I am a first year PhD student in the Mechanical Engineering Department. My research interests lie in the fields of Computational Solid Mechanics and Computational Geometry. Since both of these areas involve working with large data especially for applications in the real world, parallel computing provides a natural solution to tackling large and complex problems without compromising on the accuracy of the solution. I have taken this course as I believe it will provide me with a background in parallel computing which will enable me to attack some of the complex problems in this field.

THE PROBLEM:

Motivation
The Finite Element Method (FEM) solves a partial differential equation (PDE) over a complex domain which is approximated by a mesh of simple geometric primitives. The mesh itself may be unstructured or graded according to the irregularity of the domain or the accuracy of the solution. As no useful analysis is carried out in the mesh generation step, it is a bottleneck and a need, therefore, arises to generate meshes very quickly. For complex domains or problems where a very accurate solution is necessary, it is necessary to generate a mesh consisting of perhaps millions of elements in a very short time. Therefore, for such problems, parallel mesh generation appears to be the best solution.

Problem
A number of applications in physics, biology, and engineering require the solution of a PDE over a complex domain which is meshed in two dimensions. Even if the problem is non-planar, symmetry may be employed (such as cylindrical or spherical) in order to solve the problem over a two-dimensional mesh. Shewchuk [1] has proven that the accuracy of the numerical solution as obtained by solving the particular PDE over the mesh depends on the size and shape of the elements that make up the mesh. Hence, in order to maintain this accuracy for complex problems, several researchers have attempted to generate meshes in two dimensions.

The intuitive way to solve this large problem is to divide it into $N$ smaller subproblems, each of which are distributed to one processor and solved in parallel. As far as possible, one would like the subproblems to have least degree of coupling so that there is little communication between various processors. The solution proceeds by carrying out domain decomposition of the initial domain i.e. subdividing the initial domain into several smaller, non-overlapping sub-domains. Meshing is then carried
out in parallel over these sub-domains by mapping each sub-domain to a processor, followed by a post processing merging step in order to form the final mesh. [2]

As a specific example, we choose Constrained Delaunay Triangulation for meshing as several properties of Delaunay Triangulations offer guarantees on triangle shape and size. The partitioning of the domain into sub-domains and distribution to various processors is carried out by the use of the MeTiS graph partitioning software [3]. The meshing algorithm proceeds by creating an initial coarse background mesh which satisfies neither size nor shape constraints. It is proven in [1] that the smaller the size of the triangles and the more regular (close to equilateral) the shape of the triangles creating the mesh, the higher the accuracy of the solution. After this, the triangles in the initial mesh that do not satisfy the size constraints are simply subdivided by creating 4 triangles (joining the midpoints of the 3 sides). For triangles that do not satisfy the regularity (also termed as quality) constraint, additional points are inserted in the domain in order to improve the quality [4]. These two operations are carried out in parallel for each sub-domain.

In addition, since several of the sub-domains will share a common boundary, communication cannot be avoided between the various sub-domains. As a result, whenever the geometry or topology of an edge lying on the common boundary of two sub-domains is modified, a message is passed to the other sub-domain in order to modify the geometry of the common edge before processing it further. This, therefore, does not lead to a completely parallel algorithm, and one of the main research issues in this problem is the minimization of this communication time. [2]

The algorithm was coded and implemented by the researchers in [2] on a cluster consisting of a mixture of single cpu, dual cpu and quad cpu machines. It thus operates on distributed memory. The implementation is in C++ along with MPI. The MeTiS library [3] is used to carry out initial partitioning, and the Triangle library [5] is used to generate the initial coarse mesh. It is observed that in order to generate approximately 1.4 billion triangles for 144 processors, the parallel version of the code is about 80 times faster than the serial version of the code. The efficiency, thus, is approximately 60%. The reasons for lower efficiency appear to be communication between processors as well as possible workload imbalance.

![Fig. 1: Speed-ups by varying number of processors [2]](image-url)
This application shows very good speed-ups in generation of complex meshes consisting of over a billion triangles. It also shows very good scalability over several processors as shown above in Fig. 1. As such, it shows great potential for further experimentation and the tuning of the serial algorithm in order to optimize several other features of parallel algorithms and minimize communication times.

REFERENCES: