Unstructured FEM: Intermediate Report 2

Mohsin Ali Chaudry, Raghavan Lakshmanan, Abhishek Y. Deshmukh and A. Emre Ongut

German Research School for Simulation Sciences GmbH
SiSc Laboratory
a.deshmukh@grs-sim.de *ongut@cats.rwth-aachen.de*

Abstract: This engineering project of SiSc Lab involves the development of a parallel finite element solver using unstructured grids. The developed and optimized code is validated using analytical solutions for a time-dependent engineering heat transfer problem. This report consists of the approach followed for parallelizing the serial code.

1 Introduction

Day by day the requirement for faster computing is increasing owing to increasing complexity and the size of the problems that need to be solved within stipulated amount of time. With advances in hardware hitting the atomic wall, there needs to be a way to use whatever resources are available to the maximum capacity. The large problems need to be solved fast. This motivates the use of the parallel computing that employs multiple cores to solve the problem at hand. However, the serial program needs to be significantly modified if it is to run on multiple cores. In the following sections, the approach followed by us to parallelize the serial code is described.

There are two main computing paradigms [1]:

- Distributed computing
- Multithreading

Distributed computing involves non-uniform memory access. In general, cores and memory are distributed on several machines and are connected through an interconnect. The interconnect may have different topologies like bus, radial, star, torus, omega, etc. depending on the usage. This may or may not require the processors on different machines to communicate the data required to solve the problem being considered. A standard for this communication has been developed and is called MPI (Message Passing Interface) standard.

Multithreading usually involves uniform memory access. All the cores have a shared memory from where they have access to all the data related to the problem being solved. Here, processors don't need to communicate the data. In addition multiple threads can be spawned on each core that can work on different independent piece of data and/or program. A standard for this paradigm is called OpenMP.

There is also a hybrid paradigm that combines the distributed computing and multithreading.

The choice of computing model depends upon the hardware available at hand. The RWTH compute cluster supports all of the above computing models. For this project, we have considered distributed computing paradigm because generally clusters with distributed resources are(or may be) readily available to the users. We want our code to be readily usable to maximum possible users who may want to use it for their problem.

2 Approaches to Parallelizing

The heat transfer problem to be solved involves a physical domain of interest where the temperature and heat fluxes are to be investigated. The physical domain is discretized into a mesh with triangular elements. Higher the resolution required higher will be the number of mesh elements. In parallel solution, the meshed domain can be divided into parts and each of this part can be distributed to each of the processor for solving. This is known as data parallelism. There are two ways to achieve this. Either read the mesh files namely mxyz and mien parallely or first decompose these files into required number of processors and then each processor can read parallely from its part.

Each of the approaches has its own advantages and drawbacks.

In the first approach (Fig. 1) where processors read parallely from single mxyz and single mien file, each processor may not contain all the data related to node co-ordinates on the same processors. It requires to get missing node information from other processors. This adds the communication overhead. However, there is no need to preprocess the mesh data before running the parallel code.

In the second approach (Fig. 2), the mesh data is preprocessed. This is done to generate the separate mesh data for each processor so that it has all the elements and node information available. In this case, the serial code needs the least modification. The processors need to communicate the data for the nodes which are common the multiple processors in solver part only. The rest of the parts from serial code can be directly used as they are. This approach is useful because once we have distributed mesh, the same mesh can be reused again and again for parameter studies. It involves one time overhead of generating distributed mesh data. However, this preprocessing takes time for high resolution meshes to be distributed on less number of processors.

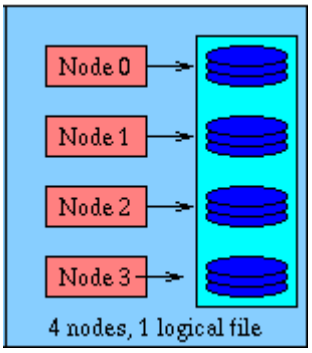

Figure 1: Parallel I/O 1 [2]

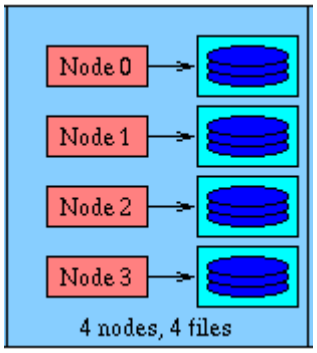

Figure 2: Parallel I/O 2 [2]

We have followed second approach in our code because once we have a distributed mesh from a preprocessor, the communication overhead is reduced significantly. We can reuse the distributed mesh for further study of the problem using different boundary conditions. We expect good scalability with this approach.

3 Mesh Partitioning

Before the mesh can be used efficiently in parallel, another important process has to be performed over it i.e. Mesh partitioning (Fig. 3, 4). Mesh partitioning is a process of dividing the generated mesh into subdomains. These subdomains can be mapped onto a processor of a parallel machine. The primary goal of mesh partitioning is to minimize communication while maintaining load balance. The communication is associated with the nodes that lie on the boundaries between subdomains and are shared by more than one processor. Processors sharing a node must communicate. Communication time depends on both the message sizes, which increase with the number of shared nodes, and the number of messages, which increases with the number of adjacent subdomains. The load can be

predicted to be proportional to the number of nodes on that processor. Prediction becomes more difficult when nonlinearities are present. In these cases, the work per node is solution-dependent.

The partitioners are based on following algorithms: Greedy, Kernighan-Lin, Recursive Coordinate Bisection, Recursive Spectral Bisection, Recursive Inertia Partition, Recursive Graph Bisection, algorithm of Miller, Teng, Thurston, and Vavasis, and Simulated Annealingthe. Miller, Teng, Thurston, and Vavasis algorithm uses geometric information to construct a separator, i.e. a set of nodes whose removal separates the mesh into two pieces of roughly equal size. Each of these pieces is then recursively partitioned until the desired number of subdomains is reached. The Miller algorithm, in practice, rapidly produces high quality partitions.

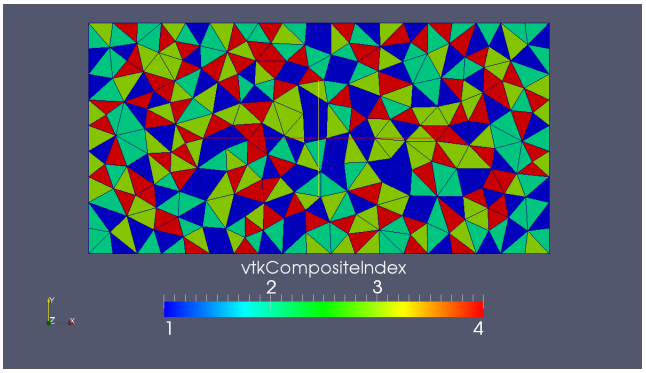

Figure 3: Before Mesh partitioning

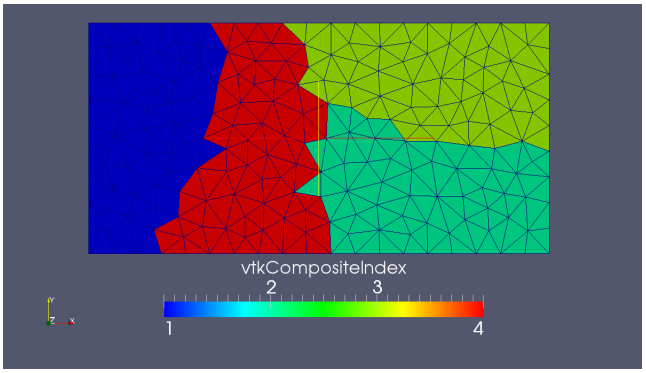

Figure 4: After Mesh partitioning

A typical graph partitioning algorithm finds a decomposition of the graph(non-weighted) with preferably the same number of nodes within each partition and with a minimal number of edges between the partitions.

There are many public mesh partitioning tools. Some of which are listed below:

1. METIS
2. TOP/DOMDEC
3. HARP
4. CHACO
5. JOSTLE
6. SCOTCH

In this project, a partition utility is used to partition the usual mien FEM connectivity into a set of contiguous subdomains using the Metis graph partitioning library available on cluster. This generates a mprm file which is used in subsequent preprocessing of mesh.

4 Preprocessor

The preprocessor takes the raw mxyz, mien, minf, mrng files and decomposes them according to the partitioning defined in mprm file into number of processors specified in "settings.in". This is a serial program. It generates an additional file "procb" which contains the information of nodes shared among the processors. This file contains following information per node: global node number, how many processors it is shared with, array of the processor ranks it is shared with. If the node is not shared with any other processor, the array has default value of -1. Sample procb file for *proc*_0 is shown below:

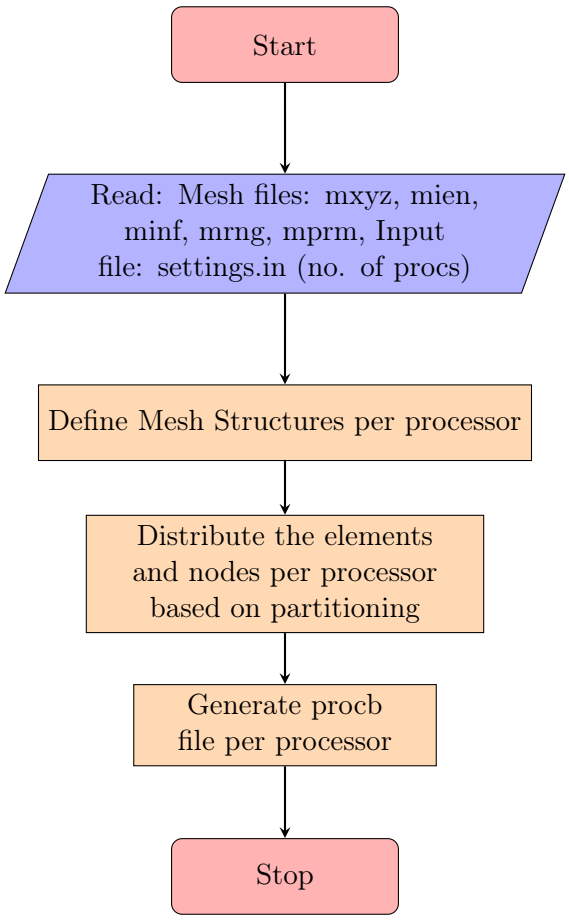

Figure 5: Preprocessor Flow Chart

0	0	-1	-1	-1	-1
1	0	-1	-1	-1	-1
2	0	-1	-1	-1	-1
3	1	1	-1	-1	-1
13	1	1	-1	-1	-1
14	1	1	-1	-1	-1
15	2	1	2	-1	-1
16	2	1	2	-1	-1

Contents of a sample procb file

5 Overall Flow of program

Overall program flow is shown in figure 5. Each processors runs through the steps shown in the flowchart. This generates VTK data per time step for each processor separately.

6 MPI Communication

Each of the processor reads its portion of mesh, calculates jacobian for each element, assembles element level matrices and finally in solver part assembles the lumped mass matrix (M) and right hand side vector (RHS) of the equation to be solved. However, for the nodes which are shared between different processors, these assembled entities still need the contribution from elements on the other processors.

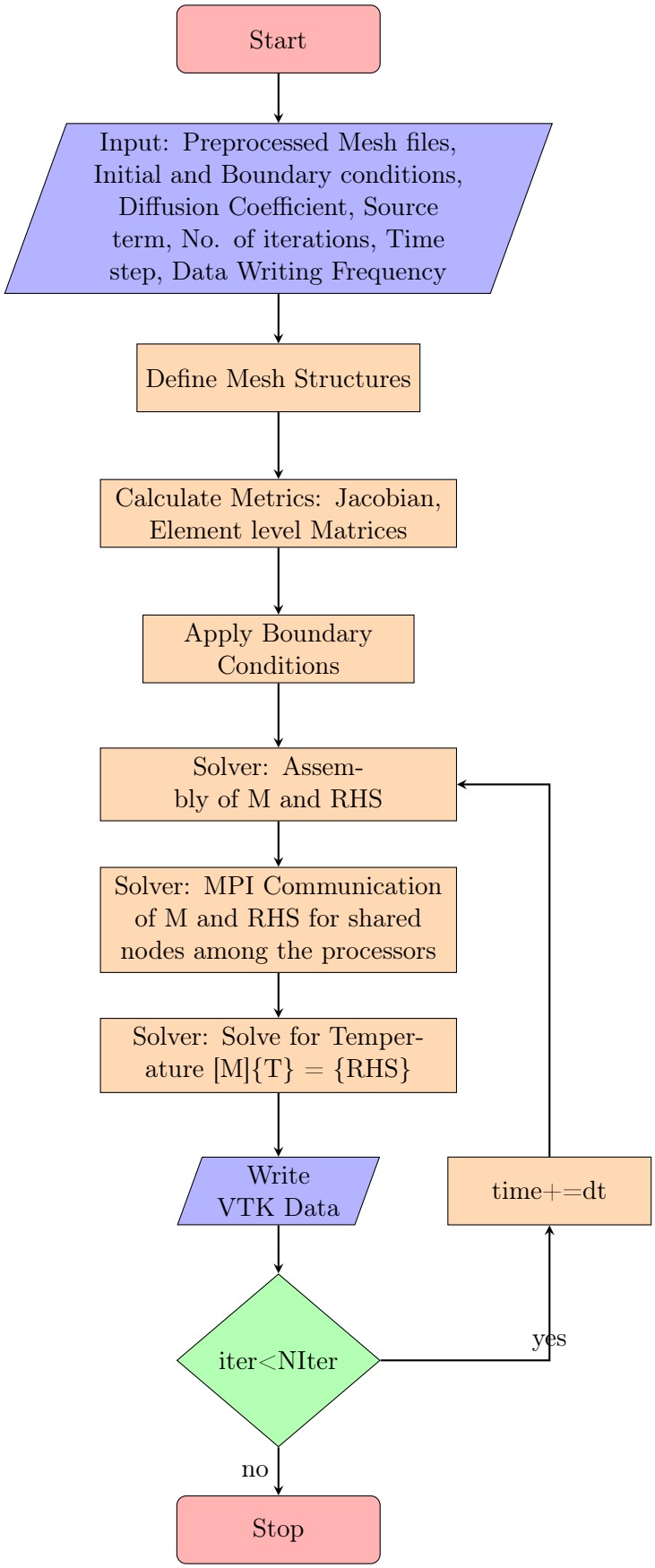

Figure 6: Overall Program Flow Chart

This information needs to be communicated between the concerned processors before the equation can be solved. The procb file which is generated by preprocessor comes into play here.

Two side communication has been used. The communication is initiated by each of the processors based on the procb file. For the node which is shared between the two processors, the data i.e. M and RHS for the corresponding node is exchanged. On the receiving processor, the addition of received M and RHS to the correct location of existing M and RHS is achieved by matching the global node number.

7 Results

7.1 Validation

Figures 7 and 8 show the result of the problem with Dirichlet conditions. A rectangular domain is considered for heat diffusion problem. Left and bottom walls are kept at 1000 K while top and right walls are maintained at 300 K and the temperature profiles is allowed to develop over time. The snapshot is taken at t=0.1s. The temperature profiles in both the figures are identical which validates the parallel code.

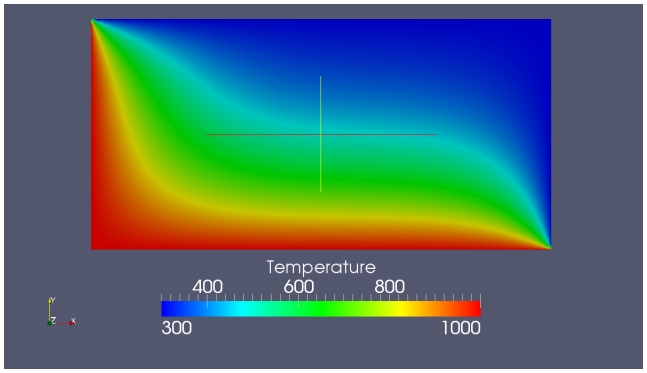

Figure 7: Serial

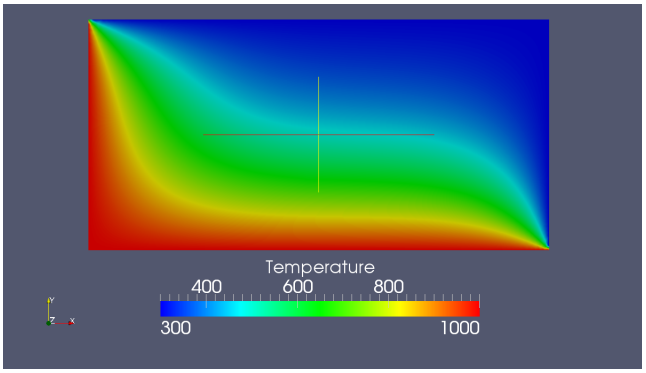

Figure 8: Parallel: nP=4

7.2 Scalability

Figure 9 shows the speedup obtained for finest mesh from Rectangular domain.

8 Summary

The overall approach of parallelization of a serial code for the problem of interest is to use a distributed computing model. The parallelized code shows some scalability though there is a lot of scope for improvement. The efforts are going on to bring the speedup close to linear one.

References

[1] Class materials for Parallel Programming I.

[2] https://computing.llnl.gov/LCdocs/ioguide/.

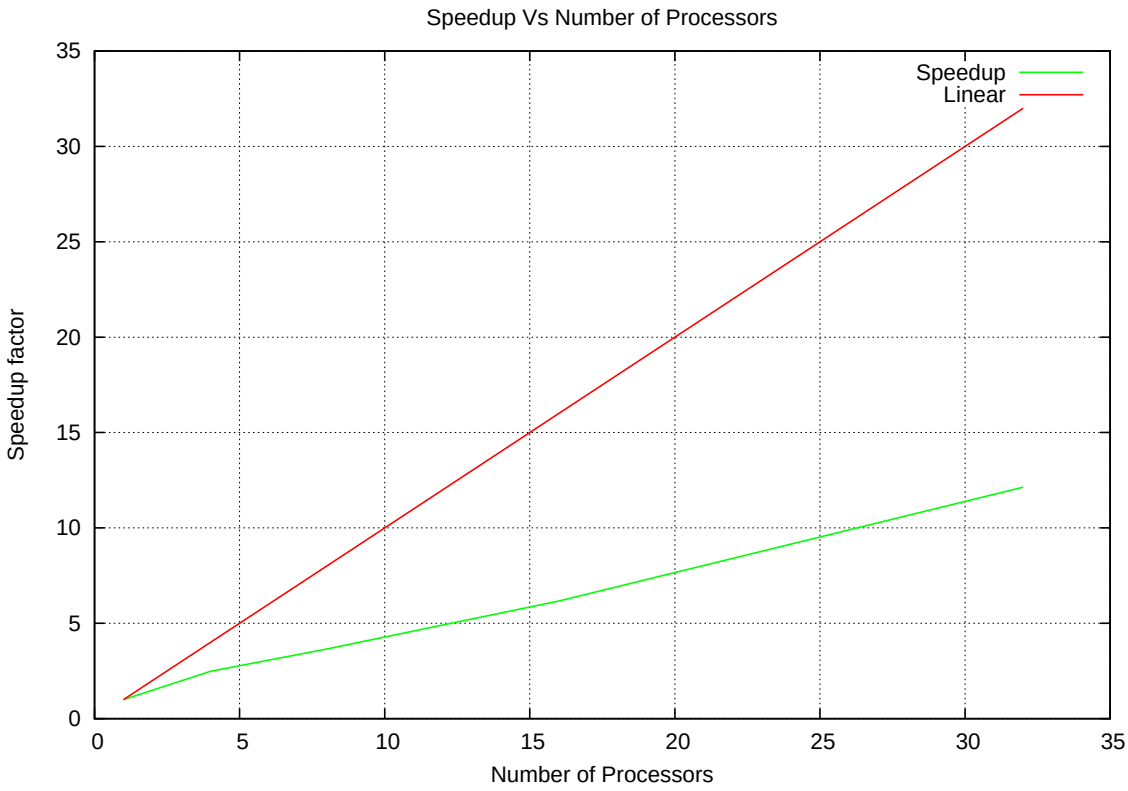

Figure 9: Speedup Vs. Number of Processors