Hybrid Parallelised Framework for the solution of PDEs on HPC Clusters

By:
NIKHIL GUPTA (B16023)
RITWIK SAHA (B16110)

Mentored By:
Dr. GAURAV BHUTANI

SCEE
IIT MANDI

AIM :

To develop a hybrid framework to utilize HPC resources to solve LINEAR EQUATIONS (and hence Partial Differential Equations) using background parallelisation tools (PETSc).

Solving LINEAR EQUATIONS

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

ITERATIVE METHODS

- Jacobi Method
- Gauss-Seidel Method
- Steepest Descent Method
- Conjugate Gradient Method
- Many more….

PROGRESS after MID TERM EVALUATION

- Finalising the problem statement.
- Running PETSC codes on cluster.
- Benchmarking GMRES method.
- Analysing Hybrid approach for Conjugate gradient method

CONTAINERIZATION

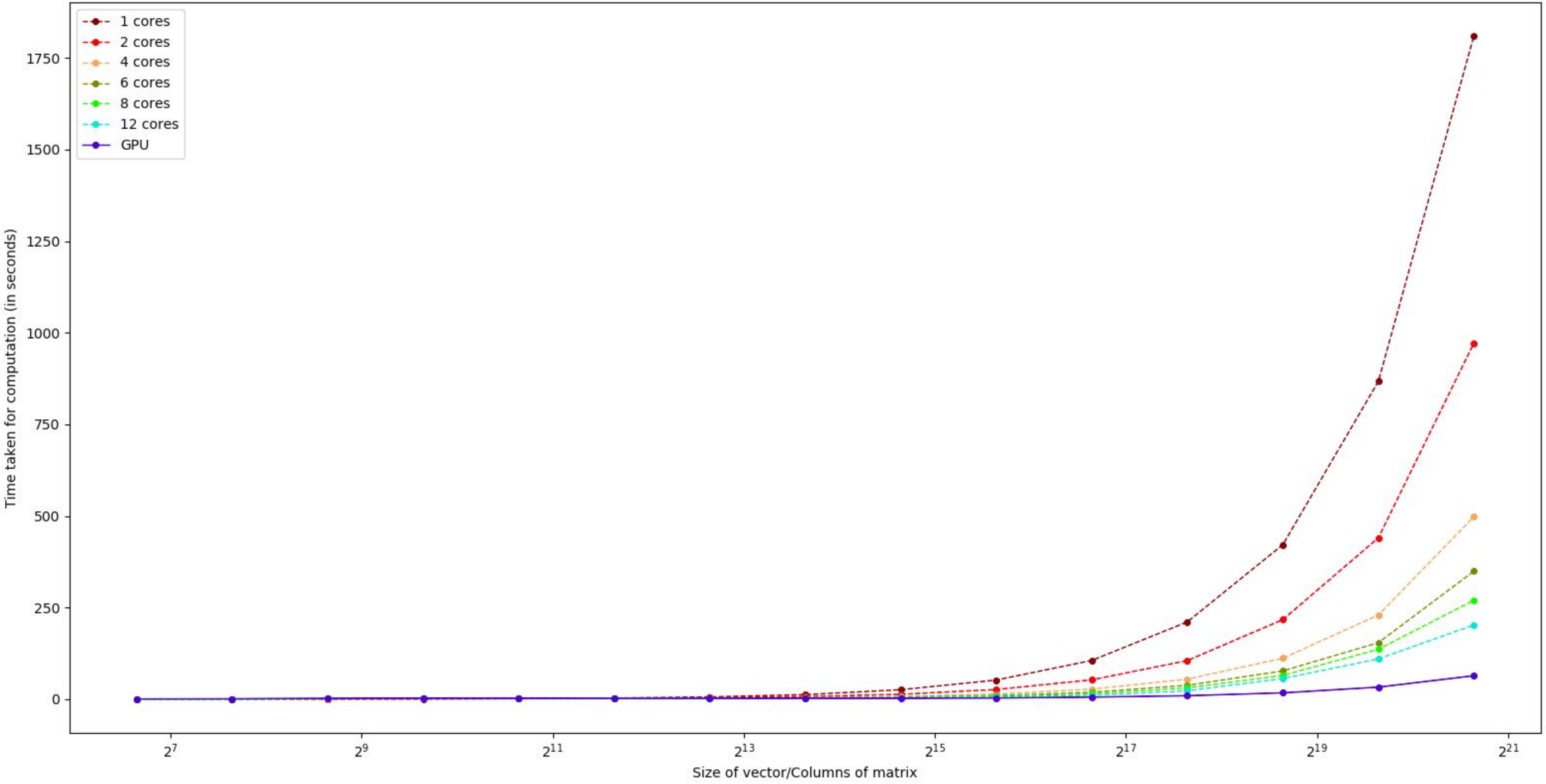
- SINGULARITY
  - To create custom runtime environment to run simulations on IIT Mandi HPC Cluster.
  - Created Singularity images for both CPU and GPU implementations.
  - Singularity also exposes the host’s working directory inside the container unlike docker.

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

%files
    petsc-3.10.5.tar.gz

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

## Binding against the driver on the peregrine
# -   LD_LIBRARY_PATH includes /usr/lib64/nvidia
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"

    # wget http://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.12.1.tar.gz
    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/'
makefile
    ./configure --with-cuda=1 --with-precision=single --with-clanguage=c --download-cusp
    make all test
    cp -r ./lib/petsc /lib
    cd /
```

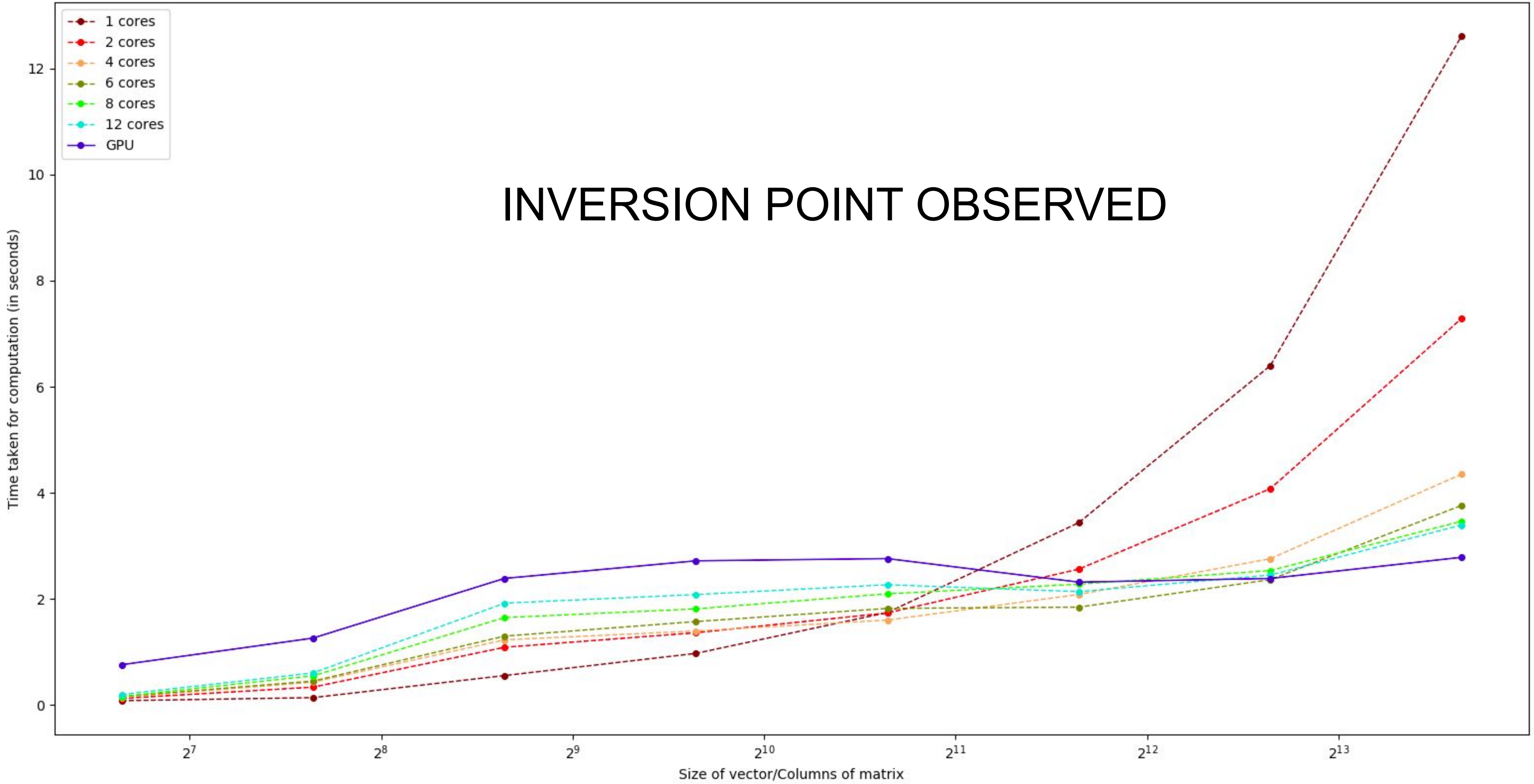
BENCHMARKING - I

- PETSC code
  - Solving Linear Equation for Tridiagonal Matrix
  - Using GMRES method and JACOBI preconditioner.
  - Multicore Simulation (#cores from 1 to 12)
  - GPU Simulation

OBSERVATIONS



OBSERVATIONS



CONJUGATE GRADIENT METHOD

- Our own implementation
  - Using PETSC data structures
  - Easy to choose between CPU or GPU implementation for same code

- Ran the code with **1 core CPU** implementation and **GPU**

- Implemented a **Hybrid version** of the same

CONJUGATE GRADIENT METHOD

```
VecSetRandom(x, NULL);
MatMult(A,x,r);
VecAXPY(r,minusone,b);


//Initialize p=-r
VecSet(p, zero);
VecAXPY(p,minusone,r);

for (i=0;i<iters;++i){
    VecNorm( r , NORM_2 , &rdot);
    rdot = rdot*rdot;
    MatMult(A,p,Ap);
    VecDot(p, Ap, &pAp);

    alpha = rdot/pAp;

    VecAXPY(x, alpha, p);

    VecAXPY(r,alpha,Ap);

    VecNorm(r,NORM_2,&val);
    val = val*val;
    beta = val/rdot;

    VecAXPBY(p,minusone, beta, r);
}
```

Compute $r_0 = Ax_0 - b, p_0 = -r_0$

For $k = 0, 1, 2, ...$ **until convergance**

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$$

$$x_{k+1} = x_k + \alpha_k p_k$$

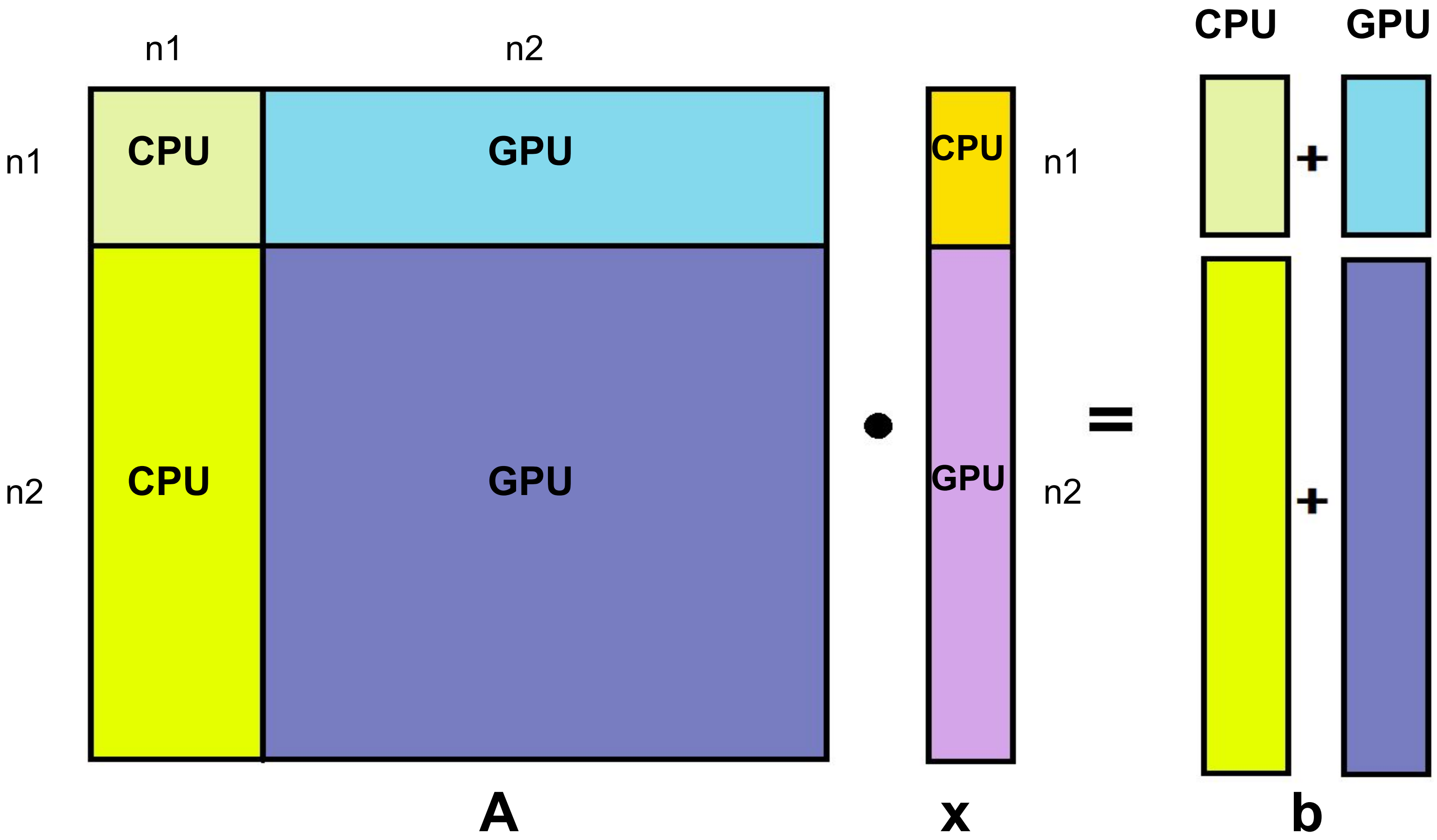
$$r_{k+1} = r_k + \alpha_k A p_k$$

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

$$p_{k+1} = -r_{k+1} + \beta_k p_k$$

End

HYBRID IMPLEMENTATION



HYBRID IMPLEMENTATION

- **CPU** routines for **VEC1 (n1)**
- **GPU** routines for **VEC2 (n2)**

- Taking maximum time of two

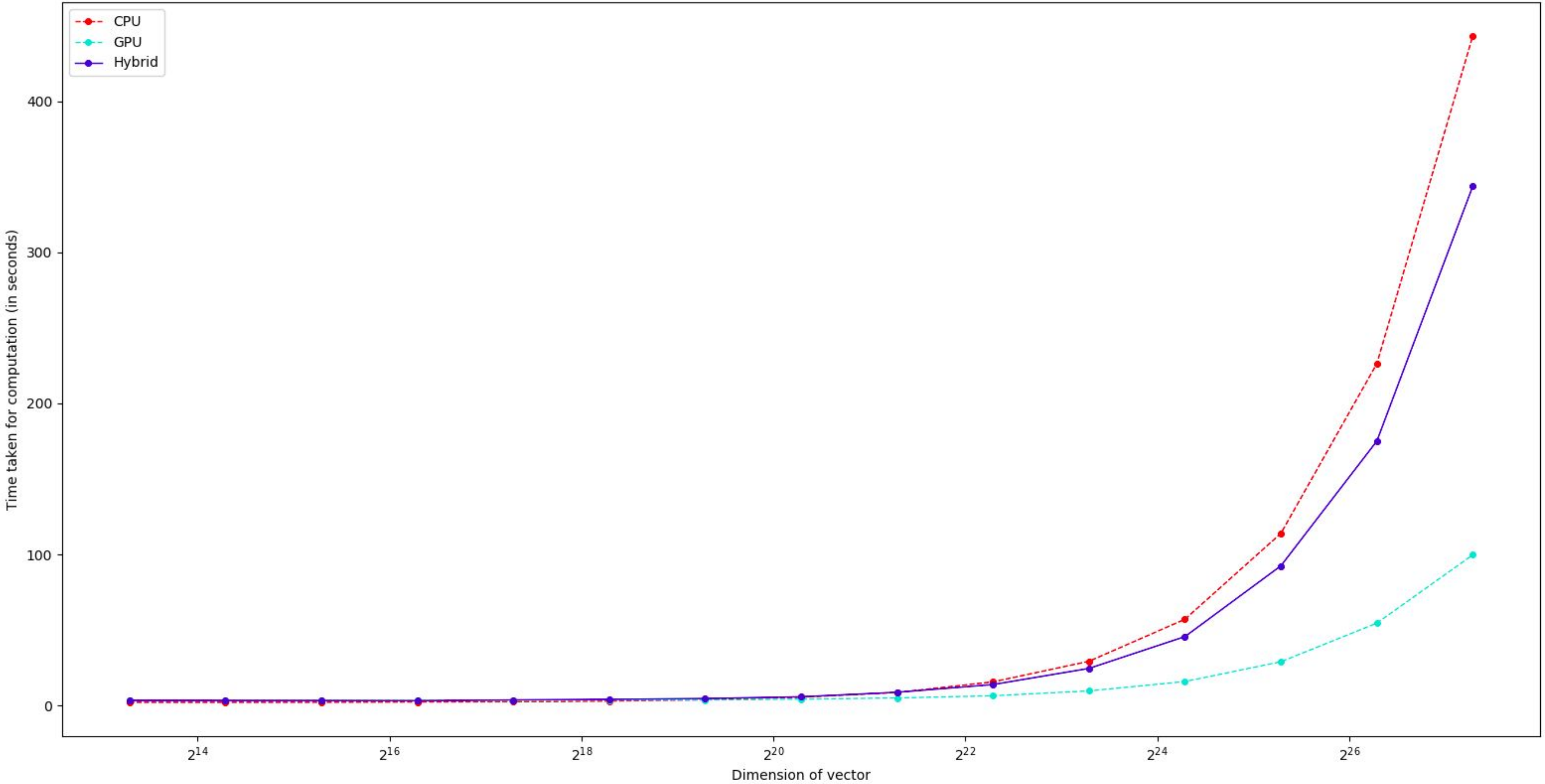
```
clock_t VecAXPBY_split(Vec y1, Vec y2, PetscReal alpha, PetscReal beta, Vec x1, Vec x2){

            clock_t begin, end, a,b,r;

            begin = clock();
    VecAXPBY(y1, alpha, beta, x1);
            end = clock();
            a = (end - begin);

            begin = clock();
    VecAXPBY(y2, alpha, beta, x2);
            b = (end - begin);

            if( a > b ) r = a ; else r = b;

    return r;
}
```

HYBRID IMPLEMENTATION (Multi -Threaded)

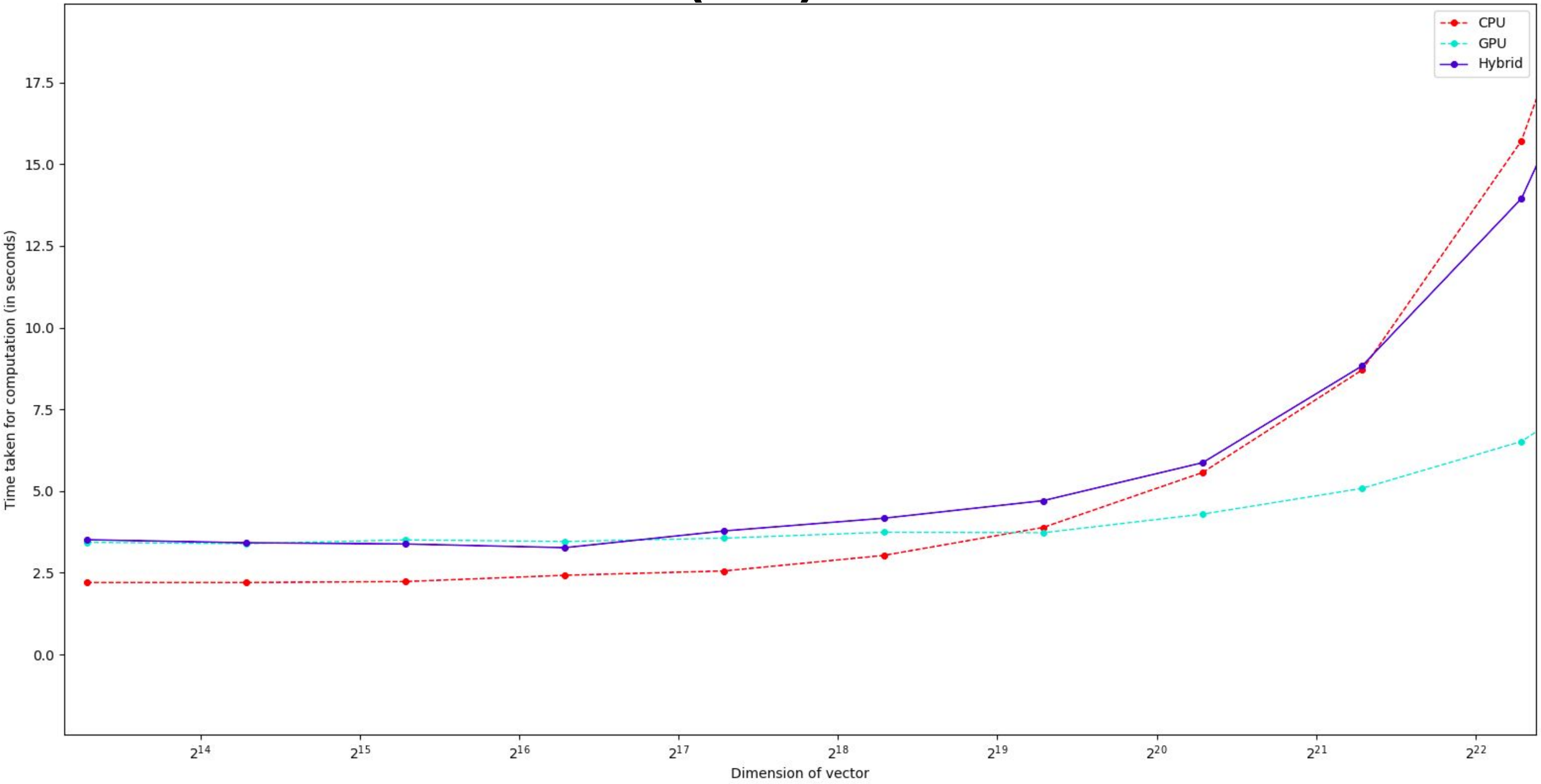
- Run **CPU** and **GPU** parts on different threads, thus almost simultaneously

```
typedef struct{
    Vec x;
    Vec y;
    PetscScalar *val;
}VecDot_struct;

void* VecDot_thread(void* ptr){
    VecDot_struct *sptr = (VecDot_struct*)ptr;
    VecDot(sptr->x, sptr->y, sptr->val);
    return NULL;
}

void VecDot_split(Vec x1, Vec x2, Vec y1, Vec y2, PetscReal *value){
    PetscReal a, b;
    pthread_t t1, t2;
    VecDot_struct s1 = {x1, y1, &a}, s2 = {x2, y2, &b};
    pthread_create(&t1, NULL, VecDot_thread, (void*)(&s1));
    pthread_create(&t2, NULL, VecDot_thread, (void*)(&s2));
    pthread_join(t1, NULL); pthread_join(t2, NULL);
    *value = (a+b);
    return;
}
```

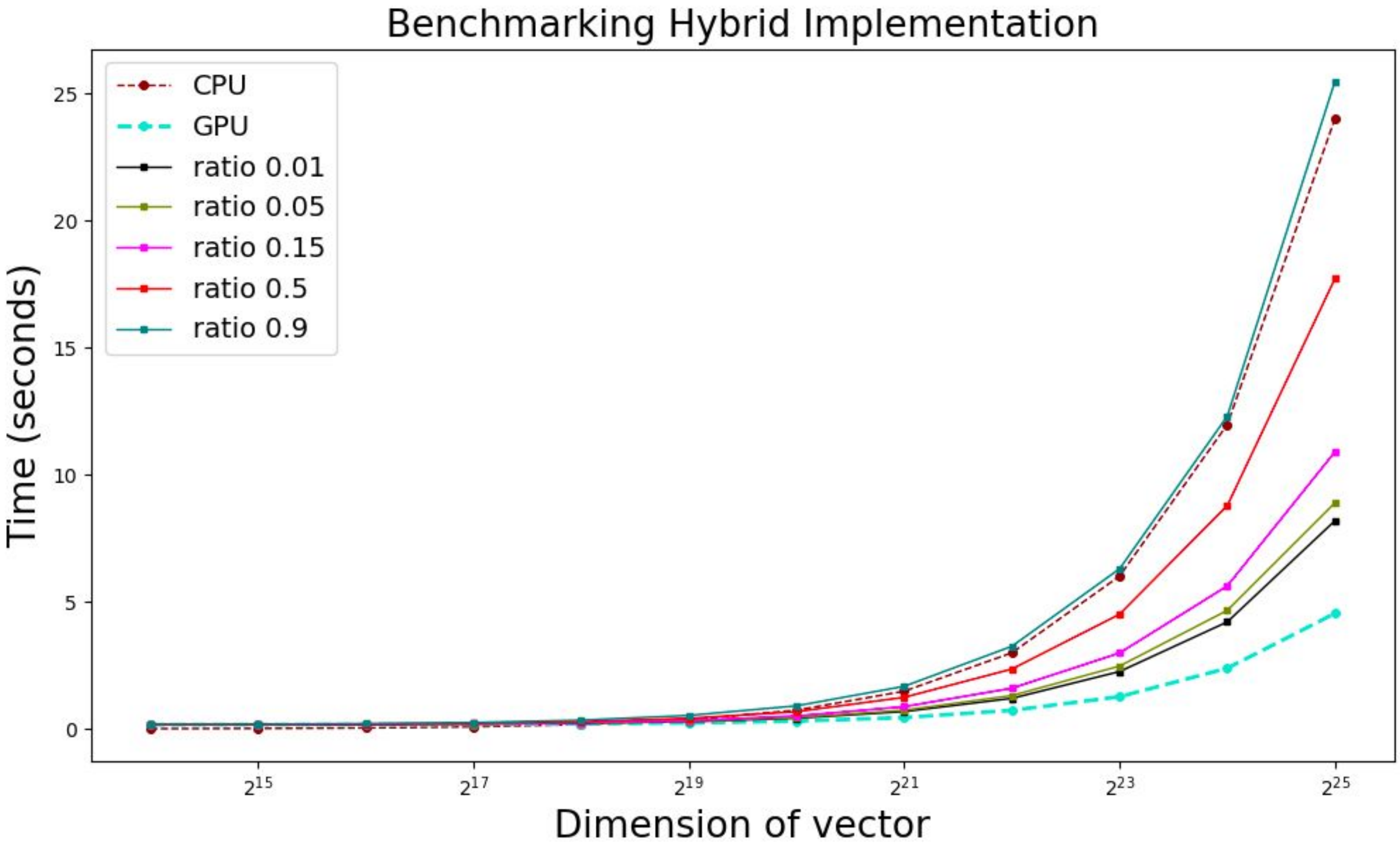
BENCHMARKING I - (n/2)



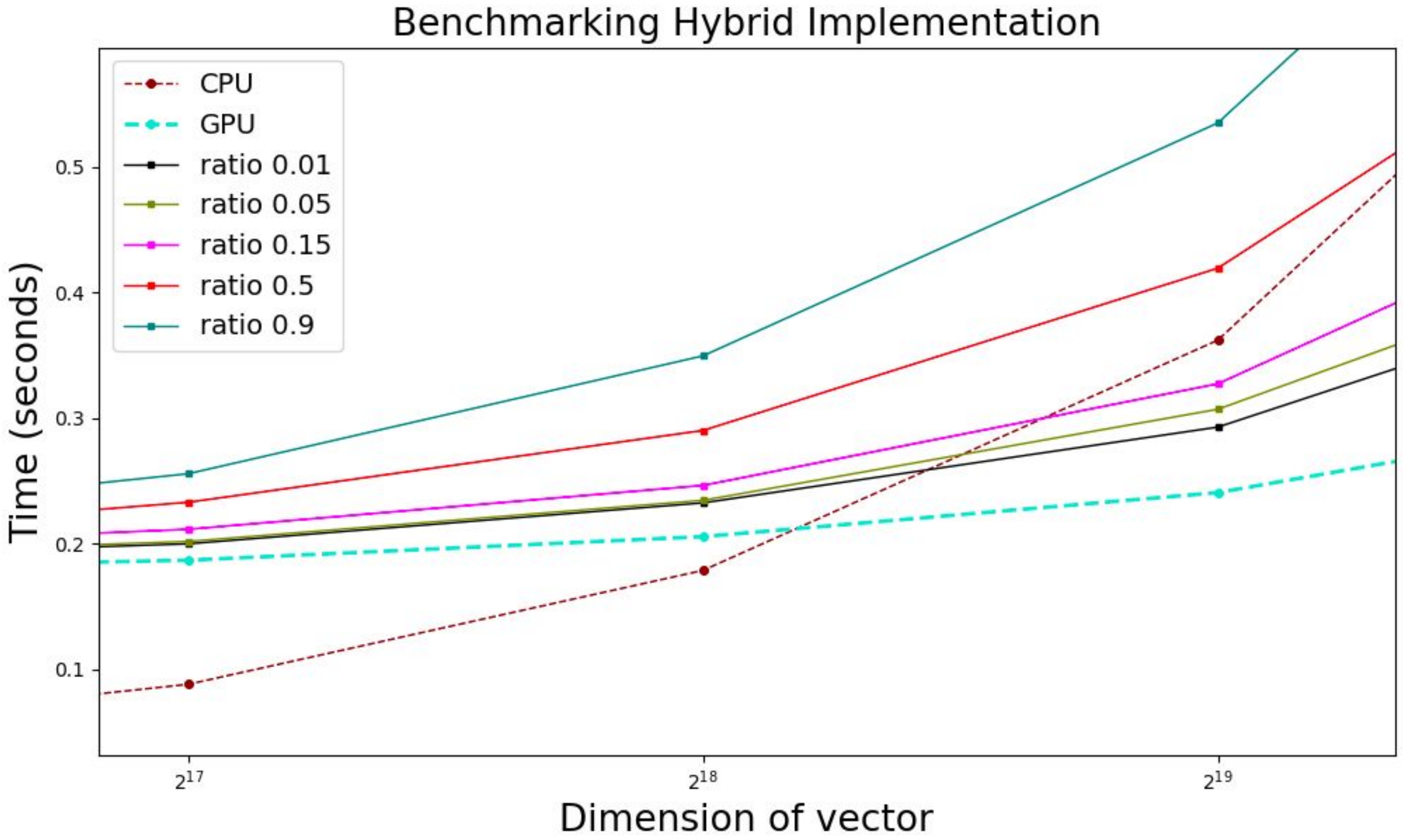
BENCHMARKING I - (n/2)



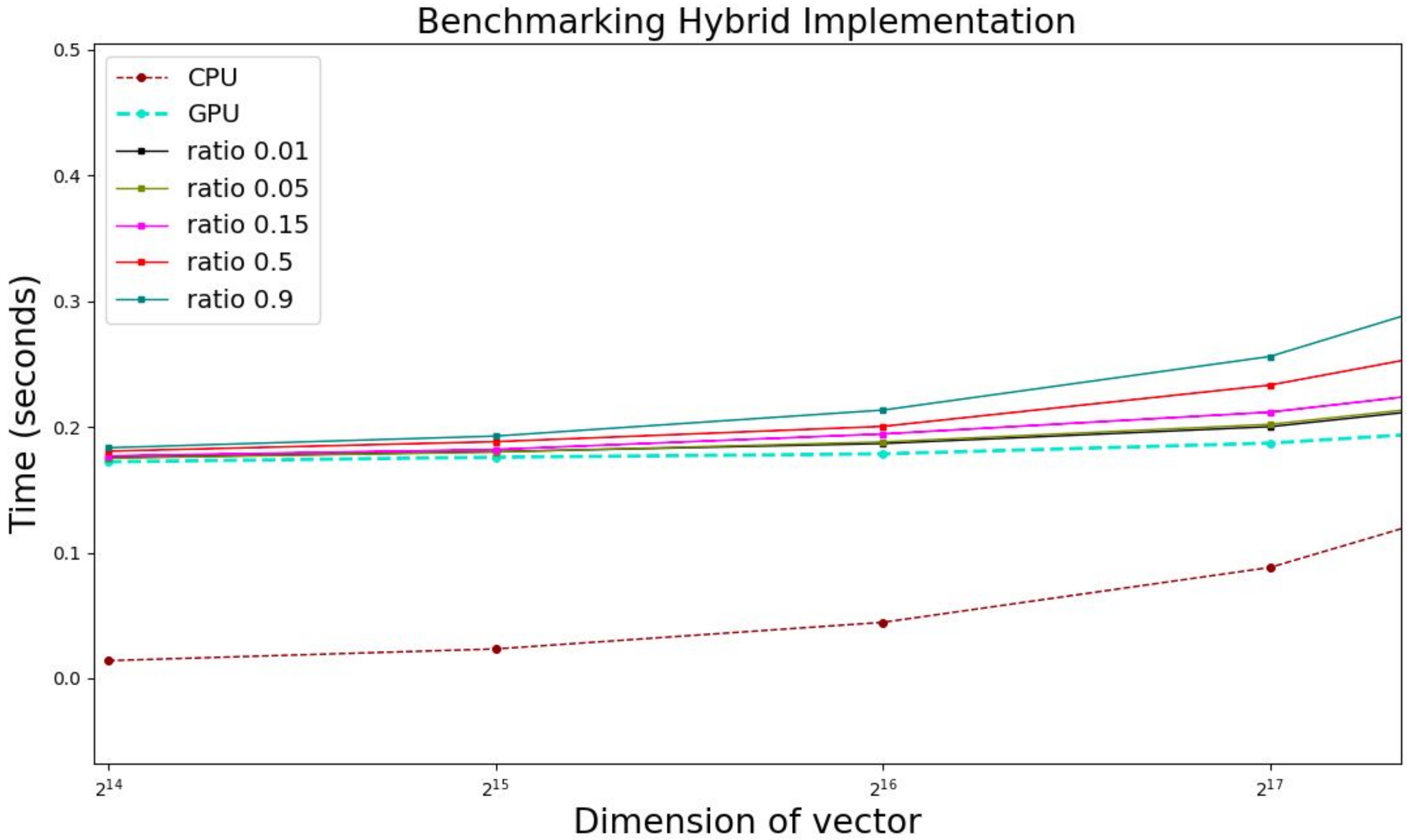
BENCHMARKING II - (n1, n2)



BENCHMARKING II - (n1, n2)



BENCHMARKING II - (n1, n2)



CONCLUSIONS

- PETSC is difficult to hybridise (lack of documentation)
- Multicore + GPU implementation leads to deadlock
  - Need to optimise
- Our Implementation
  - **Time(GPU) < Time (Hybrid code) < Time (CPU Code)**
- Our Aim is to achieve
  - **Time(Hybrid code) < Time(GPU)**
- IIT Mandi HPC Cluster incompatible with “Multinode + GPU” PETSC implementation.
  - Absence of InfiniBand connectivity

FUTURE SCOPE

- Improvise PETSC GPU implementation
  - Understanding PETSC structure in depth
  - New VECTOR and MATRIX type for hybridisation

- New Implementation of Linear Solvers
  - Hybrid implementation
  - CUDA / OpenMP / BLAS-LAPACK

VecType

String with the name of a PETSc vector

Synopsis

```
typedef const char* VecType;
#define VECSEQ         "seq"
#define VECMPI         "mpi"
#define VECSTANDARD    "standard"   /* seq on one process and mp
#define VECSHARED      "shared"
#define VECSEQVIENNACL "seqviennacl"
#define VECMPIVIENNACL "mpiviennacl"
#define VECVIENNACL    "viennacl"   /* seqviennacl on one proces
#define VECSEQCUDA     "seqcuda"
#define VECMPICUDA     "mpicuda"
#define VECCUDA        "cuda"       /* seqcuda on one process ar
#define VECNEST        "nest"
#define VECNODE        "node"       /* use on-node shared memory
```

https://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/Vec/VecType.html

THANK YOU