A Comparison of Sparse and EBE Storage Schemes on the Efficiency of Parallel Conjugate Gradient Iterative Methods for Finite Element Analysis.

Shivaraju Gowda

Date: July 18, 2002

Advisor: **Dr. Lonny L Thompson**

Department of Mechanical Engineering,

Clemson University

Computational Mechanics Laboratory

CLEMSON UNIVERSITY

Summary

- **Preconditioned conjugate gradient iterative** method for FEA in parallel on distributed memory machines using MPI and Fortran90.
- Comparing two types of matrix storage schemes

  EBE – Element-By-Element

  CSR – Compressed Sparse Row
- Implemented **Jacobi Preconditioner.**
- Scalability analysis on two type of parallel machines.
- Applied to structured and unstructured mesh for 2D and 3D problems.

Outline

Introduction

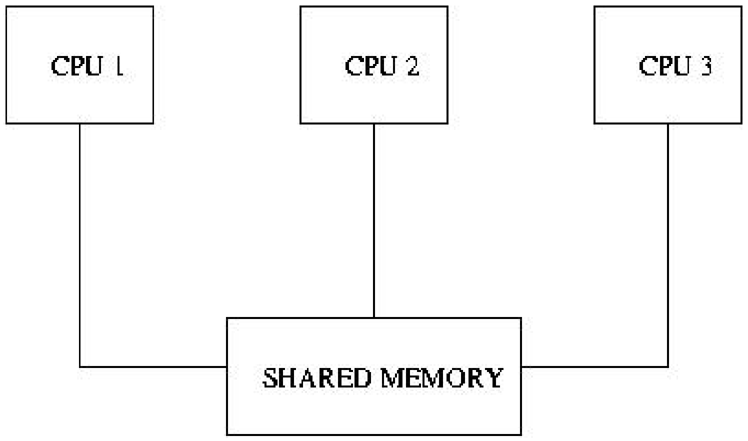
Why Parallel ?

Use of Multiple Processors to process single application simultaneously

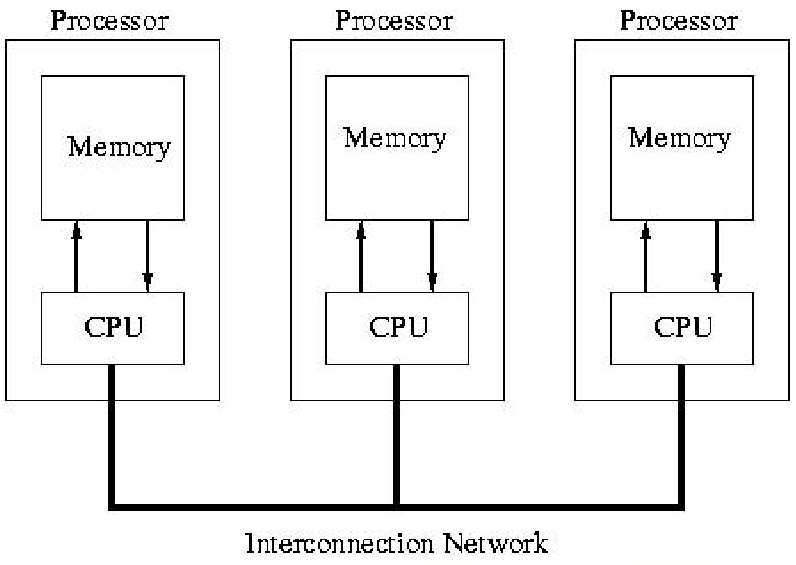
- Reduce total wall clock simulation time
- Solve large complex problems which do not fit into the local memory of a single system

Architectures:

Shared Memory (OpenMP, HPF)



Distributed memory(MPI)



Introduction

Why Iterative Methods ?

+

- **Direct Methods**, robust and predictable. For 3D problems, Computational time and memory requirements increase with $O(n^3)$.
- **Iterative methods**, suitable for such problems, especially for sparse matrices from finite element descretization.
- Iterative methods use the knowledge of the problem for better convergence
- Suitable for parallel implementation.
- With good preconditioner can match the performance of Direct methods.

−

- No general purpose routine for all problems.
- Knowledge of problems is required on the part of the end user to select the best method, preconditioner and initial value.

Finite Element Formulation

Second order linear elliptic partial differential equation

$$-\underline{\nabla}\cdot\left(a(\underline{x})\underline{\nabla}u\right)=f(\underline{x}),\qquad \underline{x}\in\Omega\subset\Re^{2}$$

subject to Dirichlet Boundary Condition

$$u\big|_{\partial\Omega}=0$$

The domain Ω is discretized into a non-overlapping set of elements.

Piecewise linear finite element approximation,

$$u=\sum_{i=1}^{N}u_iN_i(\underline{x}),\qquad v=\sum_{i=1}^{N}v_iN_i(\underline{x})$$

where $N_i(\underline{x})$ are piecewise linear basis functions for i = 1,..,N

$$\int_\Omega a\underline{\nabla}v\cdot\underline{\nabla}u\,d\underline{x}=\int_\Omega fv\,d\underline{x}$$

Matrix Problem

The set of equations may be written in matrix notation as

$$A\underline{u} = \underline{b}$$

Where,

$$\underline{u} = (u_1, \ldots, u_N)^T$$

$$A_{ji} = \int_\Omega a \underline{\nabla} N_j \cdot \underline{\nabla} N_i d\underline{x},$$

$$b_j = \int_\Omega f N_j d\underline{x}$$

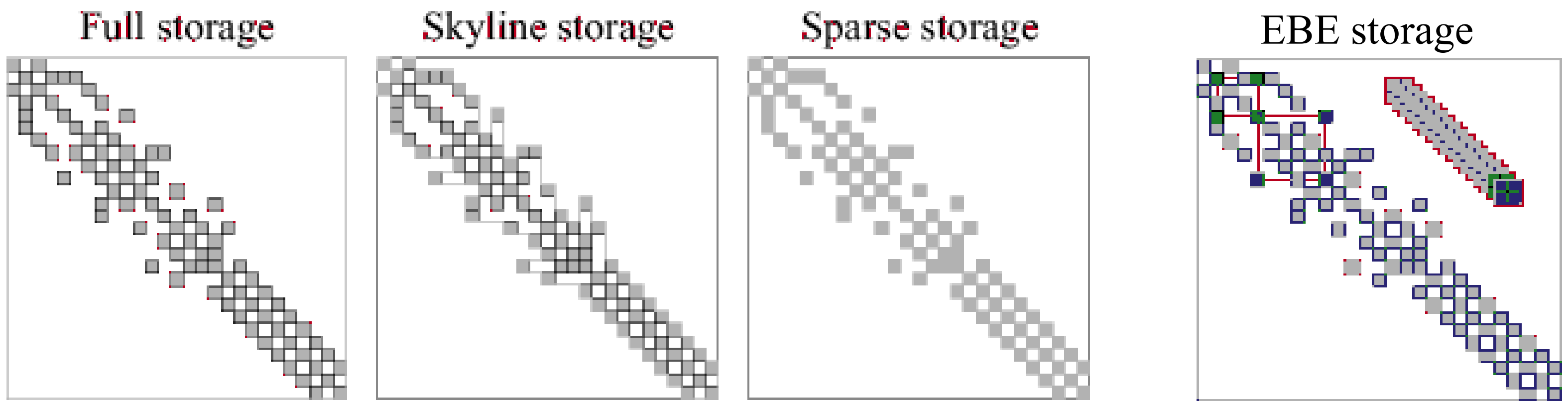
General solution Procedure :

- Calculate the stiffness coefficients of all the elements.
- Assemble the global stiffness matrix A.
- Solve the system of equations using an iterative algorithm.

Computational Mechanics Laboratory

CLEMSON UNIVERSITY

Matrix Storage



Skyline Storage

- Stores all coefficients within skyline, including zeros
- Requires renumbering to minimize skyline
- Efficient for direct solvers on shared-memory machines

Sparse Storage

- Stores nonzero coefficients only
- Predominantly indirect access patterns
- Efficient for iterative methods on distributed memory architectures

Element by Element storage

- Retains only element-level matrices, no global assembly
- Efficient for iterative methods on distributed memory
- Matrix-vector product done in 3 stages, gather to element local, matrix-vector product at element level and scatter to global
- Element matrices can be recalculated as needed minimal storage but slower

Computational Mechanics Laboratory

CLEMSON UNIVERSITY

Matrix Storage Schemes

Element by Element (EBE)

- Originally developed to reduce memory.
- Provides significant opportunities for fine-grained (vectorized) parallel computation.
- Simpler data structure.
- Requires the preconditioners to be expressed in element level calculation, limiting choices.

Compressed Sparse Row (CSR)

- Lower Memory Requirements O(3nze) Stores only the non zero entries.
- Complicated data structure to assemble the matrix, especially in parallel.
- Can manage to apply different kinds of preconditioners.

$$A = \begin{bmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{bmatrix}$$

$$\begin{aligned} AA &= [\,1.\ 2.\ 3.\ 4.\ 5.\ 6.\ 7.\ 8.\ 9.\ 10.\ 11.\ 12.\,] \\ JA &= [\,1\ 4\ 1\ 2\ 4\ 1\ 3\ 4\ 5\ 3\ 4\ 5\,] \\ IA &= [\,1\ 3\ 6\ 10\ 12\ 13\,] \end{aligned}$$

Computational Mechanics Laboratory

CLEMSON UNIVERSITY

Conjugate Gradient Iterative Algorithm

$$A\underline{x} = \underline{b}$$

```
1. Choose $\underline{x}^0$,    $\underline{r}^0 = \underline{b} - A\underline{x}^0$
2. Solve $M\underline{z}^0 = \underline{r}^0$,    $\underline{p}^0 = \underline{z}^0$
3. $\gamma^0 =$ Inner Product $(\underline{z}^0, \underline{r}^0)$
   Do $k = 0, 1, 2, \ldots k_{max}$
4.     $\underline{q}^k =$ Matrix Multiply $(A, \underline{p}^k)$
5.     $\tau^k =$ Inner Product $(\underline{p}^k, \underline{q}^k)$
6.     $\alpha^k = \gamma^k / \tau^k$
7.     $\underline{x}^{k+1} = \underline{x}^k + \alpha^k \underline{p}^k$,    $\underline{r}^{k+1} = \underline{r}^k - \alpha^k \underline{q}^k$
8.     Solve $M\underline{z}^{k+1} = \underline{r}^{k+1}$
9.     $\gamma^{k+1} =$ Inner Product $(\underline{z}^{k+1}, \underline{r}^{k+1})$
10.    If $(\sqrt{\gamma^{k+1}} \leq$ Tolerance $)$ Exit
11.    $\beta^k = \gamma^{k+1}/\gamma^k$,    $\underline{p}^{k+1} = \underline{z}^{k+1} + \beta^k \underline{p}^k$
   End Do
```

Matrix-Vector Multiplication (1)

Inner Products (2)

Vector update (3)

Preconditioner Solve (1)

Note: Independent of storage format for matrix

Reference: ***Saad Yousef,*** *Iterative Methods for Sparse Linear Systems*

Computational Mechanics Laboratory

CLEMSON UNIVERSITY

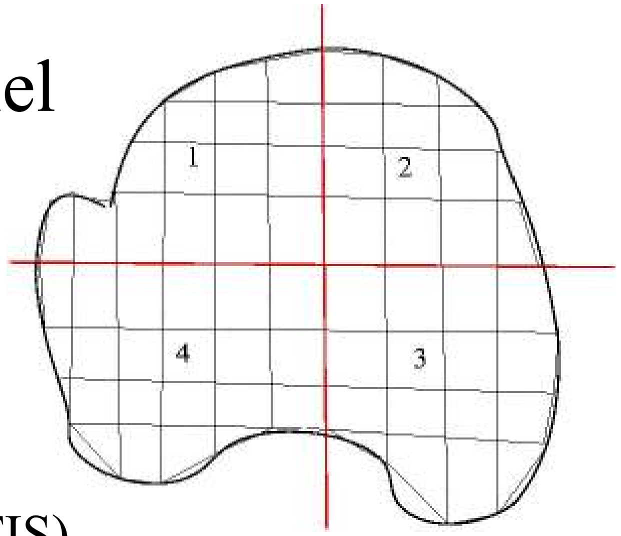
Parallel Solution

To solve the FEA problem in parallel

- Form the system of equations in parallel.
- Store the matrix in distributed manner.
- Solve equations using parallel version of CG.

Steps :

- Mesh the domain.
- Decompose the domain into subdomains (METIS).
- Assign one processor to each subdomain.
- Each processor performs calculation for local elements (CSR requires assembly).
- Interior nodes are local to the processor.
- Interface nodes on subdomain boundaries requires communication (Update).

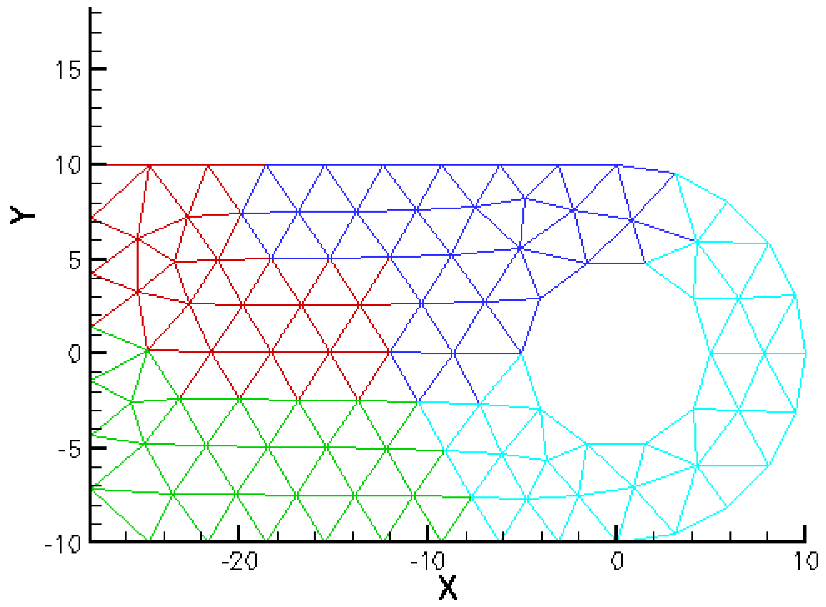


Computational Mechanics Laboratory

CLEMSON UNIVERSITY

Domain Decomposition

The interior nodes are numbered first followed by each of the nodes lying on the subdomain interface. The system of equation then becomes:



$$\begin{bmatrix} A_1 & & & & C_1 \\ & A_2 & & & C_2 \\ & & \ddots & & \vdots \\ & & & A_s & C_s \\ C_1^T & C_2^T & \cdots & C_s^T & A_I \end{bmatrix} \begin{Bmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \vdots \\ \underline{x}_s \\ \underline{x}_I \end{Bmatrix} = \begin{Bmatrix} \underline{b}_1 \\ \underline{b}_2 \\ \vdots \\ \underline{b}_s \\ \underline{b}_I \end{Bmatrix}$$

Example mesh with 4 subdomains

$$A_I = \mathop{\mathbf{A}}_{j=1}^{s} A_{I_j} \quad \text{and} \quad b_I = \mathop{\mathbf{A}}_{j=1}^{s} b_{I_j}$$

Computational Mechanics Laboratory

CLEMSON UNIVERSITY

Block Matrix Structure

For each Process:

$$\begin{bmatrix} A_j & B_j \\ B_j^T & A_{I_j} \end{bmatrix} \begin{Bmatrix} \underline{x}_j \\ \underline{x}_{I_j} \end{Bmatrix} = \begin{Bmatrix} \underline{b}_j \\ \underline{b}_{I_j} \end{Bmatrix}$$

where,

A_j - Assembled local interior node matrix

A_{I_j} - Local interface nodes matrix

B_j - Coupling of the local interior nodes to local interface nodes

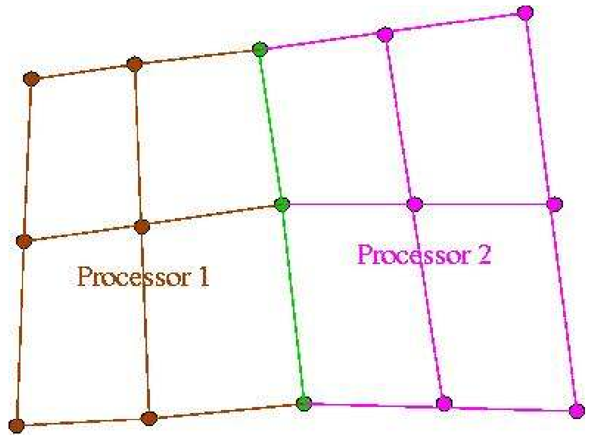
B_j^T - Coupling of the local interface nodes to local interior nodes

$\underline{x}_j$ - Local interior solution subvector

$\underline{x}_{I_j}$ - Local interface solution subvector

$\underline{b}_j$ - Local interior force subvector

$\underline{b}_{I_j}$ - Local interface force subvector



```
Subroutine Update ($\underline{x}$)

- - - - - Send - - - - - - -
      Do $j = 1, np$
            Buf(j) = x(values shared by j and this process)
            MPI Send (Buf, j)
      End do
———— Receive ——————
      Do $j = 1, np$
            MPI_Receive (Buf, j)
            $x = x +$ Buf
      End do

End Subroutine Update
```

Parallel Conjugate Gradient Algorithm

```
For each process j;

1.  $\underline{b}_{I_j} = \text{Update}\ (\underline{b}_{I_j})$
2.  $\underline{x}_j^0 = \underline{0}, \quad \underline{x}_{I_j}^0 = \underline{0}; \quad \underline{r}_j^0 = \underline{b}_j, \quad \underline{r}_{I_j}^0 = \underline{b}_{I_j}$
3.  Solve $M(\underline{z}_j^0, \underline{z}_{I_j}^0) = (\underline{r}_j^0, \underline{r}_{I_j}^0); \quad \underline{p}_j^0 = \underline{z}_j^0, \quad \underline{p}_{I_j}^0 = \underline{z}_{I_j}^0$
4.  $\gamma^0 = \text{Inner Product}\ (\underline{z}_j^0, \underline{z}_{I_j}^0; \underline{r}_j^0, \underline{r}_{I_j}^0)$
    Do $k = 0, 1, 2, \ldots,$ Maximum Iterations
5.      $(\underline{q}_j^k, \underline{q}_{I_j}^k) = \text{Matrix Multiply}\ (A, (\underline{p}_j^k, \underline{p}_{I_j}^k))$
6.      $\tau^k = \text{Inner Product}\ (\underline{p}_j^0, \underline{p}_{I_j}^0; \underline{q}_j^0, \underline{q}_{I_j}^0)$
7.      $\alpha^k = \gamma^k / \tau^k$
8.      $\underline{x}_j^{k+1} = \underline{r}_j^k + \alpha^k \underline{p}_j^k; \quad \underline{x}_{I_j}^{k+1} = \underline{x}_{I_j}^k + \alpha^k \underline{p}_{I_j}^k$
9.      $\underline{r}_j^{k+1} = \underline{r}_j^k - \alpha^k \underline{q}_j^k; \quad \underline{r}_{I_j}^{k+1} = \underline{r}_{I_j}^k - \alpha^k \underline{q}_{I_j}^k$
10.     Solve $M(\underline{z}_j^{k+1}, \underline{z}_{I_j}^{k+1}) = (\underline{r}_j^{k+1}, \underline{r}_{I_j}^{k+1})$
11.     $\gamma^{k+1} = \text{Inner Product}\ (\underline{z}_j^{k+1}, \underline{z}_{I_j}^{k+1}; \underline{r}_j^{k+1}, \underline{r}_{I_j}^{k+1})$
12.     If $(\sqrt{\gamma^{k+1}} \leq \text{Tolerance}\ )$ End
13.     $\beta^k = \gamma^{k+1} / \gamma^k; \quad \underline{p}_j^{k+1} = \underline{z}_j^{k+1} + \beta^k \underline{p}_j^k, \quad \underline{p}_{I_j}^{k+1} = \underline{z}_{I_j}^{k+1} + \beta^k \underline{p}_{I_j}^k$
    End Do
```

Requires pt. to pt. Communication (Update).

Requires Update

Requires one collective communication

Reference: ***Jimack P.K, Touheed N,*** *Developing Parallel Finite Element Software Using MPI*

Computational Mechanics Laboratory

CLEMSON UNIVERSITY

Matrix-by-Vector Multiply

$$\begin{Bmatrix} \underline{q}_j \\ \underline{q}_{I_j} \end{Bmatrix} = \begin{bmatrix} A_j & B_j \\ B_j^T & A_{I_j} \end{bmatrix} \begin{Bmatrix} \underline{p}_j \\ \underline{p}_{I_j} \end{Bmatrix} \qquad j = 1, 2, \ldots, np$$

For each process j:

1. $\underline{q}_j = A_j \underline{p}_j + B_j \underline{p}_{I_j}$
2. $\underline{q}_{I_j} = B_j^T \underline{p}_j + A_{I_j} \underline{p}_{I_j}$
3. Update $(\underline{q}_{I_j})$

```
Subroutine EBEmatmul ($Ke, p_j, p_{I_j}, q_j, q_{I_j}$)
    Do $e = 1, el$
      Do $i = 1, nen$
        If (Interior node)
          Assemble the contributions to $q_j$
        Else If (Interface node)
          Assemble the contributions to $q_{I_j}$
        End If
      End do
    End do
    Update($q_{I_j}$)
End Subroutine EBEmatmul
```

```
Subroutine CSRmatmul ($Acsr, \underline{p}_j, \underline{p}_{I_j}, \underline{q}_j, \underline{q}_{I_j}$)
    $\underline{q}_j = A_j \underline{p}_j + B_j \underline{p}_{I_j}$
    $\underline{q}_{I_j} = B_j^T \underline{p}_j + A_{I_j} \underline{p}_{I_j}$
    Send $\underline{q}_{I_j}$ using MPI Send
    Receive $q_{I_j}$ using MPI Recv
    Add $\underline{q}_{I_j}$
End Subroutine CSRmatmul
```

Inner Product and Diagonal Preconditioner

```
Function InnerProduct ($\underline{a}_j, \underline{a}_{I_j}; \underline{b}_j, \underline{b}_{I_j}$)
        sum = Dot_Product ($\underline{a}_j, \underline{b}_j$)
        n = size($\underline{b}_{I_j}$)
        sum = sum + $\sum_{i=1}^{n} [\underline{a}_{I_j}(i) \times \underline{b}_{I_j}(i)]/\text{Shared}(i)$
       MPI_AllReduce( sum, InnerProduct, MPI_SUM)
End Function InnerProduct
```

```
For all process $j$ ;
Subroutine SetupPreconditioner()
    Do $e = 1, \ldots, el$
    Assemble diagonal of $A_j$ to $\underline{di}_j$
    Assemble diagonal of $A_{I_j}$ to $\underline{d}_{I_j}$
    Update ($\underline{d}_{I_j}$)
    Invert the diagonal vectors $\underline{d}_j = 1/\underline{d}_j$;      $\underline{d}_{I_j} = 1/\underline{d}_{I_j}$
End Subroutine SetupPreconditioner
```

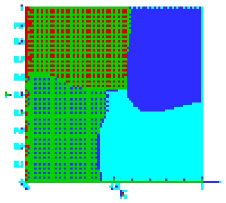
```
Subroutine Solve ($\underline{z}_j, \underline{z}_{I_j}; \underline{r}_j, \underline{r}_{I_j}$)
        $\underline{z}_j = \underline{d}_j * \underline{r}_j$
        $\underline{z}_{I_j} = \underline{d}_{I_j} * \underline{r}_{I_j}$
End Subroutine Solve
```

Implementation

$$\begin{tikzcd}
\text{Mesh Generation} \arrow[d] \\
\text{Mesh Partitioner} \arrow[d] \\
\text{Preprocessing} \arrow[d] \\
\text{Solver} \arrow[d] \\
\text{Visualization}
\end{tikzcd}$$

Mesh Generation: IDEAS, Pro-E, In-House

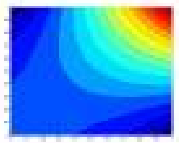
Mesh Partitioner: METIS



Preprocessing: Utility program to read in the partitioned mesh and creates N number of files. N = no of processors

Solver: Fortran90 and MPI

Visualization: Tecplot, Matlab



Numerical Examples

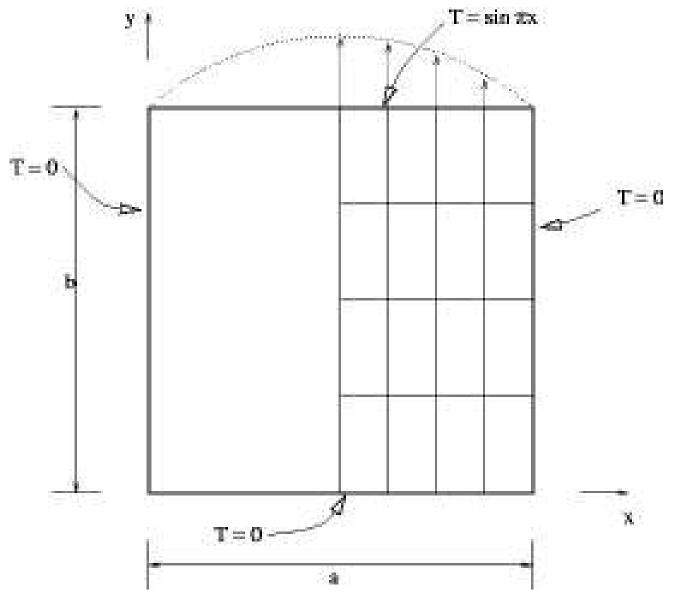
Poisson's Equation with Dirchlet BC

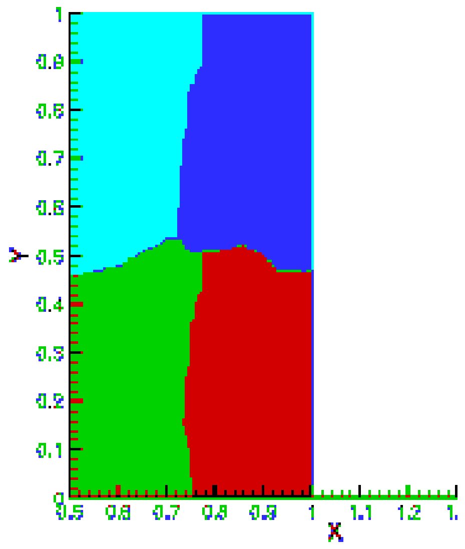
$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=0, \qquad 0<x<1, \qquad 0<y<1$$

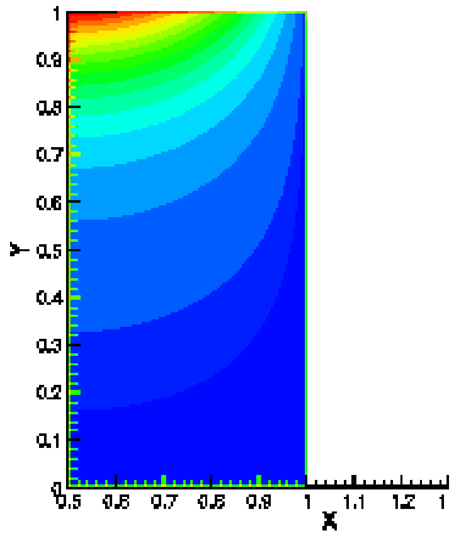
Subject to Dirchlet boundary condition

$$T(x=0)=0$$
$$T(x=1)=0$$
$$T(y=0)=0$$
$$T(y=1)=\sin \pi x$$

Exact solution is : $T(x,y)=\dfrac{\sin \pi x \cdot \sinh \pi y}{\sinh \pi}$







Code Validation

Dimension of the mesh : 200 x 400

Number of DOF: 79,800

Tolerance : 1e-06

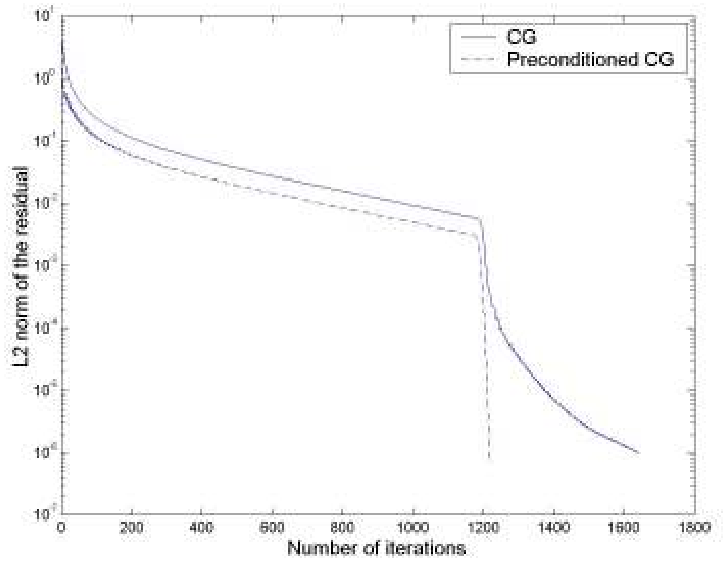
Parallel solution converges to exact solution.

Computational Mechanics Laboratory

CLEMSON UNIVERSITY

Analysis

2D structured mesh: 600 x 600 4-node Quads, 359,400 DOF



Convergence of CG and PCG.

Implemented on

- Sun HPC

  14 UltraSPARCII CPU's

  336MHz

  4MB cache

  4GB shared memory

- SGI Onyx2 Infinite Reality system

  8 MIPS R12000 CPU's

  400 MHz

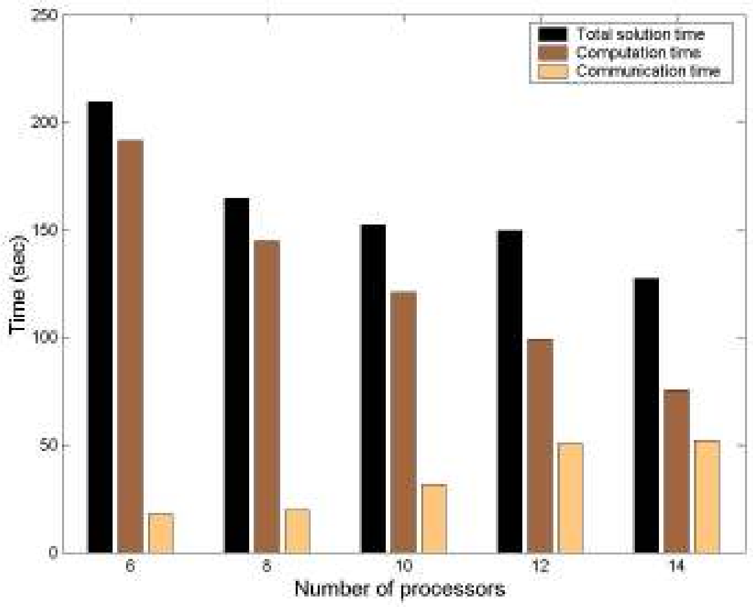
  8MB cache

  8GB shared memory

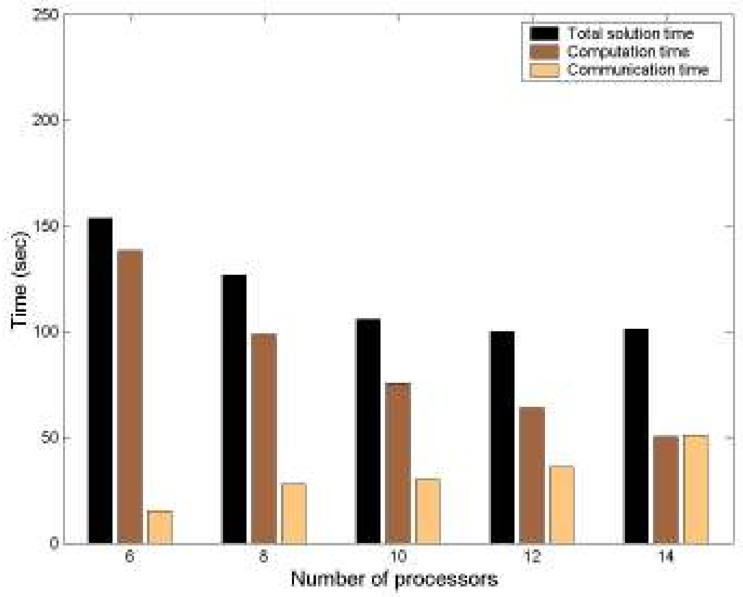
Communication & Computation Time

SUN Machine

EBE



CSR



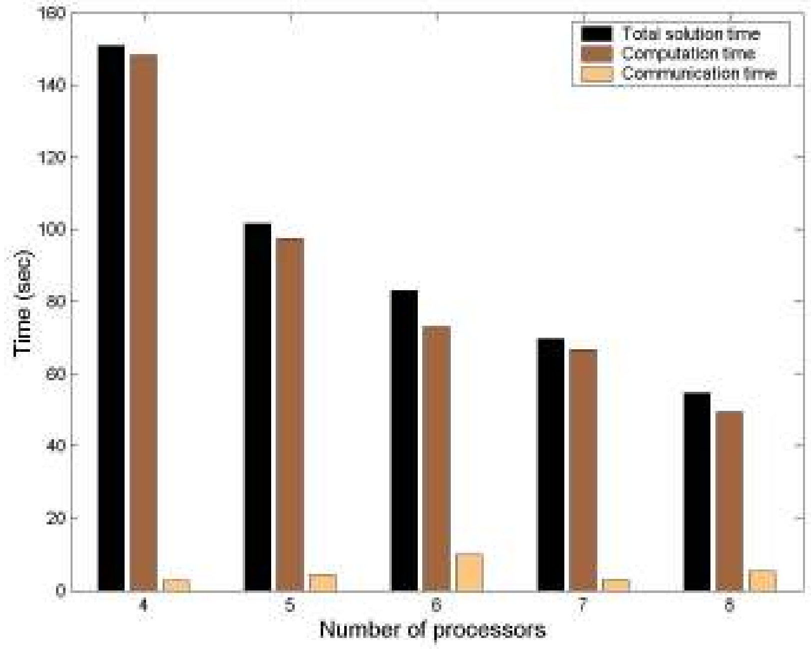
- Communication time increases with increase in number of processors.
- Computation time decreases with the increase in number of processors.

Computational Mechanics Laboratory

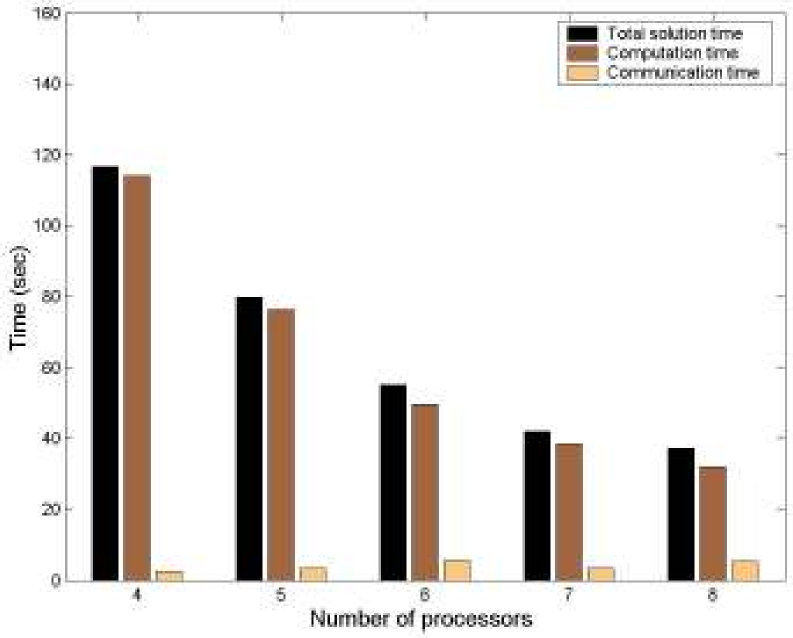
CLEMSON UNIVERSITY

SGI Machine

EBE



CSR



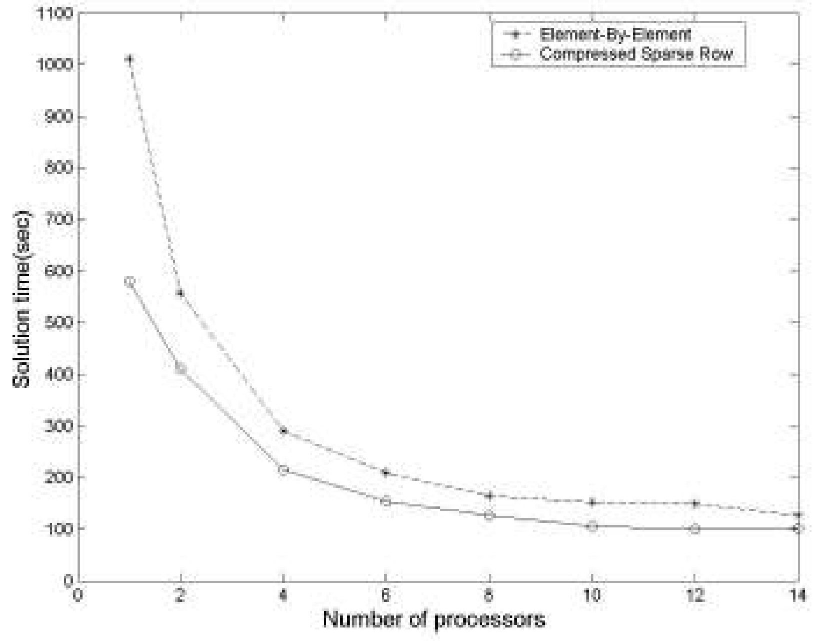
- Communication time is virtually nil.

Computational Mechanics Laboratory

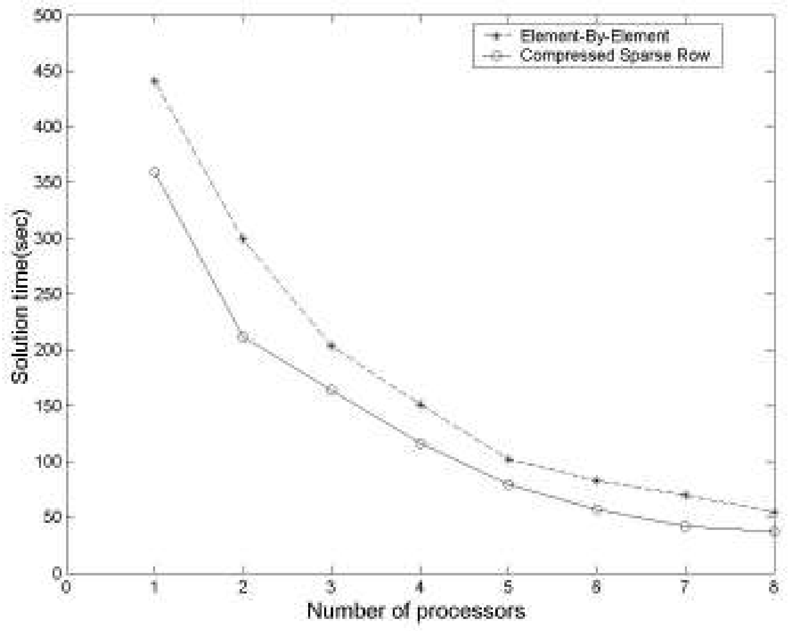
CLEMSON UNIVERSITY

Solution Timings

Sun Machine



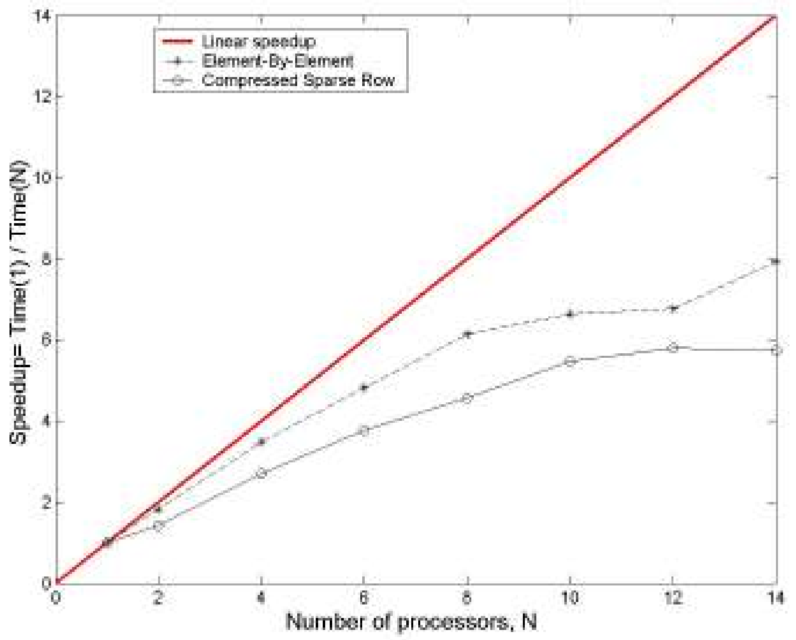
SGI Machine



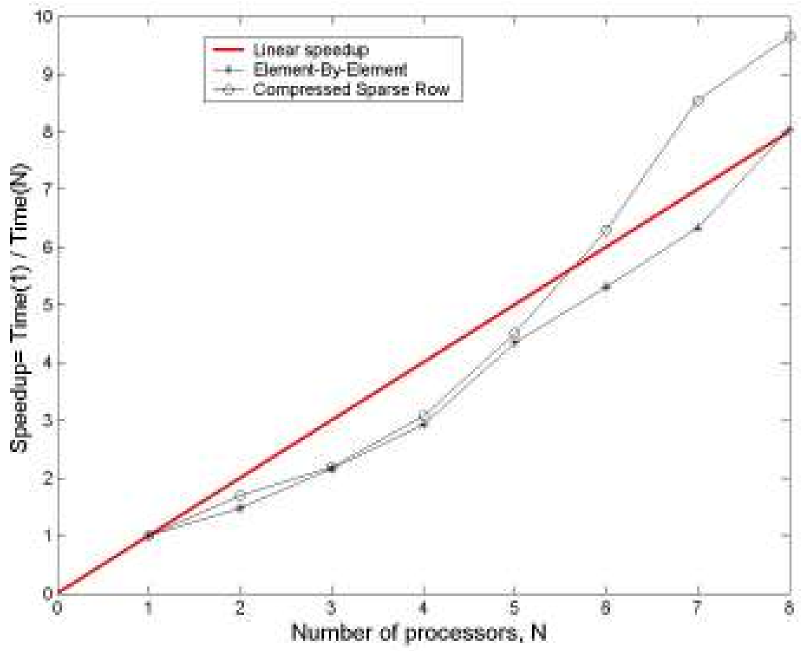
CSR format better than EBE on both machines by almost 30%.

Speedup

Sun Machine



SGI machine

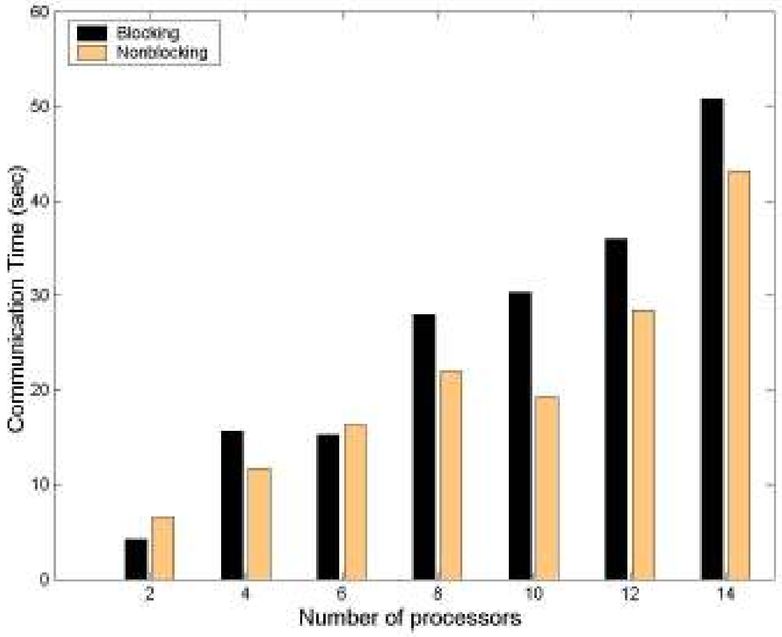


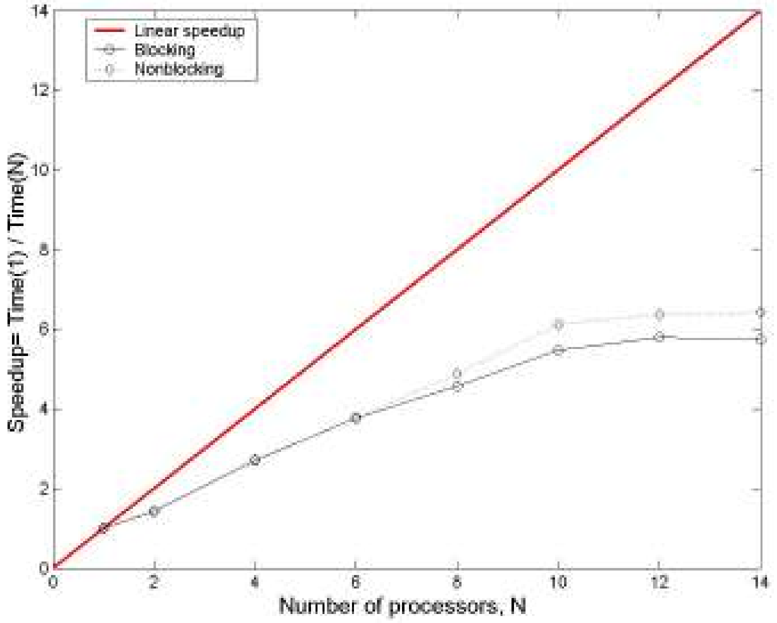
- Sun machine scales well up to 10 processors.
- SGI shows super-linear speedup for CSR.

Computational Mechanics Laboratory

CLEMSON UNIVERSITY

Comparison of Blocking and Nonblocking Communication





Nonblocking saves some communication time as the number of processors increase.

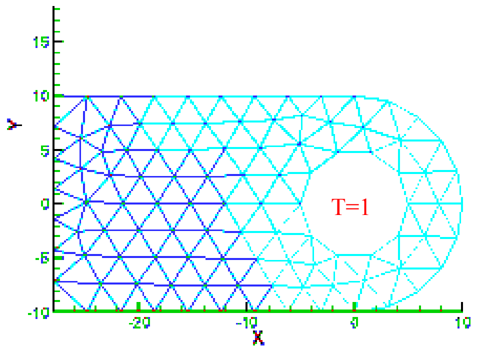
```
Subroutine CSRMatmul
    $\underline{q}_{I_j} = B_j^T \underline{p}_j + A_{I_j} \underline{p}_{I_j}$
    Send $\underline{q}_{I_j}$ using MPI_ISend
    $\underline{q}_j = A_j \underline{p}_j + B_j \underline{p}_{I_j}$
    Receive $\underline{q}_{I_j}$ using MPI_Recv
    Add $\underline{q}_{I_j}$
Subroutine CSRMatmul
```

Computational Mechanics Laboratory

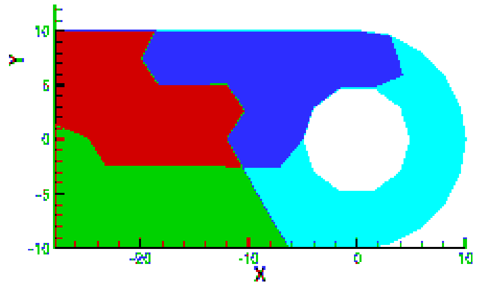
CLEMSON UNIVERSITY

Parallel code validation for unstructured Meshes – 2D

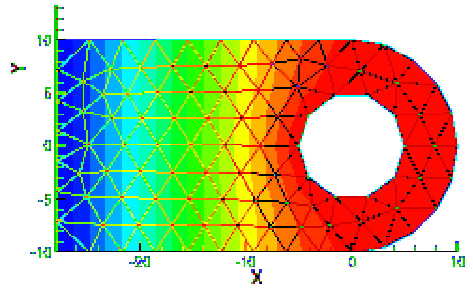
T=0



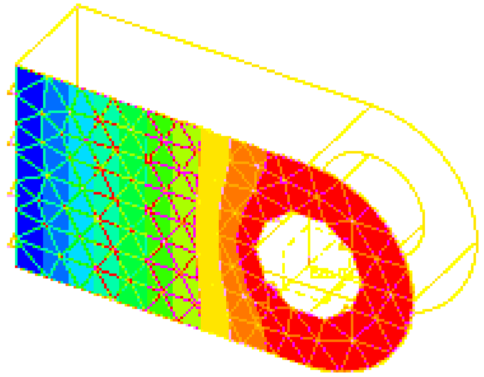
Unstructured Triangular Mesh generated In I-DEAS



The Mesh is Partitioned using METIS



Solved in Parallel using 4 Processor and Visualized using Tecplot

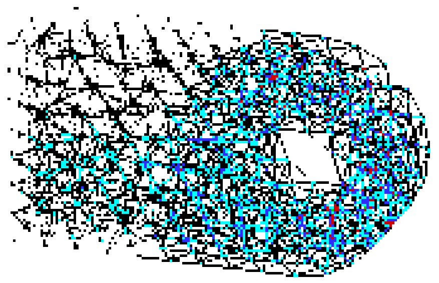


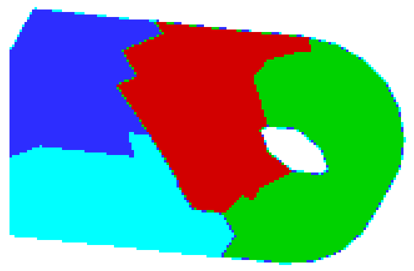
I-DEAS SOLUTION

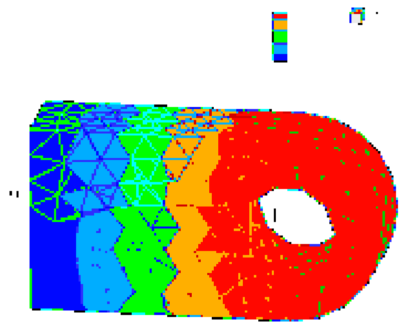
Computational Mechanics Laboratory

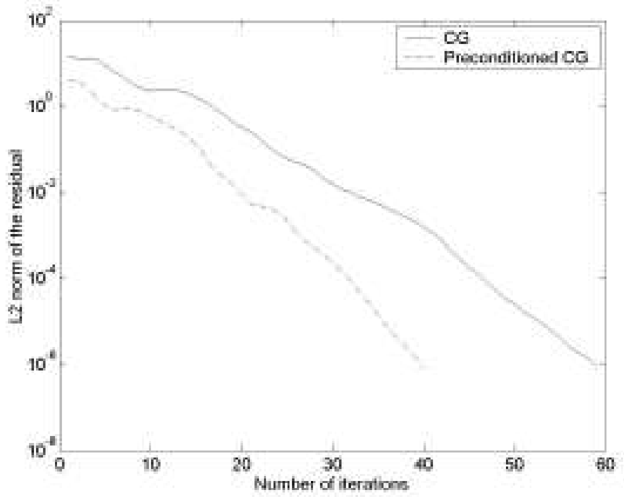
CLEMSON UNIVERSITY

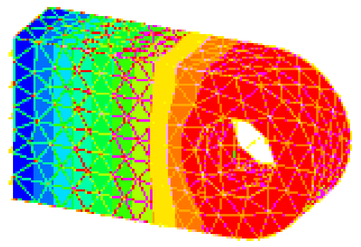
Parallel code validation for unstructured Meshes – 3D


Unstructured Tetrahedra Mesh generated In I-DEAS


The Mesh is Partitioned using METIS


Solved in Parallel using 4 Processor and Visualized using Tecplot




I-DEAS SOLUTION

Computational Mechanics Laboratory

CLEMSON UNIVERSITY

Conclusions

- CSR format takes 30% less CPU time than EBE.
- SGI with optimized vendor supplied MPI, shows super-linear speedup for the problem tested.
- Sun machine scales well up to 10 processors.
- By overlapping communication and computation total solution time can be reduced.
- Diagonal preconditioner moderately accelerates the convergence of the CG solver.

Computational Mechanics Laboratory

CLEMSON UNIVERSITY

Future Work

- Perform scalability analysis on large 3D problems.
- Scalability analysis on Beowulf cluster.
- Implement Schur Complement method.
- Implement better preconditioner.
- Generalize to non-symmetric and non positive definite systems. Other iterative methods such as Bi-Conjugate Gradient Stabilized(Bi-CGS), QMR and GMRES.

Computational Mechanics Laboratory

CLEMSON UNIVERSITY