HYBRID PARALLELISED FRAMEWORK FOR THE SOLUTION OF PDEs ON HPC CLUSTERS

MAJOR TECHNICAL PROJECT (DP 401P)

to be submitted by

RITWIK SAHA(B16110)
NIKHIL GUPTA(B16023)

for the

END-SEMESTER
EVALUATION

under the supervision of

DR. GAURAV BHUTANI

SCHOOL OF COMPUTING AND ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MANDI

KAMAND-175005, INDIA

NOVEMBER, 2019

Contents

Abstract

Computational Fluid Dynamics is a branch of fluid mechanincs which studies the flow of liquids and gas and other fluid flows using intensive numerical analysis for solving partial differential equations. Solutions of partial differential equations include substantial amount of solving linear equations of the format $Ax = b$. Various iterative methods are used to solve these equations for computational feesibility. With the recent advancements of hardware in the field of supercomputing on HPC (High perormance Computing) clusters, newer and faster algorithms and tools are required to use the resources and the improved compute power available. Similarly, the above stated linear equations are solved on such HPC clusters. This project aims at implementing a framework for solving partial differential equations on HPC clusters. Even though few frameworks already exist, but they have limitations and thus scope of improvement. This project aims at understanding and improving PETSc library to simultaneously use mulricore-CPU and GPU resources instead of stand alone implementations of them.

Keywords: CFD, PDE , HPC, PETSc

Work done in I-phase

1.1 Background and Literature Survey

1.1.1 Need for solving linear equations

In scientific applications, finite difference methods are used to solve differential equations. We consider the case of first order differentiation

Forward Difference

$$\frac{du(x)}{dx} \approx \frac{\Delta u(x)}{\Delta x} = \frac{u(x+h)-u(x)}{h} = \frac{u^{i+1}-u^{i}}{h}$$

Backward Difference

$$\frac{du(x)}{dx} \approx \frac{\Delta u(x)}{\Delta x} = \frac{u(x)-u(x-h)}{h} = \frac{u^{i}-u^{i-1}}{h}$$

Centered Difference

$$\frac{du(x)}{dx} \approx \frac{\Delta u(x)}{\Delta x} = \frac{u(x+h)-u(x-h)}{2h} = \frac{u^{i+1}-u^{i-1}}{2h}$$

In case of double differential, the formula turns out to be following, where h is the tolerable step.

$$\frac{d^2u(x)}{dx^2} \approx \frac{u(x+h)-2u(x)+u(x-h)}{h^2} = \frac{u^{i+1}-2u^{i}+u^{i-1}}{h^2}$$

Let's consider an equation of non-tubulent flow in 1-dimension, $\nabla^2\phi = 0$, where ϕ is velocity potential at a point in the liquid. Using the above formulae we can rewrite the equation as

$$\frac{\phi^{i+1}-2\phi^{i}+\phi^{i-1}}{h^2} = 0$$

When we convert this equation into matrix, for solving, we get:

$$\begin{bmatrix} \ddots & & & & & \\ & -1 & 2 & -1 & & \\ & & -1 & 2 & -1 & \\ & & & & & \ddots \end{bmatrix} \begin{bmatrix} . \\ \phi^{i-1} \\ \phi^{i} \\ \phi^{i+1} \\ . \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The boundary condition decides whether the matrix would be circulant or teoplitz or any other type of matrix. This is a sytem of equation summarized in a matrix $Ax = b$ where we need to solve for x.

1.1.2 Computationally solving PDE

The traditional method of solving for x in $Ax = b$ involves inverting the matrix A and obtaining x as $x = A^{-1}b$. Inverting the matrix A is computationally expensive for large matrixes. Moreover, matrixes A in our traget equations are usually sparse matrixes. Thus computationally faster and efficient algorithms known as iterative methods are used. Though not 100% accurate, such algorithms provide us the solution in the desirable time within desirable tolerance.

Few examples of such iterative methods are

- Jacobi Method
- Gauss-Seidel Method
- GMRES(Generalized minimal residual method)

Moreover, several algorithms exist for speicific matrix types A such as diagoanal, scalar, upper traingular, lower trainglar etc.

1.1.3 Steepest Descent Method

We would discuss steepest descent method, an iterative method and scope of parallelization in it.

We start with an initial guess x_0 and with each iteration, we improve our guess. We aim to minimize $||Ax - b||^2$. Let $F(x) = ||Ax - b||^2$. Then with each successive step we obtain:

$$x_{i+1} = x_i - \alpha \nabla F(x)$$

Calculating $\nabla F(x)$:

$$F(x) = ||\mathbf{Ax} - \mathbf{b}||^2$$
$$F(x) = \sum_i ||A_{ij}x_j - b_i||^2$$
$$\frac{\partial F(x)}{\partial x_k} = \sum_i \frac{\partial(A_{ij}^2 x_j^2 - 2A_{ij}x_j b_i + b_i^2)}{\partial x_k}$$
$$\frac{\partial F(x)}{\partial x_k} = \sum_i 2A_{ij}(A_{ij}x_j - b_i)\delta_k^j$$
$$\frac{\partial F(x)}{\partial x_k} = \sum_i 2A_{ik}(A_{ik}x_k - b_i)$$
$$\nabla F(x) = 2\mathbf{A^T}(\mathbf{Ax} - \mathbf{b})$$

As we can see, that there is a lot of scope of parallelization in calculating $\nabla F(x)$, as it requires matrix multiplication and element-wise subtraction. Similarly, the step $x_{i+1} = x_i - \alpha \nabla F(x)$ also involves element-wise subtraction and thus can be parallelized.

1.1.4 Conjugate Gradient Method

On the similar lines of Steepest Descent Method, we discuss another method, Conjugate Gradient Method minimizes the function $F(x) = \frac{1}{2}x^T Ax - x^T b$. The Algorithm is as follows:

$$\begin{aligned}
\textbf{Compute } r_0 &= Ax_0 - b, p_0 = -r_0 \\
\textbf{For } k = 0, 1, 2, ... \textbf{ until convergance} \\
\alpha_k &= \frac{r_k^T r_k}{p_k^T A p_k} \\
x_{k+1} &= x_k + \alpha_k p_k \\
r_{k+1} &= r_k + \alpha_k A p_k \\
\beta_k &= \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \\
p_{k+1} &= -r_{k+1} + \beta_k p_k \\
\textbf{End}
\end{aligned} \tag{1.1}$$

1.1.5 Use of Hybrid Cluster

Modern day supercomputers are a hybrid cluster of multiprocesssor CPU nodes (CPU unit containing multiple physical and logical processors) and GPUs (Graphic Proceesing Unit).

MPI (Message passing interface) is the standard parallel library which is used to communicate between these CPU nodes. OpenMP(Open Multi-Processing), POSIX pthreads or MPI is used to share the workload (and hence parallelize) between the multiple cores present on a single CPU node (shared memory architecture). Further, NVIDIA parallel framework CUDA(Compute Unified Device Architecture) is used to implement GPU programming.

The recent advancements in High Performance Computing and the incorporation of above stated architectures has massively increased the available compute power and hence reduced the time taken to run extensive and lengthy calculations. Like any other computational problem, PDEs are also solved on such clusters using various tools and frameworks.

1.1.6 Existing Frameworks

The focus of this project will be in and around an open source framework known as PETSc(Portable, Extensible Toolkit for Scientific Computation). PETSc is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, and GPUs through CUDA or OpenCL, as well as hybrid MPI-GPU parallelism

The incorporation of GPU parallelism in PETSc is relatively new and under development and has been the primary morivation of this project. In current implementations of solving PDE using specific algorthims and frameworks such as PETSc, few observations can be made

- The parallel architecture is usually fixed.
- MPI is used to distribute the work between CPU nodes and PETSc or native CUDA code is used to speed up the calculation using GPU. As a result, the multiple cores present on CPU nodes are hardly used and resources are wasted.
- Even if an implementation does use shared memory architecture, there is always a question of choice between POSIX pthreads and MPI. OpenMP is hardly used.

1.2 Objective and scope of the Work

We aim to provide a tool for solving PDEs using hybrid clusters of multiprocessor CPU nodes and GPU units. The tool will be scalable to distribute work according to the resources available, that is, the architecture need not be fixed. Since GPU parallelism in PETSc is under development, we aim to improve the library for further speedup and efficiency. We aim to incorporate shared memory parallelization and decide which framewor/library to use.

We aim to implement mldivide algorithm of matlab parallaly or improve the existing libraries to incorporate CPU-GPU hybid solving techniques. A brief flowchart of mldivide algorithm which solves $x = A\backslash b$ is given in Figure 1.1.

1.3 Methodology

- Understand the various parllel iterative methods for solving linear inequations $Ax = b$.
- Get hands on experience in writing scalable parallel codes aimed at HPC clusters.
- Uderstanding the PETSc library and observing various benchmarkings to detect scope of improvement.
- Use OpenACC (and preferably CUDA) to bring about the relevant changes.

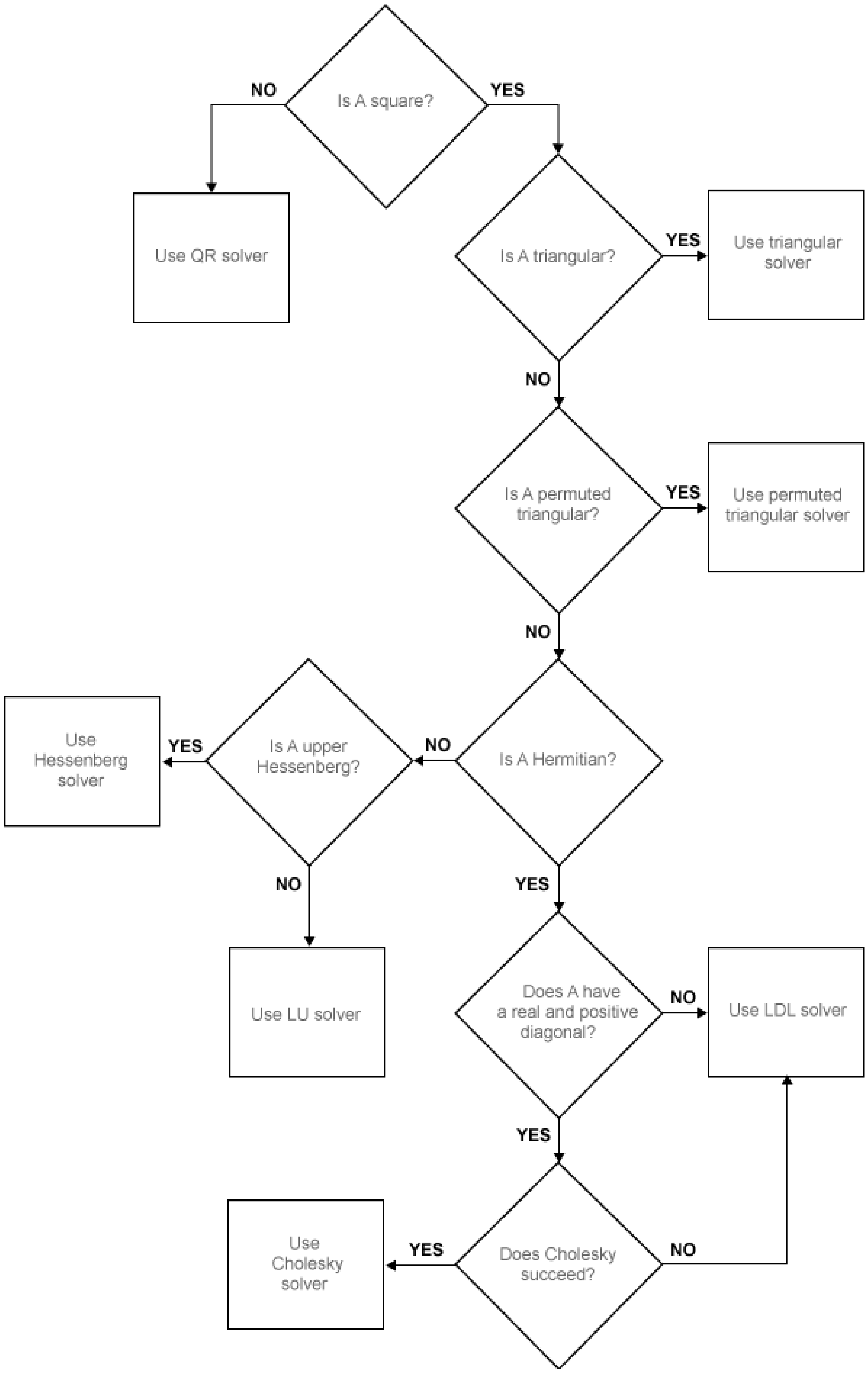

Figure 1.1: MLDIVIDE implementation in MATLAB

Work done in II-phase

2.1 A deeper understanding of problem

2.1.1 Finalising the Problem Statement

There are several iterative methods to solve Linear Solvers in PETSc. Our aim is now to benchmark the performance of running such a code on multi-core architecture and Graphic Processing Unit and comment on the Observations. If found suitable scope of improvement, codes can be modulated to leverage the compute power of both CPU and GPU.

2.2 Understanding PETSc

PETSc directives are relatively easy to relate if one has an in depth understanding of MPI directives. Just like MPI, PETSc has an initialisation collective, and finalising collective, a communicator. Many of these PETSc attributes/directives indirectly use/call MPI attributes/directives only.

2.2.1 Krylov Subspace Mthod

Our focus is to solve linear Equations of the form $Ax = b$ for which PETSc provides a sequential and parallel framework (KSP) to solve linear equations both directly and iteratively. We create an object of type *KSP which serves as KSP context. We deal with this context for everything ranging from selecting the type of solver, selecting direct or iterative method, calling the solve function, getting the iteration count etc.

2.3 Running PETSc on HPC cluster

Running Multi-node code on IIT Mandi cluster is not an easy task. Currently we can not leverage the multinode compute power of cluster (and hence MPI to its full extent) because of low bandwidth connectivity between nodes (due to absence of Infiniband connectivity). Thus *mpiexec* codes are run parallelly on multicore shared architecture. The same is the case for PETSc codes.

2.3.1 Containerized Environments

Containers provide a platform for virtualization. One can specify and install all the required dependencies, binaries, configuration files and set environment variables necessary for a simulation. A container does not contain an Operating System image, but instead it runs on the host Operating system by sharing its hardware.

One of the biggest benefits of a container is that it is very small in size as compared to a virtual machine. It does not need to lock the host device hardware resources as in the case of virtual machines and thus reduces the overhead. Container instances start almost instantly as compared to virtual machines which may take some time to boot.

There are multiple other benefits to containers and we will see the practical use of containers below.

2.3.2 Need for Containerization

Since each user requires a certain set of dependencies, certain packages, we cannot install everything into the cluster node as every user does not have root privileges. Also, doing this can potentially harm the environment variables and other installations and dependencies in the cluster.

Containers come to the rescue in such a scenario where each user uses their own version of container for simulating the required environment. Also, containers provide a guarantee that once an instance of that container runs on one system, then it should and will run on any system irrespective of the hardware differences.

2.3.3 Singularity and other alternatives

The most popular containerization software currently in the market is docker. However singularity is preferred over docker for HPC environment as Docker requires root privileges for some of its functionalities which can not be handed over to all users as discussed above.

IIT Mandi cluster also uses singularity for environment simulation. One has to build their own Singularity image with the extension .simg and use Singularity commands to run an instance of the image and do the required simulation.

2.3.4 Singularity Image for PETSc for CPU

Below is the script(petscbaseimage.singularity) used to build petscbaseimage.img for running and executing CPU based PETSc code.

```
        Bootstrap: docker
From: ubuntu:18.04
```

```
%post
    apt-get update
    apt-get -y install git wget python g++ gcc gfortran
    apt-get -y install mpich libblas-dev liblapack-dev
    apt-get -y install build-essential
    apt-get -y install python3-pip
    wget http://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.12.1.tar.gz
    tar xvzf petsc-3.12.1.tar.gz
    rm -r /petsc-3.12.1.tar.gz
    cd /petsc-3.12.1
    ./configure
    make all test
    cp -r ./lib/petsc /lib
    cd /
    pip3 install matplotlib
```

The command to build an image using this script as follows

```
singularity build petscbaseimage.simg petscbaseimage.singularity
```

The command to run an executable using mpiexec and this image is

```
singularity exec petscbaseimage.simg mpiexec -n 'num_cores' ./'executable'
```

We were successfully able to run PETSc simulations on IIT Mandi cluster using the above generated singularity image.

2.3.5 Singularity Image for PETSc for GPU

Below is the script(cudapetscbaseimage.singularity) used to build cudapetscbaseimage.img for running and executing CPU based PETSc code.

```
Bootstrap: docker
From: nvidia/cuda:9.0-devel

%files
    petsc-3.10.5.tar.gz

%post
    fileecho="## nvcc command
    export PATH=/usr/local/cuda-9.0/bin:$PATH
    export LD_LIBRARY_PATH=/usr/local/cuda-9.0/lib64:$LD_LIBRARY_PATH

    #Binding against the driver on the peregrine
    export LD_LIBRARY_PATH=/usr/lib64/nvidia:$LD_LIBRARY_PATH"

    mkdir -p /usr/lib64/nvidia
    echo $fileecho > /environment
    export PATH=/usr/local/cuda-9.0/bin:$PATH
    export LD_LIBRARY_PATH=/usr/local/cuda-9.0/lib64:$LD_LIBRARY_PATH
    export LD_LIBRARY_PATH=/usr/lib64/nvidia:$LD_LIBRARY_PATH
    nvcc --version

    sed -i 's|http://archive.ubuntu|http://jp.archive.ubuntu|g' /etc/apt/sources.list
    apt update -y
    apt -y install git wget python g++ gcc gfortran curl
    apt -y install mpich libblas-dev liblapack-dev
    apt -y install build-essential valgrind
    apt -y install python3

    curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py || echo "curl failed"
    python3 get-pip.py || echo "script failed"
    rm get-pip.py || echo "removal failed"
    python3 -m pip install --upgrade pip || echo "pip upgrade failed"
    python3 -m pip install matplotlib || echo "matplotlib failed"

    mkdir /petsc_dir/
    tar xvzf /petsc-3.10.5.tar.gz -C /petsc_dir/
    rm -r /petsc-3.10.5.tar.gz

    petsc_dir=/petsc_dir/petsc-3.10.5/

    cd $petsc_dir
    sed -i '1s/^/NVCCFLAGS += -Xcompiler -openmp\nNVCCFLAGS += -Xcompiler -fopenmp\n/' n
    ./configure --with-cuda=1 --with-precision=single --with-clanguage=c --download-cusp
    make all test
    cp -r ./lib/petsc /lib
```

```
cd /
```

The command to build an image using this script is same as that for petscbaseimage.simg.

The command to run an executable using mpiexec is same as above and for running the code on GPU is as follows

```
singularity exec --nv cudapetscbaseimage.simg ./executable -vec_type cuda -mat_t
```

2.4 Benchmarking

We ran multiple simulations to solve linear equation $Ax = b$ where the matrix A is a tridiagonal symmetric matrix which is solved using Jacobi preconditioner and GMRES KSP solver.

2.4.1 Multi core CPU simulation

The command for running the executable using mpiexec using numCore as number of cores and numDim as dimension of square matrix A is as follows

```
singularity exec petscbaseimage.simg mpiexec -n numCore ./ex -n numDim
```

We ran the simulation using numCore $\in \{1, 2, 4, 6, 8, 12\}$ and with the dimensions in the range of $100 * 2^{0-14}$.

2.4.2 GPU simulation

The command for running the executable using cuda and numDim as dimension of square matrix A as as follows

```
singularity exec --nv cudapetscbaseimage.simg ./ex -vec_type cuda -mat_type aijc
```

We ran the simulation with the dimensions in the range of $100 * 2^{0-14}$.

Unfortunately, the above commands were not running on the cluster GPU, thus we had to resort to using GPU on a personal laptop but using the same Singularity image.

2.4.3 Results and Observations

The following results (Fig 2.2) were obtained which are discussed in depth below.

This graph provides an overview which aligns with the expected results that more the number of cores, the lesser would be the time for computation. The results seem to be more pronounced for higher dimensions of vector/matrix.

As expected, computation on GPU easily wins over computation on 12 hyper-threaded cores of CPU even when the GPU is a modest NVIDIA-GTX 1050 running on a personal laptop.

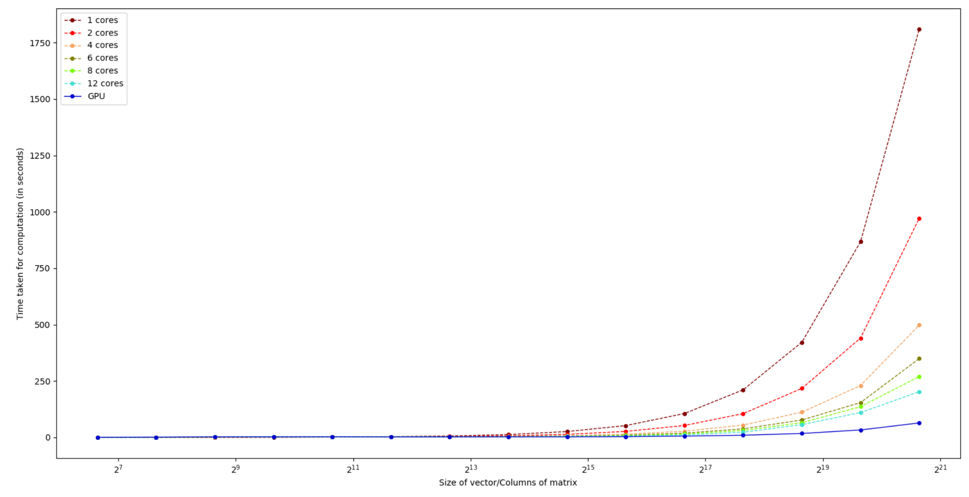

Figure 2.2: Benchmarking KSP solver for symmetric tridiagonal matrix using multiple cores and GPU

However judging the whole scenario only with this graph would be unfair as we are clearly missing out on the observations for lower dimensionality of vector/matrix. Therefore, for greater insight we would be zooming into the same graph as above upto dimensionality of 16,000 (Fig 2.3).

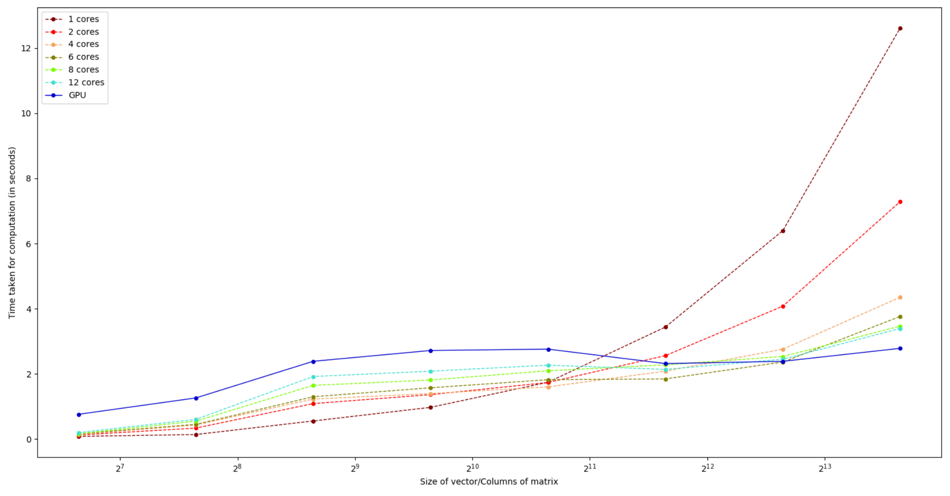

Figure 2.3: Zoomed in graph of Benchmarking KSP solver for symmetric tridiagonal matrix using multiple cores and GPU

For lower dimensions upto approximately 2^{12}, we can see that GPU takes the longest time to compute as there is data transfer overhead between the CPU memory and GPU memory caused by cudaMemCopy() function.

Also, for the same range of lower dimensionality, higher number of cores result in more time taken due to race conditions and message passing overhead(even though it is multicore architecture possibly cc-NUMA architecture).

To further justify our observation that we will get poorer performance on lower dimensionality using high number of cores, we refer to Fig 2.6 and Fig 2.9.

2.4.4 Conclusion

From the results of benchmarking, it is invariant that performance on GPU is many fold better than that on multicore CPU for matrices of larger which infcat are used in research and academic fields. Thus the tasks of implementing a hybrid framework of the two and getting significant performance improvement seem to be more difficult than estimated. This opens up more research aspects into the problem statement.

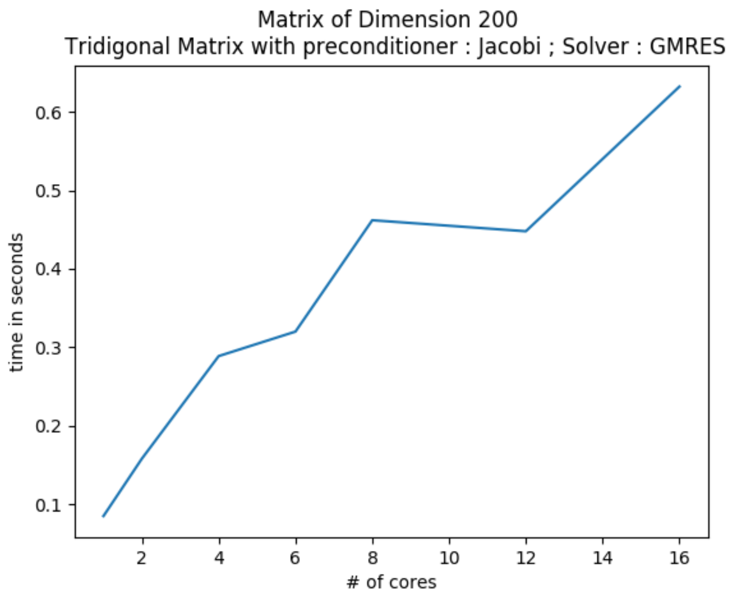

Figure 2.4: Matrix size as 200

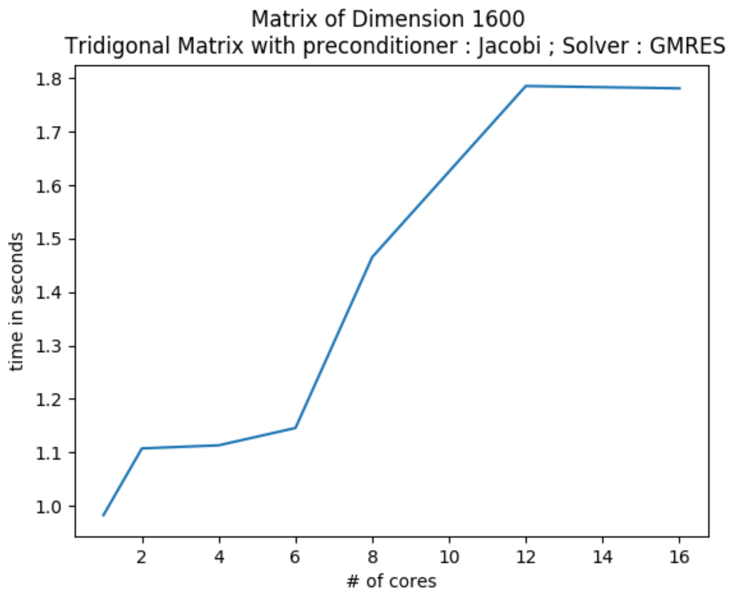

Figure 2.5: Matrix size as 1600

Figure 2.6: Time taken increases as number of cores increase for lower dimensionality (200,1600/0 and decreases for higher dimensionality(102400).

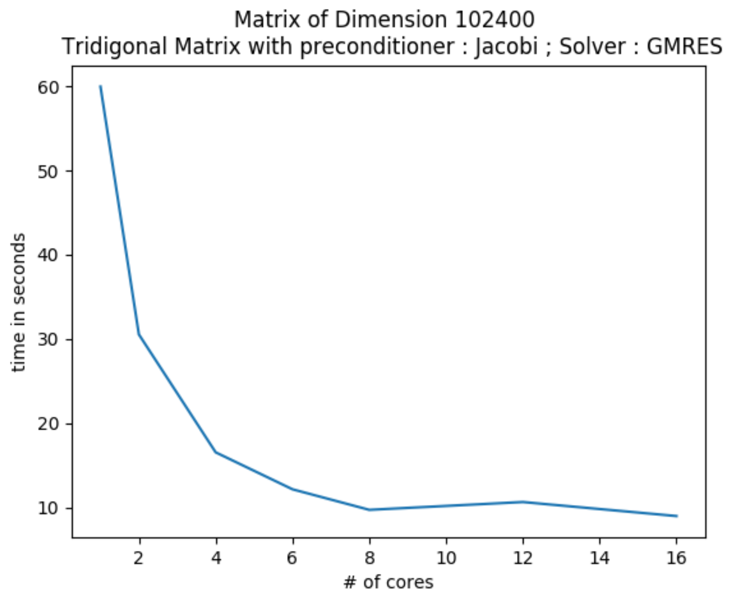

Figure 2.7: Matrix size as 102400

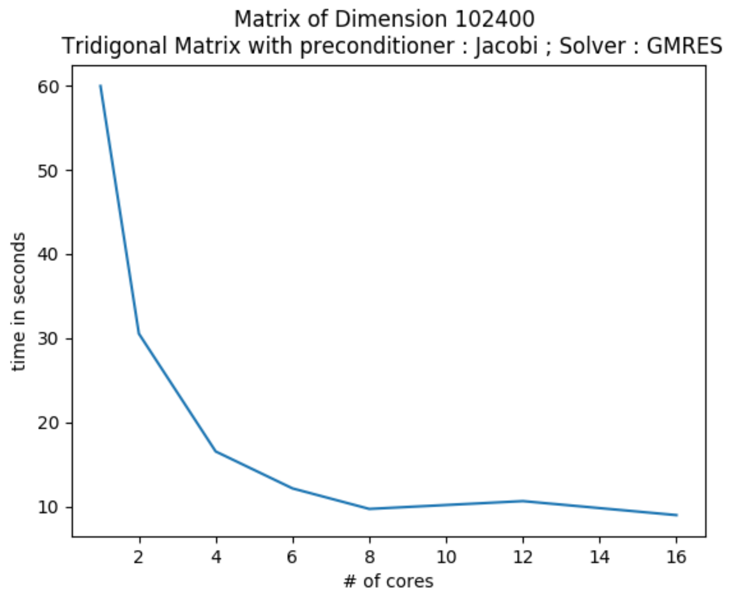

Figure 2.8: Matrix size as 1638400

Figure 2.9: Time taken decreases as number of cores increase for higher dimensionality(102400,1638400).

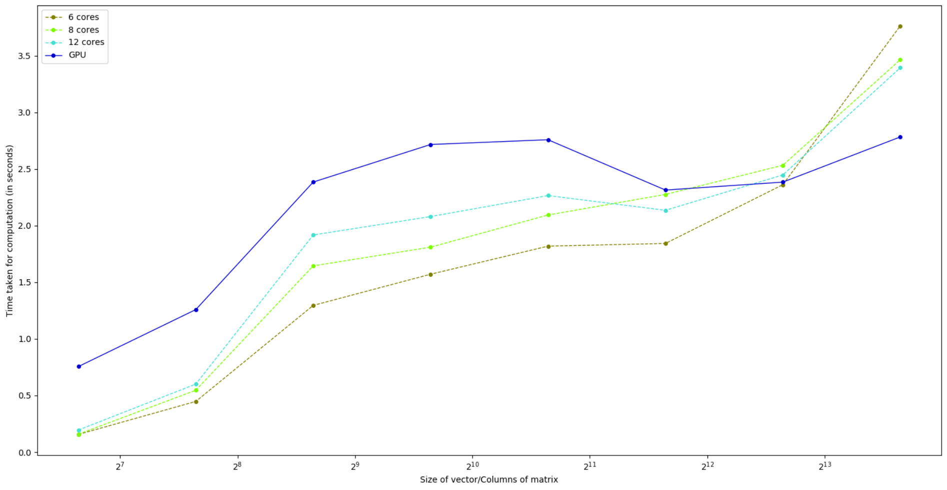

Figure 2.10: Zoomed in graph of Benchmarking KSP solver for symmetric tridiagonal matrix using multiple cores and GPU

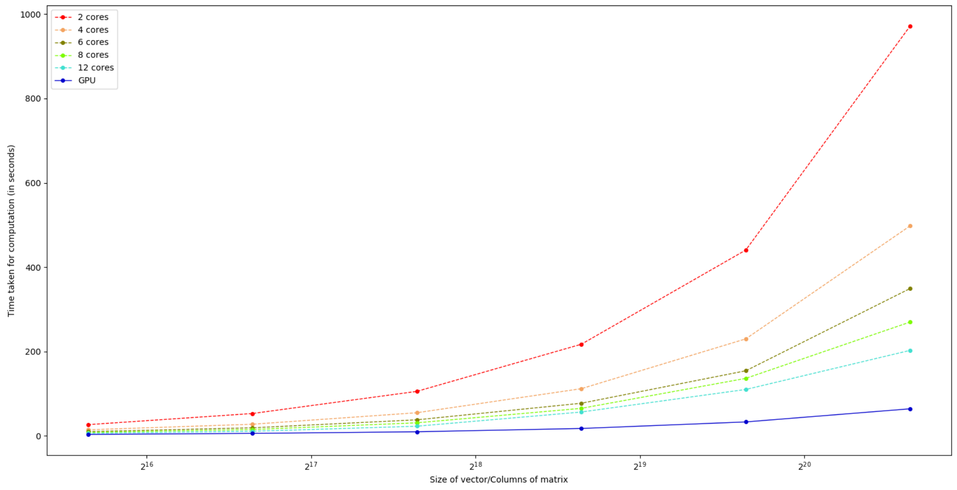

Figure 2.11: Zoomed in graph of Benchmarking KSP solver for symmetric tridiagonal matrix using multiple cores and GPU

Bibliography

[1] Jacobsen, Dana, Thibault, Julien, Senocak, Inanc. (2010). An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters. Inanc Senocak.

[2] S. Cuomo, A. Galletti, G. Giunta, L. Marcellino. Toward a Multi-level Parallel Framework on GPU Cluster with PetSC-CUDA for PDE-based Optical Flow Computation, Procedia Computer Science, Volume 51, 2015.

[3] Portable, Extensible Toolkit for Scientific Computation, PETSc/Tao, www.mcs.anl.gov/petsc/

[4] Computational fluid dynamics (CFD), whatis.techtarget.com/definition/computational-fluid-dynamics-CFD

[5] Computational Science and Engineering, Gilbert Strang.

[6] PETSc Users Manual, Revision 3.12, Mathematics and Computer Science Division, Argonne National Laboratory, https://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf

[7] PETSc Tutorial - Profiling, Nonlinear Solvers, Unstructured Grids, Threads and GPUs, https://www.mcs.anl.gov/petsc/meetings/2016/slides/tutorial2.pdf

[8] What are containers and why do you need them?, https://www.cio.com/article/2924995/what-are-containers-and-why-do-you-need-them.html

[9] The Portable Extensible Toolkit for Scientific Computing, https://www.mcs.anl.gov/petsc/documentation/tutorials/ECP19/ECP19-Intro-Solvers.pdf