Viability of Krylov Subspace Methods for Solving Dense Linear Systems

Nitish Keskar : 081020028

Guide : Prof. S. M. Gunadal
Veermata Jijabai Technological Institute (VJTI)

April 27, 2012

Table of Contents

Why Dense Matrices?

- Not everything is sparse
- Photonics, Fluid Networks, Heat Transfer, Medical Imaging
- Sparse solvers are already quite *advanced*
- Dense Linear Algebra is needed but not researched as much.
- Krylov Subspace Methods have never been used for dense matrices.

Elementary Linear Algebra Subroutines

- Cond(A)

$$Ax = b \implies A(x + \delta x) = b + \delta b \implies Ax + A\delta x = b + \delta b \implies A\delta x = \delta b \implies \delta x = A^{-1}\delta b$$

Consider 2 matrices M and N.

$$M = \begin{bmatrix} 1 & 1 \\ 1 & 1.0001 \end{bmatrix} \qquad N = \begin{bmatrix} 0.0001 & 1 \\ 1 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{1}$$

Solution :

$$x_m = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad x_n = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Now, if b were to be changed slightly,

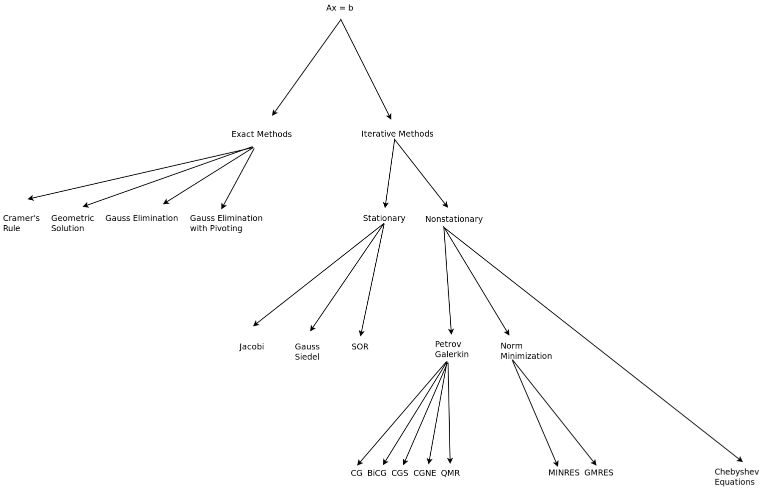
$$M = \begin{bmatrix} 1 & 1 \\ 1 & 1.0001 \end{bmatrix} \qquad N = \begin{bmatrix} 0.0001 & 1 \\ 1 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 1.1 \\ 1 \end{bmatrix} \tag{2}$$

Solution :

$$x_m = \begin{bmatrix} 1001.1 \\ -1000.0 \end{bmatrix} \qquad x_n = \begin{bmatrix} -0.1 \\ 1.1 \end{bmatrix}$$

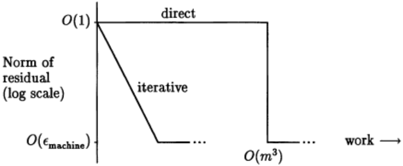
- Positive Definite : $x^T A x > 0 \qquad \forall x \in \mathbb{R}^N$

Classification of Methods to Solve $Ax = b$



Problem with $O(N^3)$

Thumbnail history of matrix computations over the years:
1950: m = 20 (Wilkinson)
1965: m = 200 (Forsythe and Moler)
1980: m = 2000 (UNPACK)
1995: m = 20000 (LAPACK)
In the course of forty-five years, the dimensions of tractable matrix problems have increased by a factor of 10^3. This progress is impressive, but it pales beside the progress achieved by computer hardware in the same period-a speedup by a factor of 10^9, from flops to gigaflops.



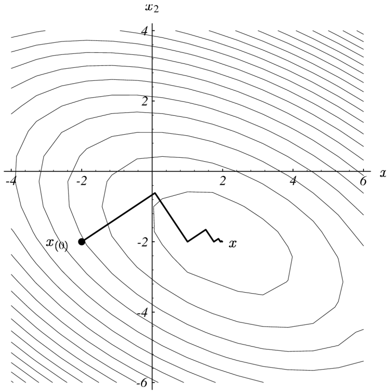
(Rough) Perspective into Sequential $O(N^3)$

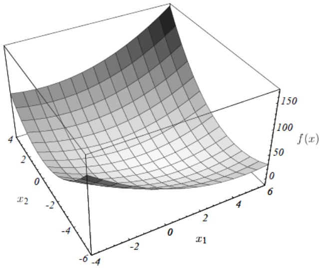
$$
\begin{aligned}
Let \qquad n &= 10^6, \\
Steps &= \frac{n^3}{3} \\
&= \frac{(10^6)^3}{3} \\
&= \frac{10^{18}}{3} \\
&\approx 3 \times 10^{17}
\end{aligned}
$$

Assuming a modest 100 GFlop machine,

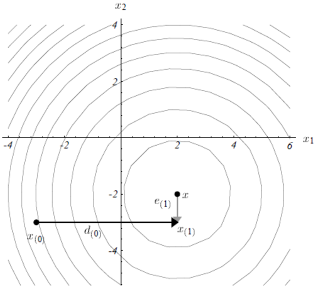
$$
\begin{aligned}
T_{Required} &= \frac{3 \times 10^{17}}{2 \times 10^{11}} \\
&= 1.5 \times 10^6 \quad seconds \\
&= \frac{1.5 \times 10^6}{86400} \quad days \\
&\approx 17 \quad days
\end{aligned}
$$

Intuitive Introduction to Krylov Subspace Methods - Steepest Descent





- Conjugate Gradients



- QMR
- GMRES

Algorithm for Generating Matrices

To get a generate a matrix with condition number c ,
Let D be a diagonal matrix whose diagonal consists of the desired eigenvalues of the matrix.
Using a similarity transformation, via Householder transformations, a matrix is formed as : $A := (I - tuu^T)D(I - tuu^T)$, where $t = 2/u^T u$.
To form this matrix with $O(n^2)$ operations,

- $v := Du$, where $u \in \mathbb{R}^N$, $u_i = 1.0 \; \forall \; i$
- $s := \dfrac{t^2 u^T v}{2}$
- $w := tv - su$
- $A = D - uw^T - wu^T$.

Numerical Experiments

- The matrices are generated uniquely by passing the values of the desired eigenvalues.
- The matrices are necessarily symmetric.
- For GMRES, the implementation was restart based with a restart value at 20.
- The algorithms are implemented using OpenMP threading for shared memory system.
- The time graphs represent the multithreaded elapsed time with dynamic thread allocation.
- All systems were solved without the use of preconditioners for a single RHS.
- The tolerance criterion was set to tolerance less than 1E-6.
- While the matrices described are of size 5000×5000, tests run on matrices with sizes 10000×10000 have exhibited consistent results.

Linear Distribution

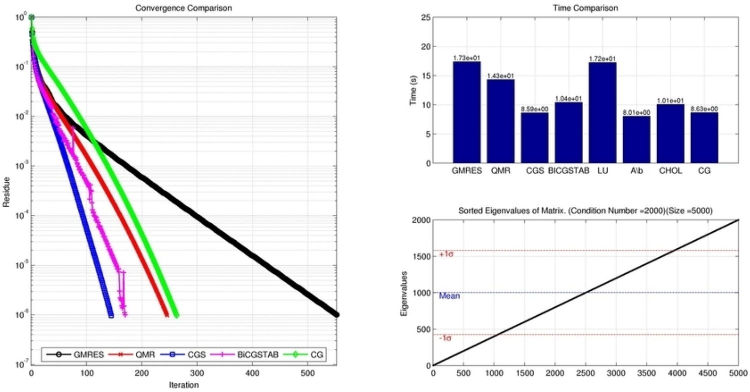


Figure: Linear Distribution from 1 to 2000

Linear Distribution with Weights

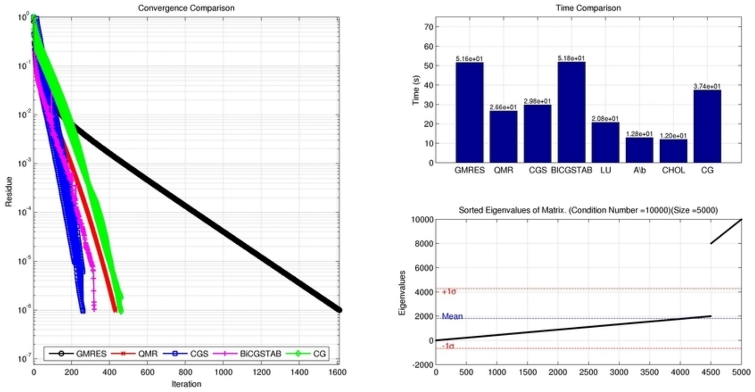


Figure: GMRES Problem with Linear Distribution. Ratio = 9:1

Clusters

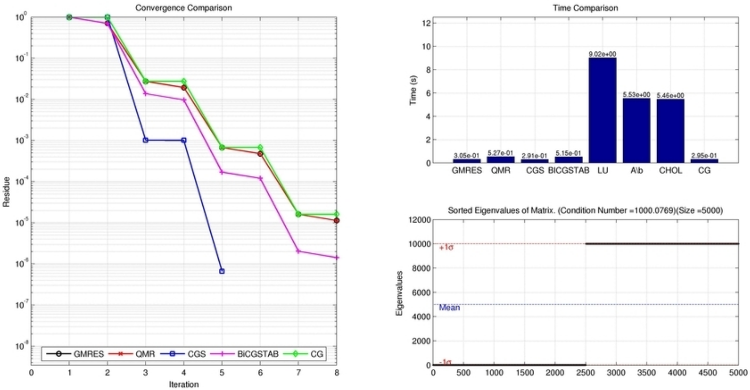


Figure: 2 clusters at Centres 10 and 10000

GMRES Restart

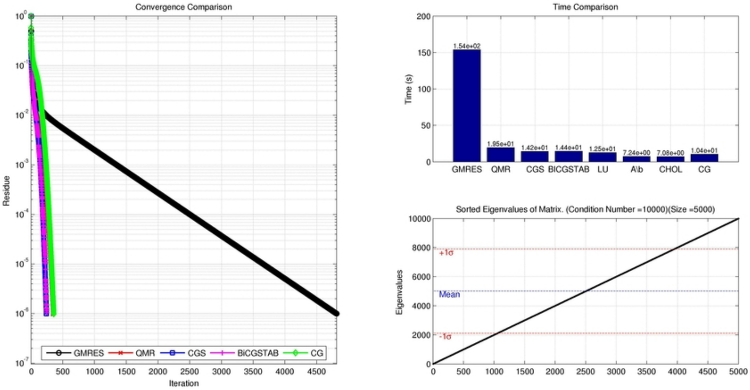


Figure: GMRES with Restart = 10

GMRES Restart

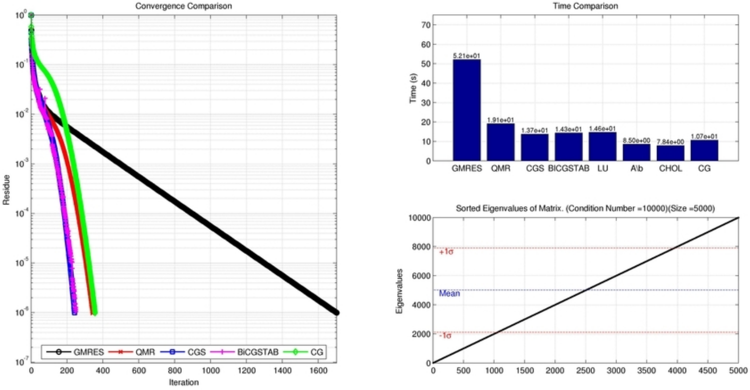


Figure: GMRES with Restart = 30

GMRES Restart

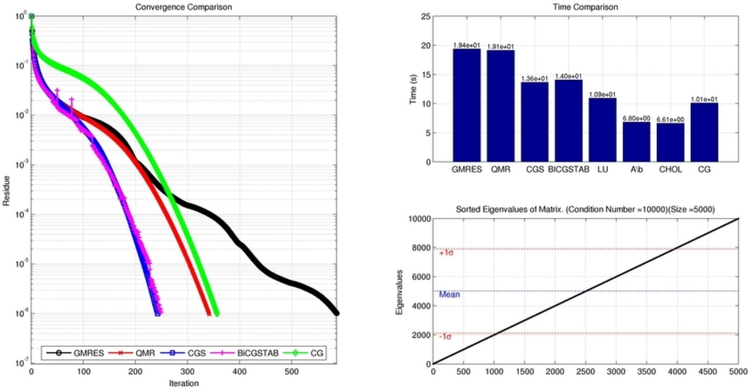


Figure: GMRES with Restart = 100

GMRES Restart

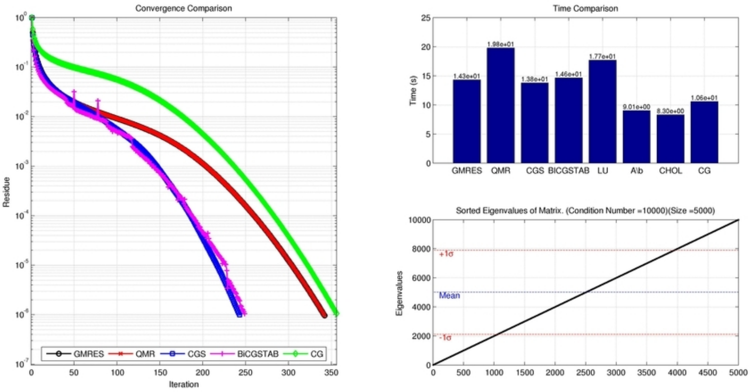


Figure: GMRES with Restart = 500

Outliers

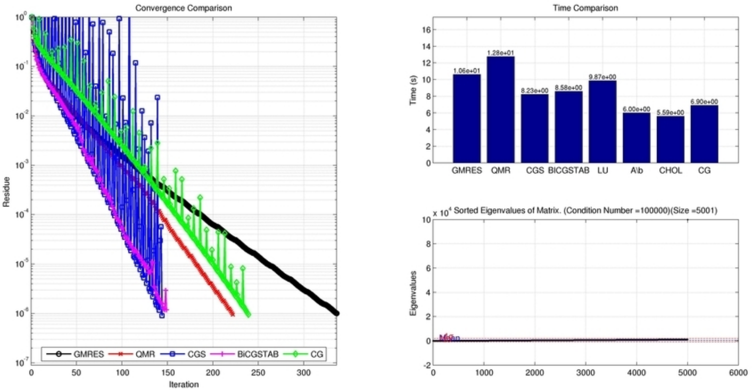


Figure: Linear Distribution from 1 to 1000 with an outlier at 100000

3 Clusters

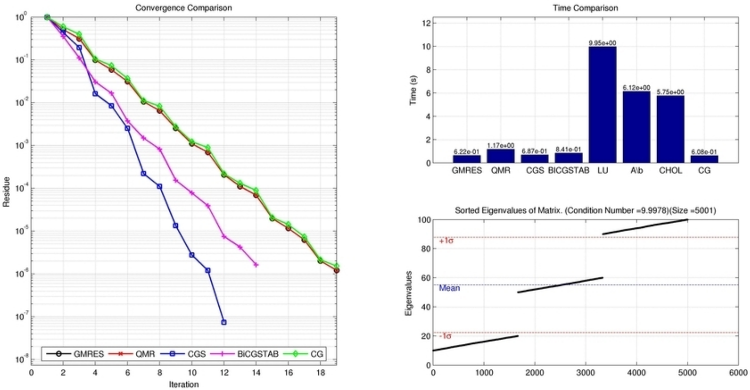


Figure: 3 Clusters with Centres at 15, 55 and 95 and Radius 10

Small Eigenvalues

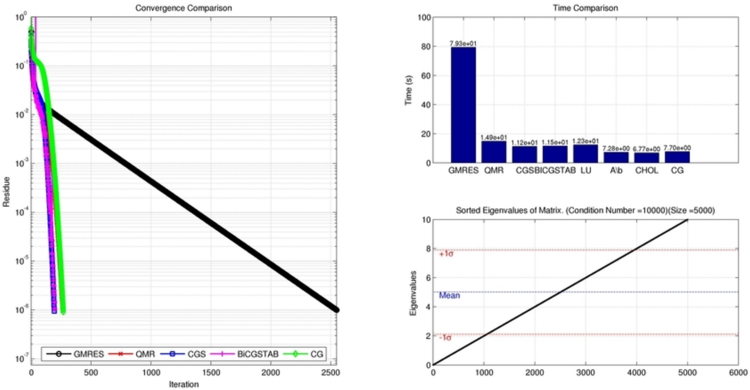


Figure: $\min(\text{eig}(A)) = 0.01$

Linear Symmetric

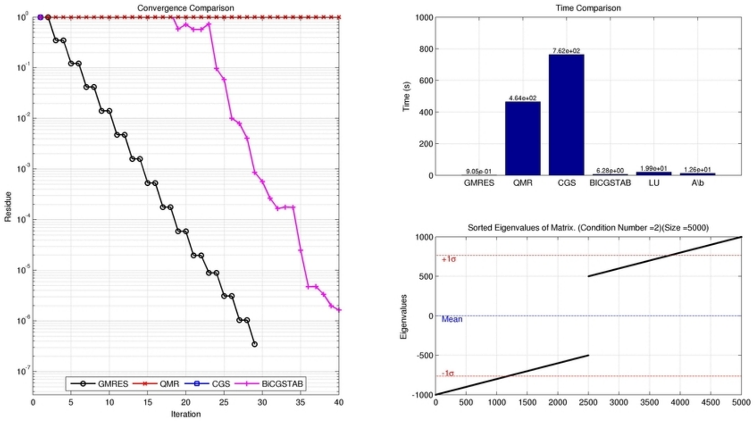


Figure: Linear (Symmetric) from 500 to 1000.

3 Clusters

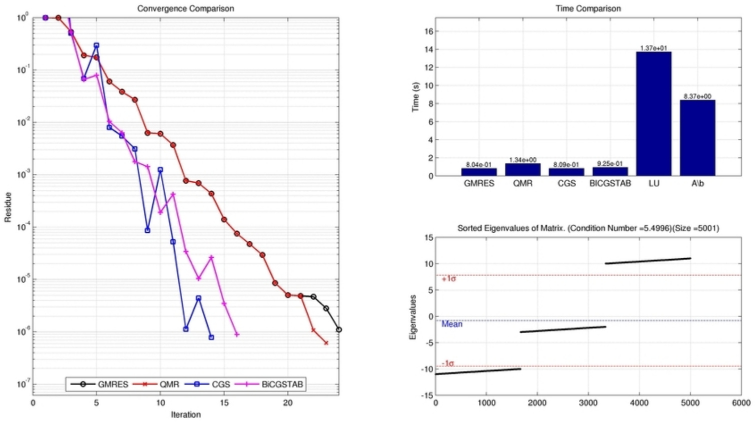


Figure: 3 Clusters with Centres at ±10 and -2 with Radius 1

Randomly Distributed Ill Conditioned Matrices

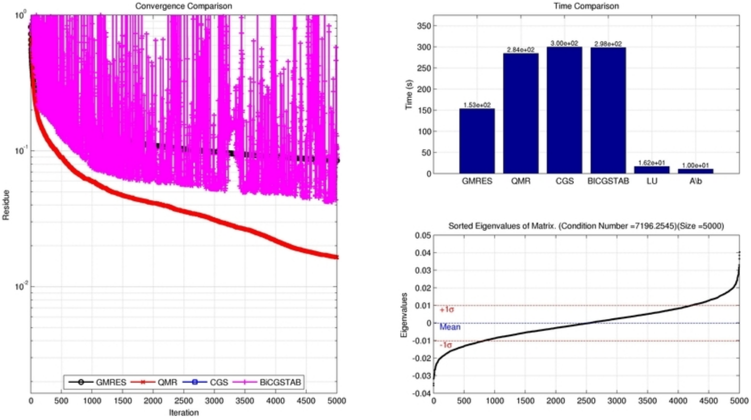


Figure: Mean $= 0$, STD $= 0.01$

Nitish Keskar : 081020028

Small Eigenvalues

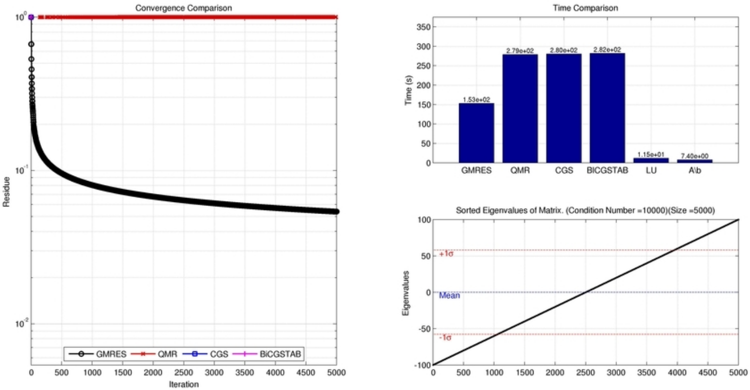
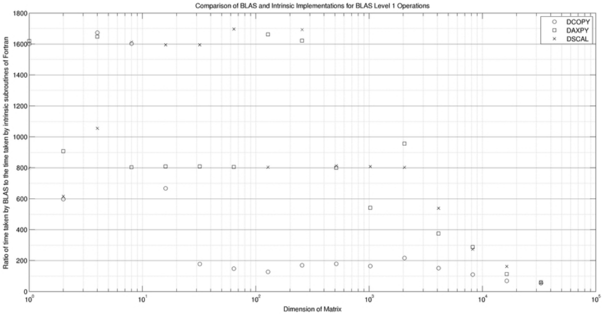


Figure: $\min(|\mathrm{eig}(A)|) = 0.01$

BLAS Helps

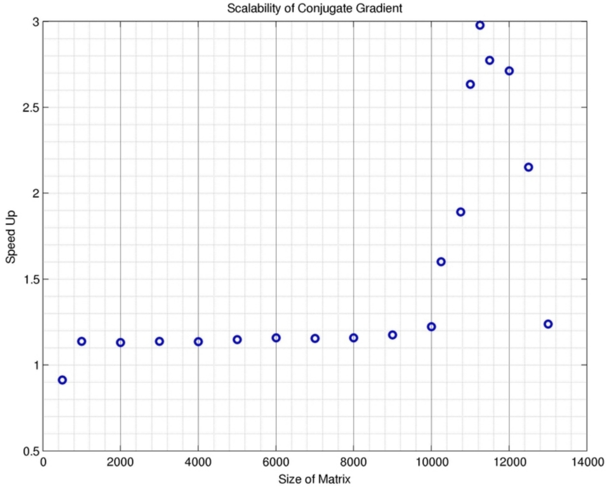
- Basic Linear Algebra Subroutines
- Intel MKL, GoTo, ACML, ATLAS, ESSL etc.
- 3 Levels : *Vector, Matrix-Vector, Matrix-Matrix*
- As high as $1000\times$ faster.
- Multithreaded

Blind BLAS Doesn't Help

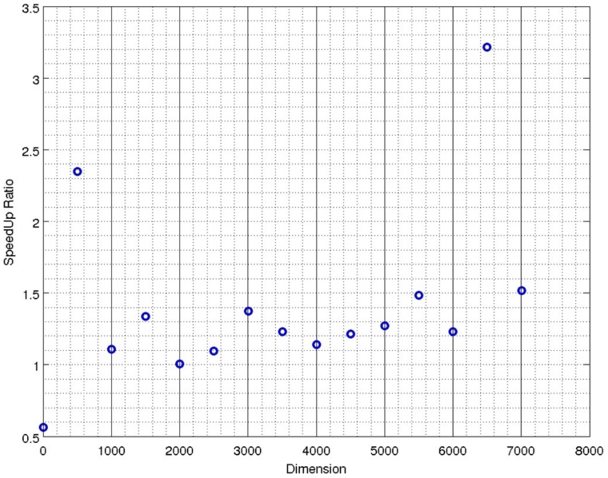


DAXPY(n,alpha,x,1,y,1)	y = alpha*x + y
DSCAL(b,alpha,a,1)	a = alpha*a
DCOPY(n,a,1,b,1)	b = a

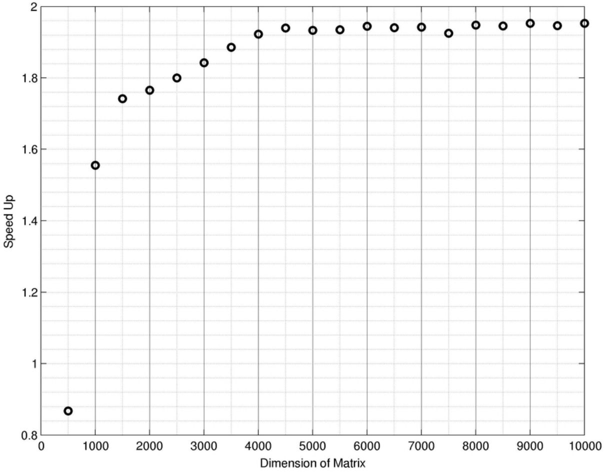
Scalability of CG



Scalability of QMR



Scalability of DGESV



Programming Languages and Paradigms

- Fortran
- C
- C++
- Python
- OpenMP v/s Pthreads
- MPI

Hello World!

Pthreads

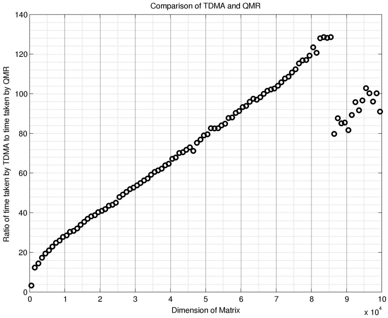
```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
   long tid;
   tid = (long)threadid;
   printf("Hello World! It's me, thread #%ld!\n", tid);
   pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc;
   long t;
   for(t=0;t<NUM_THREADS;t++){
     printf("In main: creating thread %ld\n", t);
     rc = pthread_create(&threads[t],
     NULL, PrintHello, (void *)t);
     if (rc){
       printf("ERROR; return code from
       pthread_create() is %d\n", rc);
       exit(-1);
       }
     }

   pthread_exit(NULL);
}
```

OpenMP

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
{
int nthreads, tid;
#pragma omp parallel private(nthreads, tid)
  {
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);
  /* Only master thread does this */
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }
  }
}
```

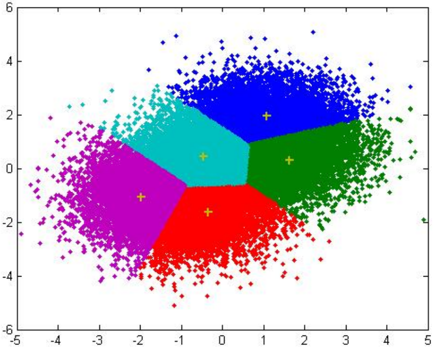
Computational Fluid Mechanics

Consider a simple case of 1D heat transfer in a slab with constant thermal conductivity, the resultant matrix is tridiagonal. The method used to solving such systems is either TDMA or Gauss-Seidel.



Data Mining

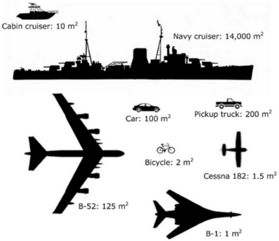
- k-means



- Nearest neighbour

Computational Electromagnetics

- Radar Cross Section (RCS)



- Stealth, Antenna Design
- FDTD, Integral Equations
- $Ax = b$, A is the (dense) impedance matrix, x is the unknown vector of amplitudes and b is the excitation vector.

Statistics

- Curve Fitting $f(x) = a_0x^0 + a_1x^1 + a_2x^2 + \ldots + a_nx^n$

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^n \\ & \vdots & \ddots & & \vdots \\ 1 & & & & \\ 1 & x_m & x_m^2 & \ldots & x_m^n \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix}}_{b} ; x = (A^TA)^{-1}A^Tb$$

- Shopping Recommendations

With Sparse Systems

- Dense subportion of Sparse Matrices
- Development of Sparse Solvers

Many More...

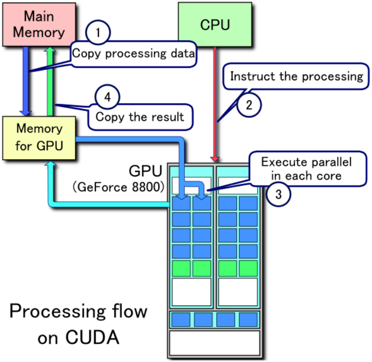
- Medical Imaging
- Material Science
- n-body Simulations
- Grand Challenges

$\vdots$

Future Scope

There are 2 major directions to proceed:

- GPU and Hybrid Computing
- Preconditioners



- Not all systems have a "good" spectrum.
- Using preconditioners, any linear system can be converted to one having a good structure.

Example :

$$A = \begin{bmatrix} 0.6551 & 0.4984 & 0.5853 \\ 0.1626 & 0.9597 & 0.2238 \\ 0.1190 & 0.3404 & 0.7513 \end{bmatrix} ; eig(A) = \begin{bmatrix} 1.3665 \\ 0.4732 \\ 0.5264 \end{bmatrix}$$

Use a preconditioner $K = \begin{bmatrix} 1.9000 & -0.5000 & -1.3000 \\ -0.3000 & 1.2000 & -0.2000 \\ -0.2000 & -0.5000 & 1.6000 \end{bmatrix}$

For the system $KAx = Kb \iff A'x = b'$,

$$A' = \begin{bmatrix} 1.0087 & 0.0245 & 0.0235 \\ -0.0252 & 0.9341 & -0.0573 \\ -0.0219 & -0.0349 & 0.9731 \end{bmatrix} ; eig(A') = \begin{bmatrix} 0.9169 \\ 0.9995 + 0.0035i \\ 0.9995 - 0.0035i \end{bmatrix}$$

Acknowledgements

I wish to thank:

- Prof. S. M. Gunadal (Project Guide) and Dr. M. A. Dharap (HOD of Mechanical Dept.), for allowing me to work on an *unconvential* project.
- Prof. M. Venkatapathi and my labmates at Computational Photonics Laboratory in SERC, IISc for their guidance. Supercomputing would not have been possible without the high performance workstations they provided me access to.

Thank You

Any Questions?