re-mesh appropriately. Since one cannot reside too often because it is expensive, one can lose track of the accuracy of the simulations.

Finally, in adaptive meshing, it is also important to be able to generate good quality meshes that have no acute angles that will cause instability in the formulations. Both in finite elements and finite volumes, where non-uniform elements exist, this is an extra issue that can be tackled using h and r-refinement, however this process is also time costly as well.

In the case of a multiphysics model, for example the simultaneous solution of the Poisson equation with the charge continuity equations in plasma applications, one needs to take into consideration error estimators for each of the different partial differential equations to be numerically analyzed, hence the error indicator must be normalized, usually from 0 to 1, such that there is universal reference of the mesh refinement that needs to be performed to satisfy all partial differential equations needs for accurate and efficient simulations. In case that one of the variables of a partial differential equation needs re-meshing, all the partial differential equations need to also refine, even if this is not necessary.

B. Adaptive mesh theory

There are four main methods used for mesh adaptation [2-5], which are the h-refinement/coarsening, the r-refinement, the p-refinement and the m-refinement [6]. In the h-refinement/coarsening, addition/removal of mesh nodes, as well as edge swapping techniques are used (for triangular meshes for example, the longest edge bisection, and the regular split techniques [7]), resulting in an overall increase/decrease in the number of unknowns of the existing mesh. In the r-refinement, the total number of existing nodes remains the same, with the only difference that the mesh nodes are relocated to achieve optimum element quality within a fixed number of degrees of freedom. In the r-refinement method, the geometry of the domain, the structure of the mesh, and the identification and labeling of nodes and elements must remain the same, even though mesh nodes are reallocated. In the p-refinement, a fixed mesh is used, and the polynomial degree of the ansatz space is increased by employing higher order numerical schemes to improve local accuracy, as well as to approximate troublesome derivatives. Finally, in the m-refinement method, depending on the behavior of the approximated solution, one switches into a different physical model by solving different differential equations to achieve a better solution for the problem.

The leading author of this paper in a previous paper [8] has utilized this adaptive mesh algorithm in a specific plasma application to simulate real physical problems of plasmas. Here, the actual algorithm that has been applied for mesh quality improvement is presented.

C. Mesh Quality

Here we present in short, an adaptive mesh generator and mesh quality algorithm that has proven to work efficiently in the simulation of plasma related problems in two-dimensional cartesian and cylindrical axisymmetric coordinates, and could be extended in three dimensions with similar principles into tetrahedral element meshes as well.

1) h-refinement techniques

These techniques involve node addition and removal, and edge swapping and removal both on the boundary and non-boundary nodes such that the optimum connectivity is achieved for the nodes, which is a non-boundary node to be connected to six other nodes and a boundary node to be connected to four nodes. The authors have devised a nine-step algorithm that has the capability to produce ideal element connectivity in all the mesh elements.

2) r-refinement techniques

Mesh smoothing techniques improve the mesh quality by relocating the vertices of the mesh. There are many algorithms and techniques both for simple and complex geometric domains. Examples of techniques dealing with simple geometric domains are the optimal Delaunay triangulation (ODT) proposed by Chen [9], whereas for complex domains, one has the Laplacian smoothing [10], the smart Laplacian smoothing [11], the Centroidal Voronoi Tessellation (CVT)-Based Smoothing [12], the Optimal Delaunay Triangulation (ODT)-Based Smoothing [13], the Angle-Based Smoothing [14] and the Well-Centered Triangulation (WCT) Smoothing [15]. A detailed comprehensive review of the various smoothing techniques, as well as extensions of the above algorithms can be found in the work of Erten et al. [16]. In this paper, the Laplacian smoothing by relocating the vertices at the center of the polygon is used which is the simplest method of smoothing, with the drawback that inverted elements may be generated. To avoid the generation of inverted elements, the Smart Laplacian Smoothing is utilized, where the point is relocated only when there is an improvement in the overall quality of the mesh. This adds an extra computational cost which is not considerable, and guarantees that no inverted elements will be created.

3) Implementation procedure

Some features of the mesh refinement procedure are based on the ones used by Lohner [17, 18] and Berger et al. [19-21]. However, new features are exploited when compared to the above work with regard to the mesh improvement procedure by including different refinement and mesh quality treatment techniques. Furthermore, the coarsening of the mesh is achieved differently by exploiting three different meshes at any time. These are: (a) the adapted initial mesh before remeshing, (b) the reference coarse mesh which is the same throughout the simulation, and (c) the final adapted mesh after remeshing to be used in the simulation. It must be pointed out that the reference coarse mesh is not used in the actual simulation, since it is an intermediate tool used to decide on the amount of mesh refinement and to achieve coarsening of the mesh. The usage of the above three meshes allows one to perform coarsening of the mesh and at the same time to minimize interpolation errors by interpolating between two meshes which do not differ greatly. The main steps used to implement the adaptive mesh algorithm can be found in [8].

The authors have developed an element quality improvement algorithm that guarantees in two-dimensions the improvement of the quality of any existing mesh to nearly ideal standards for uniform and non-uniform geometric domains. The element quality improvement algorithm uses a combination of edge swaps, node reallocation, and node addition/removal methods, such that the quality of existing bad quality meshes is greatly improved. Typical bad element qualities can be even of 0.5 or less, which are expected to cause spurious oscillations and in the long run instability in the results. To achieve this, as a first step, the author has developed methods that will ensure that most interior nodes are connected to six nodes, and a small number of them to five or seven nodes. As far as the boundary nodes are concerned, the boundary nodes are treated in such a way to ensure that they are all connected to four nodes, unless their curvature varies abruptly, where they are allowed to be connected to any number other than four nodes. The first step of improving the connectivity of an existing bad quality mesh is implemented using nine cases of the h-refinement/coarsening techniques developed by the author. Bad quality meshes are defined as meshes in which one or more triangular elements have large difference in the length of its three sides, thereby largely deviating from the ideal case of an equilateral triangle.

4) Interpolation between meshes tool

For an adaptive mesh algorithm to be efficient, results need to be interpolated often from one mesh to the other. It is imperative that this method is done as quickly and accurately as possible, since wrong interpolation can cause instabilities on the results. Furthermore, long times consumed to perform such interpolation between meshes, can make the adaptive mesh developed disadvantageous, when compared to non-adaptive mesh techniques.

If one needs to interpolate the results of Mesh1 to Mesh2, first one needs to identify the element of Mesh2 (in this case triangular elements) at which each node of Mesh1 resides. One way to achieve this is to use a non-adaptive interpolation technique, which is to pass through each element of Mesh2, and check whether each node of Mesh1 resides in that triangle. This results in a number of operations equal to the number of nodes of Mesh1 multiplied by the elements of Mesh2. In highly demanding simulations, the above method can be highly time consuming.

Alternative general methods of finding the point location in triangles include the Jump and Walk [22], and the Quad-tree data structure methods [23]. The Jump and Walk method picks a small group of sample points within the mesh and starts to walk from the sample point which is the closest to the requested

point, until the triangle containing the location point is found. The Quad-tree data structure method separates the two-dimensional space into consecutive four quadrants or regions by using a tree data structure in which each internal node has exactly four children.

The author uses an alternative simplistic approach by using an adaptive interpolation technique. Since one knows beforehand the geometric domain in which the mesh resides, a square box is defined that includes the above geometric domain of the problem. This square box is subdivided into many square boxes in both directions, with each square box numbered sequentially from left to right and bottom to top. This division is performed only once and the uniform division and labeling of the geometric domain allows one to use it as a reference domain to find approximately nodes of the two meshes that reside in similar geometric domains. This is achieved by passing from each node of Mesh1, and registering in which box each node of Mesh1 resides. Also, by passing from each element of Mesh2, and then through each of their nodes, one can also identify at which square box each triangle of Mesh2 approximately resides. Now instead of having to search for each node of Mesh1 through all the elements of Mesh2, one only needs to search through the elements which are within or close to that box. This reduces the amount of operations significantly. However, in the above procedure, there will be boxes that will register no triangular elements because it may be the case that one element covers one whole square box. So, it is not enough to just search through that box, but also in the surrounding boxes as well, until a match is found, where a node of Mesh1 resides in a triangle of Mesh2. The algorithm developed by the author of this paper, that discovers which near boxes to search in, is basically an algorithm that takes anticlockwise each box around the initial box that the node of Mesh1 resides, until it finds a match between node and element. The way to decide whether a node resides within an element is done by using the barycentric coordinates. The barycentric coordinates of a node p relative to a triangle are found by the cross product between one vector joining one of the triangle's vertices q and point p, and an edge vector of the triangle starting from the above triangle vertex q, divided by the total area of the triangle.

If a node is situated within a triangle, by calculating its barycentric coordinates, they should add up to one. If the node is situated outside the triangle, the barycentric coordinates always add up to a number greater than one. This would be a straight forward task, if computers were holding numbers accurately. However, due to precision problems, the coordinates of the nodes are not exact; thereby values just greater than the value of one will appear. This inevitably creates problems, since there is no way of identifying how much bigger than one, the sum of barycentric coordinates should be. A way to avoid this is to distinguish between boundary and interior nodes that are situated within the mesh. The interior nodes must not be moved, since when interpolating from one mesh to the other, there will be nodes which they will find no triangle to reside in. The way to avoid this problem is to pass through each node through each boundary element and calculate the barycentric value, and use the element of Mesh2 that produces the smallest value for the barycentric coordinates. This makes the procedures slightly slower, but guarantees that the interpolation is successful. Another matter to address is that once each node is found in which triangle it resides, then it is a must that the barycentric coordinates do not exceed the value of 1, since if that happens, oscillations on the results will appear during the interpolation process. To avoid this, it is necessary to move these points such that they reside on the closest boundary node, such that the addition of barycentric coordinates of a node relative to a triangle are always one.

5) Mesh Quality Results

Fig. 1a and 1b show the Delaunay triangulation and its geometric dual, the Voronoi diagram of a mesh that has been created in a commercial mesh generation software before, and after the treatment of the mesh with the element quality improvement algorithm, respectively. The Voronoi diagrams were chosen to be included due to their capacity to show the space (by separating it into cells) at which any point in this space is closest to the node of that cell, which is also the same node found in Delaunay triangulation, that the author uses in finite elements. In the case of a mesh with ideal elements, it would be expected that the Voronoi cells would be exact regular hexagons, depicting the equidistance of a mesh node to its surrounding six nodes. A close look at the two meshes before and after the mesh quality improvement operation shows the improvement in the regularity of the Voronoi cells, especially in the transition region from the finer to the coarser mesh, and vice versa.

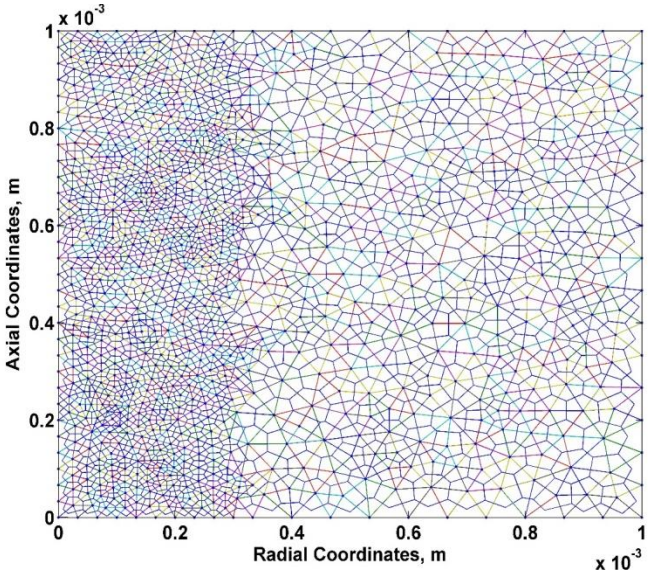

Fig. 1a. Schematic diagram of the Delaunay triangulation and its geometric dual, the Voronoi diagram (solid blue lines), created in a commercially mesh generation software.

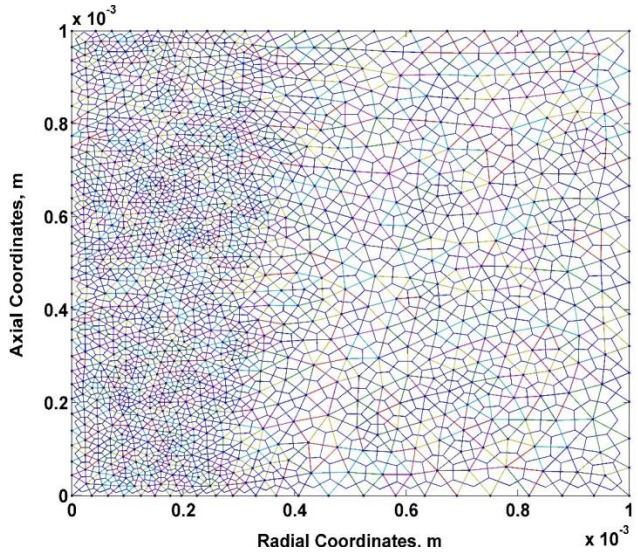

Fig. 1b. Schematic diagram of mesh after the element quality improvement algorithm developed including its Delaunay triangulation (multicolour solid lines) and its geometric dual the Voronoi diagram (solid blue lines).

Fig. 2a shows a bar chart displaying the number of elements that have similar element quality values of a mesh created in the commercial software, and Fig. 2b shows a bar chart displaying the number of elements that have similar element quality values of the mesh created in the commercial software, after being treated by the element quality improvement algorithm. Fig. 2a shows that in the mesh, a few bad quality elements exist that are expected to cause instabilities and oscillations in the results [13]. After the mesh quality treatment algorithm is applied, it is shown that not only the average value of the element quality of the mesh increases, but most importantly the bad elements disappear, achieving minimum element quality values of around 0.85. This can be easily shown by counting the total number of elements above a certain value which shows that an improvement in the element quality has been achieved overall, even for the elements which their values are very close to 1.

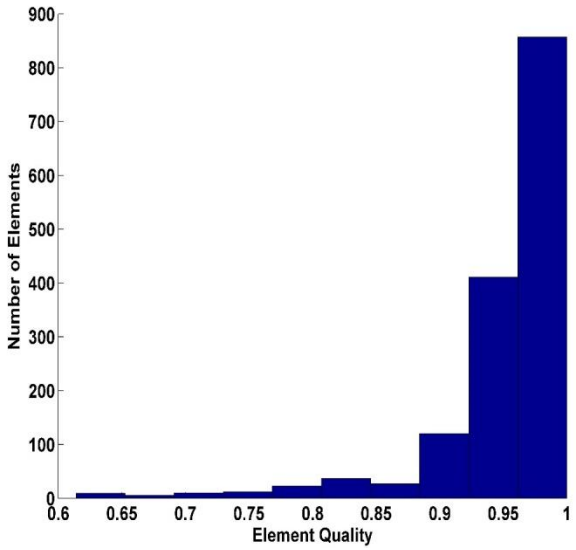

Fig. 2a. Bar chart displaying the number of elements that have similar element quality values of the mesh created in a commercially mesh generation software.

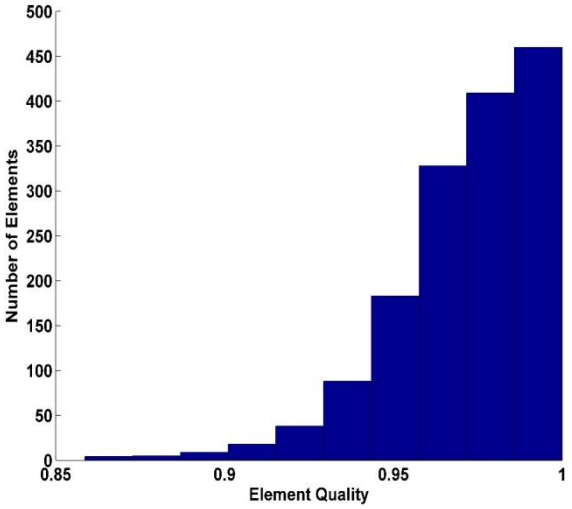

Fig. 2b. Bar chart displaying the number of elements that have similar element quality values of the mesh created in a commercially mesh generation software after being treated by the element quality improvement algorithm.

In order to test the element quality improvement algorithm in non-regular geometries, the case of a point-plane configuration has been tested due to the geometrical complexity of the point. Fig. 3a and 3b show the Delaunay triangulation of the mesh before and after being treated by the element quality improvement algorithm. On both diagrams, the element quality values that are less than 0.85 are displayed. In Fig. 3a, it is shown that many elements are below the threshold value especially at the proximity of the point, whereas after the treatment, Fig. 3b shows that no single element has element quality value less than 0.85. Since the nodes that define the boundary hyperboloid point cannot be moved randomly, but only using the hyperbolic equation to calculate the curve defining the point boundary, all the parameters of the analytical equation to form the hyperboloid point must be known in advance, such that boundary nodes are moved without changing the geometry of the mesh during the mesh jiggling, and node addition/removal operations.

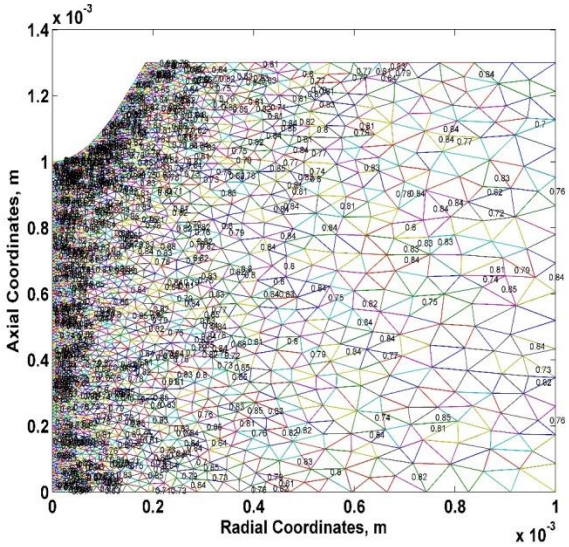

Fig. 3a. Schematic diagram of the Delaunay triangulation of a point-plane configuration mesh created in a commercially mesh generation software.

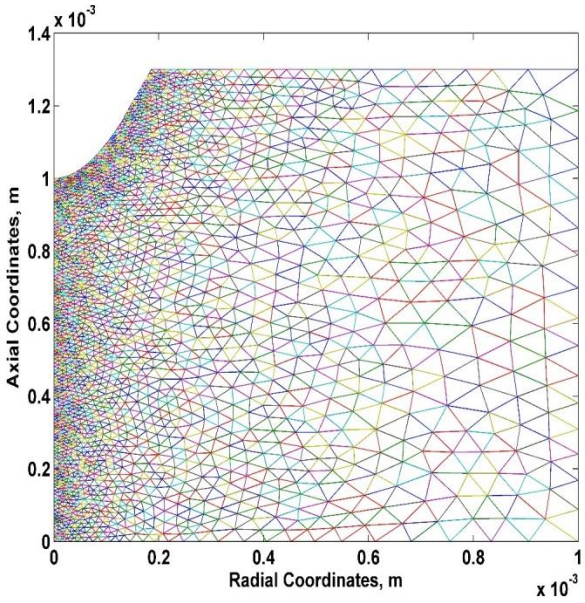

Fig. 3b. Schematic diagram of the Delaunay triangulation of a point-plane configuration mesh created in a commercially mesh generation software.

The strong asset of the mesh quality improvement algorithm is that typical two-dimensional meshes have been treated by the mesh quality improvement algorithm, and have shown to have minimum element quality values of around 0.85, which is ideal as far as element quality of any mesh is concerned. The above algorithm has been tested in a series of random meshes, with both regular and irregular domains such as point-planes, and it has been shown to always guarantee the best element qualities.

III. MPI AND CUDA

KYAMOS software is built based on excellent, speed and accuracy, and user-friendliness and tackles complicated multiphysics simulations, using GPU InfiniBand computing, under a state of the art, software protocol for conducting simulations.

In order to scale the computer simulations to multiple CPUs and GPUs, one needs a protocol that will connect the various nodes together. One option is to place as many cores on one computer, which currently the maximum being 64 cores, 128 threads into a single computer and perform shared computing, which means sharing one computers architecture, simultaneously. Even though this is attractive and utilized through the OPENMP protocol in high performance computing simulations, it is self-limiting, due to the limitation in the number of CPU cores we can put on a single machine. The other option is to utilize multiple number of these machines; however, we need a way to connect these machines and perform distributive computing. This can be achieved through a communication protocol such as the Message Passing Interface (MPI). Since communication time is of outmost important, we wish to minimize communication between nodes, one builds its problem in such a way to minimize such communication using partitioning of the mesh. A mesh partitioning tool can be used to split the elements in different equal partitions according to the number of CPUs and GPUs, necessary. Generally, it is a good practice to allocate 1 CPU to 1 GPU, even though using the HyperQ protocol, it enables multiple CPU threads or processes to launch work on a single GPU simultaneously, thereby dramatically increasing GPU utilization and hence significantly reducing CPU idle times. Once the partitioning is performed, it is a good practice to utilize ghost cells such that to minimize communication between nodes. The values of the ghost cells are updated at the end of each repeatable calculation and each processor can go its way to calculate independently the results, until the next update is necessary. For this exchange process to be efficient though, one needs to separate for each partition, the elements which are not shared with any nodes, and gather the elements which are expected to be sent to the various partitions and then gather the elements which are going to be received from each partition. Hence a complicated repartitioning from the initial mesh is necessary. One of the issues that may arise is the fact that it maybe that one element in a partition needs to be sent to 2 or more partitions, which this causes a major issue using the MPI routines since a single value can only be sent to one of the partitions, at least in an efficient way. To solve this problem, we first identify these elements and we ensure that these elements are duplicated in the mesh partitions as many times they need to be sent. This provides a very little overhead, however at the same time, solves a major issue in inter-communication and updating between the various partitions.

In the case of the Poisson solution, we are utilizing an iterative solver, the Conjugate Gradient method to calculate the result for the voltage. To achieve this, we setup the Poisson solver in such a way such that each partition provides its own contribution to the calculation of the voltage, since there will be common nodes that will need the contribution for multiple partitions for the solution, which is very different with the approach used for the simulation of the convection-diffusion equation, which each centered element being responsible to calculate is own nodes, only borrowing any neighboring values for the calculation of its mesh elements. The approach used in the Poisson seems to be more attractive since no extra added elements are needed to be duplicated in the domain and less calculations are necessary, since the addition of these duplicated elements provides additional overhead that should be ideally avoided.

Regarding CUDA capabilities, we utilize Tesla K80 cards which have a compute capability of 3.7, and are able to conduct parallel simulations at high scale, specifically 4.113 TFlops at single precision and 1.371 TFlops at double precision. Each Tesla K80 card has 2 GPUs of 12 Gb RAM at GDDR5. It has a base clock of 562 MHz and boost up to 824 MHz. It utilizes a 384-bit memory bus and has a bandwidth of 240.6 GB/s. They have the capacity to conduct stream calculations which allow the simultaneous calculations of independent functions in the software, GPU Boost technology to overclock the graphic cards for faster simulations, peer to peer access between multiple GPUs, which allows memory access from GPU memory to GPU memory,

bypassing the host memory within a node, and Remote Direct Memory Access (RDMA) to allow GPU to GPU memory direct access through the InfiniBand network. Finally, NVIDIA GPUs have the ability to support dynamic parallelism which allows one thread to launch a number of other threads, which is very useful in multiple for nested loops.

IV. INFINIBAND AND SWITCHES

Another way to minimize communication time is to use faster communication hardware. One of the major breakthroughs has come recently from an Israeli company called Mellanox which is now bought by the leader in GPU computing, NVIDIA. They have managed to build switches that have very low latency, and high bandwidth that make the communication times attractive.

The InfiniBand technology is developing rapidly mainly to Mellanox technologies. The SDR InfiniBand technology at 8 Gbps were introduced to the market in 2002 with latency 5 μs, then the DDR InfiniBand at 16 Gbps followed in 2005 with latency 2.5 μs, the QDR InfiniBand technology at 32 Gbps immersed in 2008 with latency 1.3 μs, then the FDR10 at 40 Gbps and FDR at 56 Gbps in 2011 with latency 0.7 μs, the EDR technology in 2014 at 100 Gbps with latency 0.5 μs, the HDR at 200 Gbps in 2017 and the NDR at 400 Gbps is expected after 2020. All the above speeds depict the throughput using 4 links.

KYAMOS software uses FDR technology to connect the various nodes together, with one of the 7 PCIE slots on the motherboard sacrificed for communication purposes.

V. FINITE ELEMENT FORMULATION OF POISSON EQUATION IN CUDA AWARE MPI

In the finite element Galerkin context, after the domain is discretized, the unknown potential within each element can be approximated according to Jin [24] with a linear shape function in three-dimensions as follows:

$$\varphi^e(x,y,z) = a^e + b^e x + c^e y + d^e z \quad (1)$$

where a^e, b^e, c^e and d^e are constant coefficients to be determined within each element e. Since tetrahedral elements are used in this case, four equations can be written for the potential at the tetrahedron four nodes, where the shape functions should obey the following relations:

$$\varphi_1^e(x,y,z) = a^e + b^e x_1^e + c^e y_1^e + d^e z_1^e \quad (2)$$

$$\varphi_2^e(x,y,z) = a^e + b^e x_2^e + c^e y_2^e + d^e z_2^e \quad (3)$$

$$\varphi_3^e(x,y,z) = a^e + b^e x_3^e + c^e y_3^e + d^e z_3^e \quad (4)$$

$$\varphi_4^e(x,y,z) = a^e + b^e x_4^e + c^e y_4^e + d^e z_4^e \quad (5)$$

From the above four equations, one can obtain the linear elemental shape function constant coefficients by solving the above four equations for a^e, b^e, c^e and d^e in terms of $\varphi_1^e(x,y)$, $\varphi_2^e(x,y)$, $\varphi_3^e(x,y)$ and $\varphi_4^e(x,y)$ to give the following equations:

$$a^e = \frac{1}{6V^e}\begin{bmatrix} \varphi_1^e & \varphi_2^e & \varphi_3^e & \varphi_4^e \\ x_1^e & x_2^e & x_3^e & x_4^e \\ y_1^e & y_2^e & y_3^e & y_4^e \\ z_1^e & z_2^e & z_3^e & z_4^e \end{bmatrix} \quad (6)$$

$$a^e = \frac{1}{6V^e}(a_1^e\varphi_1^e + a_2^e\varphi_2^e + a_3^e\varphi_3^e + a_4^e\varphi_4^e) \quad (7)$$

Similarly, for b, c and d coefficients:

$$b^e = \frac{1}{6V^e}(b_1^e\varphi_1^e + b_2^e\varphi_2^e + b_3^e\varphi_3^e + b_4^e\varphi_4^e) \quad (8)$$

$$c^e = \frac{1}{6V^e}(c_1^e\varphi_1^e + c_2^e\varphi_2^e + c_3^e\varphi_3^e + c_4^e\varphi_4^e) \quad (9)$$

$$d^e = \frac{1}{6V^e}(d_1^e\varphi_1^e + d_2^e\varphi_2^e + d_3^e\varphi_3^e + d_4^e\varphi_4^e) \quad (10)$$

where the volume of the tetrahedral element is:

$$V^e = \frac{1}{6}\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1^e & x_2^e & x_3^e & x_4^e \\ y_1^e & y_2^e & y_3^e & y_4^e \\ z_1^e & z_2^e & z_3^e & z_4^e \end{bmatrix} \quad (11)$$

If one substitutes equations (7, 8, 9, 10) into (1), it becomes:

$$\varphi^e(x,y,z) = \sum_{j=1`}^{4} N_j^e(x,y,z)\varphi_j^e \quad (12)$$

where the interpolation function is calculated as:

$$N_j^e(x,y,z) = \frac{1}{(6V_e)\left(a_j^e + b_j^e x + c_j^e y + d_j^e z\right)} \quad (13)$$

The governing equation for the Poisson equations is:

$$-\nabla.\left(\varepsilon(\nabla\Phi)\right) = \rho, \vec{\boldsymbol{x}} \in \Omega \quad (14)$$

with the Robin boundary condition:

$$\hat{\mathrm{n}}.b\nabla\Phi = \zeta(g - \Phi), \qquad \vec{\boldsymbol{x}} \in d\Omega \quad (15)$$

where $\hat{n}$ is the outward unit normal vector to the closed loop surface A, ζ is a positive value parameter, and g is the Dirichlet boundary condition. Depending on the choice of ζ and b, one can apply different boundary conditions of the Robin type. When ζ = 0,

one applies homogeneous Neumann boundary condition equal to 0. When b = 0, one can apply Dirichlet boundary conditions since ζ becomes irrelevant and we have that the value of the dependent variable Φ is set to the Dirichlet boundary condition. When both ζ and b have positive values, a Robin boundary condition is imposed. The left term represents the normal derivative at the boundary.

Performing the divergence operation on $\varepsilon(\nabla V)$ and substituting $\varepsilon = \varepsilon_o \varepsilon_r$ gives:

$$-\frac{\partial(\varepsilon_r \nabla\Phi)}{\partial x} - \frac{\partial(\varepsilon_r \nabla\Phi)}{\partial z} - \frac{\partial(\varepsilon_r \nabla\Phi)}{\partial z} = \frac{\rho}{\varepsilon_\circ} \quad (16)$$

where ε_r is the relative permittivity of the medium and ε_o is the permittivity of free space.

Performing the ∇ operator on Φ gives:

$$-\frac{\partial\left(\varepsilon_r \frac{\partial\varphi}{\partial x}\right)}{\partial x} - \frac{\partial\left(\varepsilon_r \frac{\partial\varphi}{\partial y}\right)}{\partial y} - \frac{\partial\left(\varepsilon_r \frac{\partial\varphi}{\partial z}\right)}{\partial z} = \frac{\rho}{\varepsilon_\circ} \quad (17)$$

Therefore, according to finite element Galerkin method, one needs to take the residual of the governing equation and try to minimize it. Hence, the residual of the Poisson equations is:

$$r = -\frac{\partial\left(\varepsilon_r \frac{\partial\varphi}{\partial x}\right)}{\partial x} - \frac{\partial\left(\varepsilon_r \frac{\partial\varphi}{\partial y}\right)}{\partial y} - \frac{\partial\left(\varepsilon_r \frac{\partial\varphi}{\partial z}\right)}{\partial z} - \frac{\rho}{\varepsilon_\circ} \quad (18)$$

The weighted residual within a tetrahedral finite element is as follows:

$$R_i^e = \iiint_{\Omega^e} N_j^e r^e dxdydz \; i = 1,2,3,4 \quad (19)$$

Substituting the residual formula (18) into the above equation (19), results in:

$$R_i^e = \iiint_{\Omega^e} N_j^e \left[-\frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial x}\right)}{\partial x} - \frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial y}\right)}{\partial y} - \frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial z}\right)}{\partial z} - \frac{\rho}{\varepsilon_\circ}\right] dxdydz \quad (20)$$

The following identity of partial differentiation for the x-directions holds:

$$\frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial x} N_j^e\right)}{\partial x} = N_j^e \frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial x}\right)}{\partial x} + \varepsilon_r \frac{\partial\varphi^e}{\partial x}\frac{\partial N_j^e}{\partial x} \quad (21)$$

$$N_j^e \left[\frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial x}\right)}{\partial x}\right] = \frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial x} N_j^e\right)}{\partial x} - \varepsilon_r \frac{\partial\varphi^e}{\partial x}\frac{\partial N_j^e}{\partial x} \quad (22)$$

Similarly, for y:

$$N_j^e \left[\frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial y}\right)}{\partial y}\right] = \frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial y} N_j^e\right)}{\partial y} - \varepsilon_r \frac{\partial\varphi^e}{\partial y}\frac{\partial N_j^e}{\partial y} \quad (23)$$

Similarly, for z:

$$N_j^e \left[\frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial z}\right)}{\partial z}\right] = \frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial z} N_j^e\right)}{\partial z} - \varepsilon_r \frac{\partial\varphi^e}{\partial z}\frac{\partial N_j^e}{\partial z} \quad (24)$$

Substituting the last three equations into equation (20), yields:

$$R_i^e = \iiint_{\Omega^e} \left(\varepsilon_r \frac{\partial\varphi^e}{\partial x}\frac{\partial N_j^e}{\partial x} + \varepsilon_r \frac{\partial\varphi^e}{\partial y}\frac{\partial N_j^e}{\partial y} + \varepsilon_r \frac{\partial\varphi^e}{\partial z}\frac{\partial N_j^e}{\partial z} - \frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial x} N_j^e\right)}{\partial x} - \frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial y} N_j^e\right)}{\partial y} - \frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial z} N_j^e\right)}{\partial z} - \frac{\rho}{\varepsilon_\circ} N_j^e\right) dxdydz \quad (25)$$

The divergence theorem is as follows:

$$\iiint_\Omega \left(\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} + \frac{\partial W}{\partial z}\right) d\Omega = \oint_A [(Ux + Vy + Wz]] . \hat{n} dA \quad (26)$$

where A is the enclosed surface for the control volume.

By applying the divergence theorem on the last three terms of equation (25) yields a formula:

$$\iint_{A^e} (N_j^e \vec{D}) . \widehat{n^e} dA = \iiint_{\Omega^e} \left(\frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial x} N_j^e\right)}{\partial x} + \frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial y} N_j^e\right)}{\partial y} + \frac{\partial\left(\varepsilon_r \frac{\partial\varphi^e}{\partial z} N_j^e\right)}{\partial z}\right) dxdydz \quad (27)$$

where:

$$\vec{D} = \varepsilon_r \frac{\partial\varphi^e}{\partial x}\vec{\imath} + \varepsilon_r \frac{\partial\varphi^e}{\partial y}\vec{\jmath} + \varepsilon_r \frac{\partial\varphi^e}{\partial z}\vec{k} \quad (28)$$

Substituting the above equation (27) into equation (25) gives:

$$R_i^e = \iiint_{\Omega^e} \left(\varepsilon_r \frac{\partial\varphi^e}{\partial x}\frac{\partial N_j^e}{\partial x} + \varepsilon_r \frac{\partial\varphi^e}{\partial y}\frac{\partial N_j^e}{\partial y} + \varepsilon_r \frac{\partial\varphi^e}{\partial z}\frac{\partial N_j^e}{\partial z} - \frac{\rho}{\varepsilon_\circ} N_j^e\right) dxdydz - \iint_{A^e} N_j^e \vec{D} . \widehat{n^e} dA \quad (29)$$

Regarding the last term $\iint_{A^e} N_j^e \vec{D} . \widehat{n^e} dA$ of the above equation, there is only contribution from the boundary elements and none from the inside elements of the mesh. If one assumes in the entire geometry domain, Dirichlet and Neumann boundary conditions, which is usually the case, then there is no contribution from this term in the finite element formulation and this term can be neglected.

Assuming such a case, the equation for the residual becomes:

$$R_i^e = \iiint_{\Omega^e} \left(\varepsilon_r \frac{\partial\varphi^e}{\partial x}\frac{\partial N_j^e}{\partial x} + \varepsilon_r \frac{\partial\varphi^e}{\partial y}\frac{\partial N_j^e}{\partial y} + \varepsilon_r \frac{\partial\varphi^e}{\partial z}\frac{\partial N_j^e}{\partial z} - \frac{\rho}{\varepsilon_\circ} N_j^e\right) dxdydz \quad (30)$$

Substituting the elemental shape functions from equation (12) yields:

$$R_i^e = \iiint_{\Omega^e} \left(\varepsilon_r \frac{\partial \left(\sum_{j=1}^{4} [\![N_j^e(x,y,z) \varphi_j^e]\!] \right)}{\partial x} \frac{\partial N_j^e}{\partial x} + \varepsilon_r \frac{\partial \left(\sum_{j=1}^{4} N_j^e(x,y,z) \varphi_j^e \right)}{\partial y} \frac{\partial N_j^e}{\partial y} + \varepsilon_r \frac{\partial \left(\sum_{j=1}^{4} N_j^e(x,y,z) \varphi_j^e \right)}{\partial z} \frac{\partial N_j^e}{\partial z} - \frac{\rho}{\varepsilon_\circ} N_j^e \right) dxdydz \quad (31)$$

Since the voltage within an element φ_j^e is constant, it can come out of the differential and the equation becomes:

$$R_i^e = \iiint_{\Omega^e} \left[-\varepsilon_r \frac{\partial \left(\sum_{j=1}^{4} N_j^e(x,y,z) \right)}{\partial x} \frac{\partial N_j^e}{\partial x} - \varepsilon_r \frac{\partial \left(\sum_{j=1}^{4} N_j^e(x,y,z) \right)}{\partial y} \frac{\partial N_j^e}{\partial y} - \varepsilon_r \frac{\partial \left(\sum_{j=1}^{4} N_j^e(x,y,z) \right)}{\partial z} \frac{\partial N_j^e}{\partial z} \right] \varphi_j^e dxdydz - \iiint_{\Omega^e} \frac{\rho}{\varepsilon_\circ} N_j^e dxdydz \quad (32)$$

Applying the residual formula to all four nodes of the tetrahedral element from i=1,2,3,4, yields:

$$\sum_{i=1}^{4} R_i^e = \sum_{i=1}^{4} \sum_{j=1}^{4} \iiint_{\Omega^e} \left[\varepsilon_r \frac{\partial N_j^e}{\partial x} \frac{\partial N_j^e}{\partial x} + \varepsilon_r \frac{\partial N_j^e}{\partial y} \frac{\partial N_j^e}{\partial y} + \varepsilon_r \frac{\partial N_j^e}{\partial z} \frac{\partial N_j^e}{\partial z} \right] \varphi_j^e dxdydz - \sum_{i=1}^{4} \iiint_{\Omega^e} \frac{\rho}{\varepsilon_\circ} N_j^e dxdydz \quad (33)$$

which can be written in a global matrix form as:

$$R^e = [K^e][\varphi^e] - [C^e] \quad (34)$$

where the elemental 4x4 matrix can be calculated as:

$$K_{ij}^e = \sum_{i=1}^{4} \sum_{j=1}^{4} \iiint_{\Omega^e} \left[\varepsilon_r \frac{\partial N_j^e}{\partial x} \frac{\partial N_j^e}{\partial x} + \varepsilon_r \frac{\partial N_j^e}{\partial y} \frac{\partial N_j^e}{\partial y} + \varepsilon_r \frac{\partial N_j^e}{\partial z} \frac{\partial N_j^e}{\partial z} \right] dxdydz \quad (35)$$

$$C_i^e = \sum_{i=1}^{4} \iiint_{\Omega^e} \frac{\rho}{\varepsilon_\circ} N_j^e dxdydz \quad (36)$$

In equation (35), the triple integral along the three-dimensions yields the volume of the tetrahedral element on the denominator, however the derivatives of the two shape functions yield a term of 1/(36 Ve^2) and when combined together, yield a term of 1/36Ve as a result in the denominator, hence the elemental assembly matrix becomes:

$$K_{ij}^e = \frac{1}{36V_e} \varepsilon_r \left(b_i b_j + c_i c_j + d_i d_j \right) \quad (37)$$

which represents a 4x4 matrix for each element in the three-dimensional tetrahedral mesh. Similarly, Ce, the load vector, after the two volumes from the volume integral on the numerator and the shape function on the denominator cancel each other out, it can be approximated by:

$$C_i^e \cong \frac{\rho_c}{\varepsilon_\circ} \frac{V_e}{4} \quad (38)$$

where the value of ρ_c is evaluated approximately at the centroid of the tetrahedral element. Now all the contributions from the elemental matrix are assembled into a global matrix R as follows:

$$R = \sum_{e=1}^{n} [\overline{R^e}] = \sum_{e=1}^{n} \overline{[K^e][\varphi^e]} - \overline{[C^e]} = [0] \quad (39)$$

where n is the number of total volume elements. Since the global residual elemental matrix must be minimum and must be set to zero, one has as a result the stiffness matrix and the load vector:

$$[K][\varphi] = [C] \quad (40)$$

There is generally a confusion in the literature regarding the interpolation coefficients within the interpolation functions for the mathematical formulation of the Poisson equation both in two and three-dimensions. Some formulations, when calculating the a, b, c and d coefficients, the mean only the distances between points, hence a division by 6Ve is necessary to calculate the interpolation function, likewise with our case, whereas in other formulations, the a, b, c and d coefficients represent the distances between points divided by 6Ve.

VI. Conjugate Gradient Method

The Laplace and Poisson equations are 2nd order diffusion elliptic equations which are time independent and provide a solution for example for an electric field or gas pressure. The Poisson equation can be solved in the context of finite elements using the Galerkin finite element method, which creates a matrix form which consists of a linear set of equations of the form Ax =b that needs to be solved. The solution of this equation can be solved using direct or indirect methods which are usually iterative methods. One of the most well-known methods for solving a linear system of equations is the conjugate gradient method that takes an iterative approach. The conjugate gradient method is based on the assumption that the matrix A is symmetric and positive definite. For a matrix to be symmetric, this means that $A=A^T$ and for the matrix to be positive definite, then **x** must be real and the following relation should hold: $x^TAx > 0$ for all non-zero vectors of **x** that belong to R^n and one needs to know vector **b**. Two vectors are conjugate when they are orthogonal to the inner product. Hence, two non-zero vectors u and v are conjugate to the A matrix if the following expression holds:

$$uTAv = 0 \quad (41)$$

Due to the fact that A is symmetric and positive definite, an inner product can be defined which involves u and v. Now we assume a set of m mutually conjugate vectors with respect to the matrix A as follows: P = (p_1,…, p_m) can form the basis for R^n and the solution of Ax=b can be expresses as a series of these basis vectors as:

$$x* = \sum_{i=1}^{m} \alpha_i p_i \tag{42}$$

which leads to:

$$a_k = \frac{< p_k, b >}{< p_k, p_k >_A} \tag{43}$$

Hence one needs to find a series of m conjugate vectors and calculate the a_k coefficients.

The conjugate gradient method can be simplified and implemented in an iterative approach by careful choice of the vectors $\mathbf{p_k}$ so that not all $\mathbf{p_k}$ vectors are necessary. Hence we start with an initial guess, usually x_o = 0. The solution depicted by x* happens also to be a unique minimizer of the quadratic function:

$$f(x) = \frac{1}{2} x^T A x - x^T b \tag{44}$$

where $x \in R^n$. The 1st derivative of f(x) becomes:

$$\nabla f(x) = Ax - b \tag{45}$$

which is identical to the solution of our linear system and since the 2nd derivative is: $\nabla^2 f(x) = A$, where A is positive definite, this means that a unique minimizer exists for the solution of the above equation. Since the gradient of f(x) =Ax-b and since we are taking an initial guess value of x_o, our initial p_o = b-Ax_o, is also the residual of the equation above but also the negative of the gradient of f(x=x_o).

Let's define the residual at step n; r_n = b - Ax_n, where r_n is the negative gradient of the quadratic function f(x) at x = x_n. One must ensure though that all search directions are conjugate, hence orthogonal to each other and this can be ensured by calculating the next search direction to be a function of the current residual and all the previous search directions as follows:

Define initial guess values for solution x at 0 i.e. $x_i^o = 0, x_{si}^o = 0,$
Update b_{si}
Set $r_i^o = b_i, r_{si}^o = b_{si}$
Set $p_i^o = r_i^o$
$\gamma^o = InnProd\,(r_i^o, r_{si}^o, r_i^o, r_{si}^o)$
Loop k STARTS k=0, 1……
$q_i^k = A_i p_i^k + B_i p_{si}^k$
$q_{si}^k = Update\,(B_i^T p_i^k + A_{si}^T p_{si}^k)$
$\tau^k = InnProd\,(p_i^k, p_{si}^k, q_i^k, q_{si}^k)$
$a^k = \frac{\gamma^k}{\tau^k}$
$v_i^{k+1} = v_i^k + a^k p_i^k$ $v_{si}^{k+1} = v_{si}^k + a^k p_{si}^k$
$r_i^{k+1} = r_i^k - a^k q_i^k$ $r_{si}^{k+1} = r_{si}^k - a^k q_{si}^k$
$\gamma^{k+1} = InnProd\,(r_i^{k+1}, r_{si}^{k+1}, r_i^{k+1}, r_{si}^{k+1})$
$if\,(\sqrt{\gamma^{k+1}} < Tol * Tol)\; END$
$\beta^k = \frac{\gamma^{k+1}}{\gamma^k}$
$p_i^{k+1} = r_i^{k+1} + \beta^k p_i^k$ $p_{si}^{k+1} = r_{si}^{k+1} + \beta^k p_{si}^k$
Loop k ENDS

$$p_n = r_n - \sum_{i<n} \frac{p_i^T A r_n}{p_i^T A p_i} p_i \tag{46}$$

and the next location is found by the following equation:

$$x_{n+1} = x_n + a_n p_n \tag{47}$$

with a_n calculated by:

$$a_n = \frac{p_n^T r_n}{p_n^T A p_n} \tag{48}$$

Now since we are following a partitioned implementation of the conjugate gradient method using

distributive computing, one needs to take into consideration the various partitions within the mesh and how to implement efficiently this algorithm. The formulation of the Poisson equation is based on the linear Galerkin finite element approximations.

The conjugate gradient method in the context of finite element Galerkin method in distributive computing can be summarized in the flowchart shown below. Using the above finite element formulation of a mass and consistent matrix as well as source term, the resulting matrix is of the form Ax=b which takes the form:

$$\begin{bmatrix} A_1 & \dots & \dots & \dots & B_1 \\ \dots & A_2 & \dots & \dots & B_2 \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & A_p & B_p \\ B_1^T & B_2^T & \dots & B_p^T & A_S \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_p \\ x_s \end{bmatrix} = \begin{bmatrix} b_1 \\ b_1 \\ \dots \\ b_p \\ b_s \end{bmatrix} \quad (49)$$

have dependency with neighboring partitions. It must be noted that all the non-interior nodes are accumulated at the end of the matrix stacked. To achieve this, we go through all the nodes of the partition and we identify the type of the node.

Some useful variables which are crucial in implementing our distributed computing algorithm are interior nodes which are the nodes not shared with other partitions, and the boundary nodes which are the nodes shared between two or more partitions. In order to distinguish between interior and boundary nodes, we use a Node class structure, which holds two members, .Type and .Local. If .Type=1, it represents an Interior Node and takes a local numbering value .Local from 0 and above in increasing order, whereas if it is a boundary node, .Type is 2 or higher, and .Local takes a value from 0 and above. Additionally, Dirichlet boundary conditions take .Type=0 and the .Local=-1 values.

Additionally, a Shared vector counts how many processes share a specific boundary node and the corresponding Node[I].Type value is set as >1 i.e. it can have values of 2 or 3 or 4 and above. This shared value is used to know how much fraction of the total contribution each processor has in the InnerProduct function.

The MaxCommon value denotes the maximum number of nodes that one process shares with any of the other processors which is connected to. This vector is used to communicate using MPI the boundary nodes contribution between processes, i.e. for example if process 0 shares with process 1, ten boundary nodes and with process 2, fifteen boundary nodes, then MaxCommon value will be 15.

The matrix for the interior nodes Ap is a square matrix of size InteriorNodes, whereas the matrix Bp is a rectangular matrix of size InteriorNodes x InteriorBoundaryNodes.

The matrix As is a square matrix that will hold values for the interior boundary nodes and it will be of size: InteriorBoundaryNodes x InteriorBoundaryNodes.

Since we are exploiting a distributed computing scheme, the overall matrix is partitioned with accordance to partitioning, with the index numbers 1, 2 and so on, referring to the different partitions of the mesh. Since one needs to deal with the dependency between neighboring meshes, we utilize the well-known method of ghost cells such that synchronization between processes or partitions is not necessary at all times, but only at the end of each time step. Hence the index p stands for primary nodes and refers to the interior nodes of a partition which do not share any dependency with neighboring partitions and index s stands for secondary which refers to the nodes which

The vectors bp and bs denote the RHS source terms vectors for the primary and secondary nodes and have size InteriorNodes and InteriorBoundaryNodes, respectively, whereas xp and xs denote the actual solution of the variable for the primary and secondary nodes, again with sizes InteriorNodes and InteriorBoundaryNodes, respectively.

VII. Capacitance and Total Energy in Poisson Domminated Phenomena

The total stored energy in a capacitor (We) is calculated as follows:

$$W_e = \frac{1}{2}\int\int\int_v \vec{D}.\vec{E}\,\partial V \quad (50)$$

and:

$$\vec{D} = \varepsilon\vec{E} \quad (51)$$

where $\vec{D}$ is the electric field density, $\vec{E}$ the electric field strength, ∂V is the elemental volume and v represents the volume of the capacitor and $\boldsymbol{\varepsilon}$ is the dielectric permittivity of the material.

Substituting equation (50) into (51), yields:

$$W_e = \frac{1}{2}\int\int\int_v \varepsilon\vec{E}.\vec{E}\,\partial V \quad (52)$$

Also:

$$\vec{E} = -\nabla V \quad (53)$$

where V is the voltage.

Substituting equation (53) into equation (52) results in:

$$W_e = \frac{1}{2}\int\int\int_{v_n} \varepsilon \nabla V . \nabla V \, \partial x \, \partial y \, \partial z \quad (54)$$

The value of ∇V depends on the elemental shape functions Ne within each element and so the integration domain is split into sub-domains corresponding to the tetrahedral elements such as:

$$W_e = \sum_{n=1}^{Ne} W_e^n \quad (55)$$

The energy within each element (W_e^n) is found by:

$$W_e^n = \frac{1}{2}\int\int\int_{v_n} \varepsilon \nabla V . \nabla V \, \partial x \, \partial y \, \partial z \quad (56)$$

where v_n represents the volume of the n[th] element in the domain.

The electric field strength is found as follows:

$$\vec{E} = \frac{1}{2V_e}\left(-\vec{\imath}\sum_{i=1}^{4} b_i V_i - \vec{\jmath}\sum_{i=1}^{4} c_i V_i - \vec{k}\sum_{i=1}^{4} d_i V_i\right) = -\nabla V \quad (57)$$

where b_i, c_i d_i are the elemental shape functions in x, y and z-directions.

Substituting for ∇V from equation (57) into equation (56), yields:

$$W_e^n = \frac{1}{2}\iiint_{v_n} \frac{\varepsilon}{6V_e^3}\left[\sum_{i=1}^{4} (\vec{\imath}b_i + \vec{\jmath}c_i + (\vec{k}d_i)\right) V_i \sum_{j=1}^{4} (\vec{\imath}b_j + \vec{\jmath}c_j + (\vec{k}d_j) V_j \sum_{k=1}^{4} (\vec{\imath}b_k + \vec{\jmath}c_k + (\vec{k}d_k) V_k] \quad (58)$$

Since b, c, d and V are constant within an element, equation (58) can be written:

$$W_e^n = \frac{1}{2}\iiint_{v_n} \frac{\varepsilon}{6V_e^3}[\sum_{i=1}^{4}\sum_{j=1}^{4}\sum_{k=1}^{4} (b_i b_j b_k + c_i c_j c_k + d_i d_j d_k) V_i V_j V_k \partial x \partial y \partial z] \quad (59)$$

Performing the triple integration, gives:

$$W_e^n = \frac{\varepsilon}{12V_e^3}\sum_{i=1}^{4}\sum_{j=1}^{4}\sum_{k=1}^{4} (b_i b_j b_k + c_i c_j c_k + d_i d_j d_k) V_i V_j V_k \quad (60)$$

Using equation (60) and equation (55), the total stored energy within the capacitor is calculated by integrating all the energy from all the elements in the domain.

But the total energy W_e within the capacitor is also equal to:

$$W_e = \frac{1}{2}CV^2 \quad (61)$$

where C is the capacitance and V is the potential difference between the two plates of the capacitor. Consequently, the total capacitance can then be found by using the following expression:

$$C = \frac{2W_e}{V^2} \quad (62)$$

VIII. CUDA AWARE MPI POISSON RESULTS

A. *Time test*

In order to benchmark the capabilities of the CUDA aware MPI solver, we have benchmarked the following case. We start with a cubic box of size 1 m and apply Dirichlet boundary conditions of 0 and 100 V at the cathode and anode, respectively, with ε_r = 1. Then, we vary the number of processors and calculate the time taken to conduct a successful simulation for a mesh constructed in NETGEN mesh software. The mesh size is chosen to be large such as to bottleneck a single GPU card with 68,430 Nodes in the mesh. For the benchmarking simulations, 18 GPU Tesla K80 cards were used and an AMD EPYC 32core/64 thread processor has been used.

Table No1: Running time with mesh variation and benchmarking of 3D Poisson equation with a mesh size of 68,430 Nodes.

ProcNo	2	4	8	10	14
Time	7 min, 57.7s	1m, 57.4s	62.8s	51, 6s	46.5s
ProcNo	16	18	20	32	
Time	44.95s	42.39s	45.6s	1min 1.9s	

In the above table, one can see that using 2 processors, it takes very long time and when 4 processors are used, the time taken is highly reduced. The trend continuous for the case of 8 threads but at a much slower rate and continues until using 18 threads, where it takes the minimum time. Thereafter, thread initiation and communication times of the threads take over, increasing the computed times of the simulations, up to 32 threads.

Additionally, we have constructed a series of six mesh sizes and benchmarked the time taken according to the number of processors. During the simulations, a single Tesla K80, CUDA enabled device was used with HyperQ technology for simultaneous access of the various threads. The results are shown in Fig. 4 below.

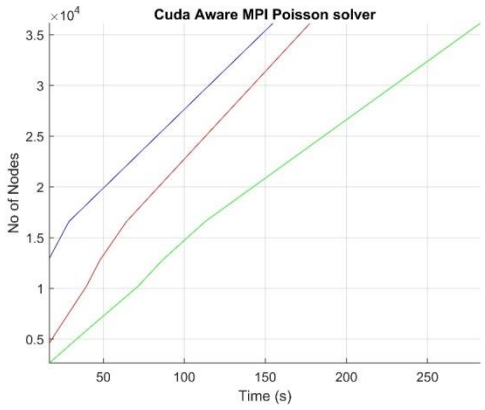

Fig. 4 Scaling test for the Poison solver for 6 different meshes under 3 different CPU units (blue-2, red 6, green 12).

Since the software is based mostly in conducting simulations on the GPU, one can see that the number of processors has a negative effect on the simulations, whereas the capability of the GPU is what matters. Since the GPU can handle single-handedly the size of all 6 meshes, it will be faster than any scaling of the threads and even the usage of multiple GPUs. To conclude, since our algorithms are implemented for nearly all the simulations to be performed on the GPU, it is not expected that any scaling will apply with the GPUs. The multiple GPUs are used in order to be able to solve larger computational problems, where a single GPU, bottlenecks because of processing power and/or available graphic card memory.

B. Accuracy test – Average errors

In order to test the accuracy of the CUDA aware MPI Poisson solver that we have constructed, we present the Poisson test result for a benchmark solution of a box of size 1 m, when applied with a voltage difference of 100 V between top and bottom plates. The analytical solution in free air is straightforward and poses an ideal scenario to test the accuracy and convergence of a Poisson solver. Table 2 shows the error which is the Mean Percentage Absolute Error (MPAE) comparison between analytical and actual solution in an attempt to calculate the average errors for both the voltage and electric field. It is shown that on average, the voltage has an error of 3.08 % and the electric field has on average, an error of 5.12 %. The convergence criteria used here were automatic and was decided by calculating the mesh tolerance which is the shortest distance between edges, and then multiple by 0.8 for finding the mesh tolerance. Thereafter, the convergence criteria were decided according to the criterion from the above flow chart: $\sqrt{\gamma^{\kappa+1}} < Tol * Tol$). The fact that we have used 6 different meshes meant that there was a different criterion for all 6 meshes since they each have their own tolerance; hence it does not necessarily mean that a finer mesh will produce more accurate results, especially if the mesh sizes are not very different, likewise with this case.

Table No2. Table plot of the voltage and electric field MPAE error when compared with analytical solution for 6 different meshes of varying accuracy.

Mesh	No1	No2	No3	No4	No5	No6
V_{error} (%)	2.24	1.58	3.82	5.56	3.16	2.07
E_{error}(%)	3.85	4.51	5.93	6.87	5.15	4.39

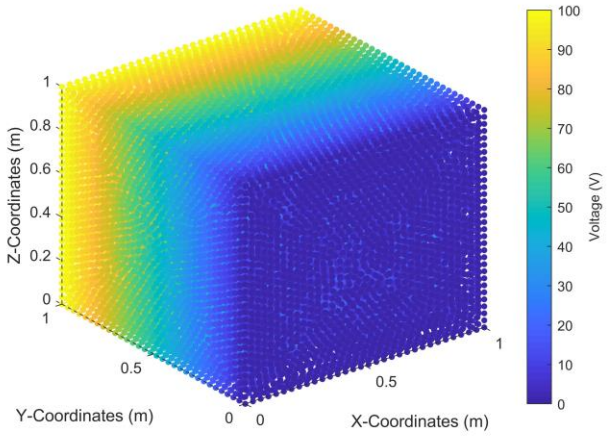

Fig. 5 Three-dimensional scatter plot of the voltage in a cubic box of size 1 m, when applied a voltage difference of 100 V.

Finally, in Fig. 5, we show the three-dimensional scatter plot of the result for the voltage. One can see from the graph that the results are as expected since the voltage increases linearly from the cathode to the anode from 0 to 100 V.

IX. CONCLUSIONS

KYAMOS software aims to realize the formulation, development, validation, and optimization of engineering problems by utilizing high performance computing through cloud-based distributed GPUs and state-of-the-art mathematical algorithms. Two necessary techniques to achieve this are the adaptive meshing with element quality improvement technique, and the efficient solution of partial differential equations. In this paper, we demonstrated the solution of elliptic dominated problems accurately and efficiently using state of the art software protocols, such as the CUDA aware MPI. To conclude, through state-of-the-art algorithms, and implementation procedures, we aim in disrupting the market due to: (a) low pricing schemes from utilization of free open source software for the viewing, geometry, mesh and plot editors, (b) fast and accurate algorithms developed in state of the art CUDA aware MPI protocol and hopefully become leaders in the CAE industry in the long run.

ACKNOWLEDGMENT

This work was co-funded by the European Regional Development Fund and the Republic of

Cyprus through the Research and Innovation Foundation (Project: START-UPS/0618/0058).

REFERENCES

[1] J. W. Thomas, "Hyperbolic Equations," in *Numerical Partial Differential Equations: Finite Difference Methods*, J. W. Thomas, Ed. New York, NY: Springer New York, 1995, pp. 205-259.

[2] J. F. Thompson, B. K. Soni, and N. P. Weatherill, *Handbook of grid generation*. CRC press, 1998.

[3] J. F. Thompson, Z. U. Warsi, and C. W. Mastin, *Numerical grid generation: foundations and applications*. North-holland Amsterdam, 1985.

[4] Y. Kallinderis, "Adaptive methods for compressible CFD," *Computer methods in applied mechanics and engineering,* vol. 189, no. 4, 2000.

[5] M. Filipiak, "Mesh generation," *Edinburgh parallel computing centre, the University of Edinburgh, Edinburgh,* 1996.

[6] S. McRae, "Adaptive mesh algorithms-a review of progress and future research needs," in *15th AIAA Computational Fluid Dynamics Conference*, 2001, p. 2551.

[7] A. B. Díaz Morcillo, L. Nuño Fernández, J. V. Balbastre Tejedor, and D. A. Sánchez Hernández, "Adaptative mesh refinement in electromagnetic problems," 2000.

[8] A. Papadakis, G. E. Georghiou, and A. Metaxas, "New high quality adaptive mesh generator utilized in modelling plasma streamer propagation at atmospheric pressures," *Journal of Physics D: Applied Physics,* vol. 41, no. 23, p. 234019, 2008.

[9] L. Chen, "Mesh Smoothing Schemes Based on Optimal Delaunay Triangulations," in *IMR*, 2004, pp. 109-120: Citeseer.

[10] D. A. Field, "Laplacian smoothing and Delaunay triangulations," *Communications in applied numerical methods,* vol. 4, no. 6, pp. 709-712, 1988.

[11] L. A. Freitag, "On combining Laplacian and optimization-based mesh smoothing techniques," Argonne National Lab., IL (United States)1997.

[12] Q. Du, V. Faber, and M. Gunzburger, "Centroidal Voronoi tessellations: Applications and algorithms," *SIAM review,* vol. 41, no. 4, pp. 637-676, 1999.

[13] L. Chen and J.-c. Xu, "Optimal delaunay triangulations," *Journal of Computational Mathematics,* pp. 299-308, 2004.

[14] T. Zhou and K. Shimada, "An Angle-Based Approach to Two-Dimensional Mesh Smoothing," *IMR,* vol. 2000, pp. 373-384, 2000.

[15] E. VanderZee, A. N. Hirani, D. Guoy, and E. Ramos, "Well-centered planar triangulation–an iterative approach," in *Proceedings of the 16th International Meshing Roundtable*, 2008, pp. 121-138: Springer.

[16] H. Erten, A. Üngör, and C. Zhao, "Mesh smoothing algorithms for complex geometric domains," in *Proceedings of the 18th international meshing roundtable*: Springer, 2009, pp. 175-193.

[17] R. Löhner, "An adaptive finite element scheme for transient problems in CFD," *Computer Methods in Applied Mechanics and Engineering,* vol. 61, no. 3, pp. 323-338, 1987.

[18] R. Löhner, "Regridding surface triangulations," *Journal of Computational Physics,* vol. 126, no. 1, pp. 1-10, 1996.

[19] M. J. Berger and P. Colella, "Local adaptive mesh refinement for shock hydrodynamics," *Journal of computational Physics,* vol. 82, no. 1, pp. 64-84, 1989.

[20] M. J. Berger and A. Jameson, "Automatic adaptive grid refinement for the Euler equations," *AIAA journal,* vol. 23, no. 4, pp. 561-568, 1985.

[21] M. J. Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial differential equations," *Journal of computational Physics,* vol. 53, no. 3, pp. 484-512, 1984.

[22] L. Devroye, E. MŘcke, and B. Zhu, "A note on point location in Delaunay triangulations of random points," *Algorithmica,* vol. 22, no. 4, pp. 477-482, 1998.

[23] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta informatica,* vol. 4, no. 1, pp. 1-9, 1974.

[24] J.-M. Jin, *The finite element method in electromagnetics*. John Wiley & Sons, 2015.