Name: Aniket R Vishwakarma.
HPC-1 hw-3.

High Performance Computing 1: Homework #3

Due Sunday, Nov. 27, 2022 at 11:59 pm

Submission Instructions

Code may be written in `C`, `C++`, or `Fortran`. Clarity of code and plots will be a component of the credit you receive. Please `tar` or `zip` the components listed below into a SINGLE bundled file and submit it electronically through UBLearns. In the bundled submission please include:
(a) A SINGLE PDF file containing your results description, graphs, and any tables of the program output.
(b) Any source code, text files generated, and shell script.
(c) `README` file the states the contents of the `tar` or `zip` file and the steps to compile and run different experiments.

Problem 1

The Gram-Schmidt process, as discussed in class, is an algorithm for orthogonalizing a set of vectors. There are excellent resources online, such as the Wikipedia entry and this presentation by Per-Olof Persson. We can express it using the notation from linear algebra. In Algorithm 1, we present both the Classical variant (CGS), and the Modified variant (MGS), which differ only by the line ending with the respective parentheticals below. For this problem, we will implement the Gram-Schmidt process in PETSc using

input : A set of n vectors $\{a_i\}$
output: A set of orthonormal vectors $\{q_i\}$ with the same span

for j = *1 to* n **do**
  $v_j = a_j$;
  for i = *1 to j-1* **do**
    $r_{ij} = q_i^\dagger a_j$ (CGS);
    $r_{ij} = q_i^\dagger v_j$ (MGS);
    $v_j = v_j - r_{ij} q_i$;
  end
  $r_{jj} = \|v_j\|_2$;
  $q_j = v_j / r_{jj}$;
end

Algorithm 1: Gram-Schmidt Orthogonalization

the basic operations in the Vec class. Make the type signature of your method, meaning the list of inputs, the following:

```
PetscErrorCode GramSchmidt(PetscInt n, Vec a[], Vec **q)
```

1. Write a serial PETSc program to orthogonalize a set of vectors using the Gram-Schmidt process using the Classical variant (CGS). Use VecSetRandom() to set the vector values. Check that the output vectors are orthogonal in the code.
2. In words, describe is needed to make this code run in parallel?

Problem 2

For this problem, we will investigate the scaling of the code from Problem 1 with vector size. You can use PetscLogEventRegister() to create a logging event, PetscLogEventBegin()/PetscLogEventEnd() to time the code, and `-log_view` to output the timing.

1. Run the code in serial for a set of $k = 20$ vectors. Make a table of the time taken for different vector sizes, $N = 1000$, 5000, 10000, 50000, and 100000.
2. Plot the time taken for each run.
3. Does the time T of your runs above have a polynomial dependence on the vector size N, $T = CN^{\alpha}$, where C is some constant? If so, what is α approximately?

Problem 3

For this problem, we will investigate the scaling of the code from Problem 1 with the number of vectors we orthogonalize. Timing can be performed as in Problem 2.

1. For vectors of size $N = 50000$, run the code in serial for different set sizes, $k = 2, 4, 8, 16, 32, 64$. Make a table of the time taken.
2. Plot the time taken for each run.
3. Does the time T have a polynomial dependence on the set size k, $T = Dk^{\beta}$, where D is a constant? If so, what is β approximately?

Problem 1 : ① Petsc program to orthoganalize vectors using Gram-Schmidt algorithm could be found in "Q1-GramSchmidtAlgo" directory

② As Petsc is built upon MPI, hence this written code could also be ran using MPI.

for example : To run this code on 8 processors use following commands :

① make gsa

② mpiexec -n 8 ./gsa -log_view

or

② mpirun -np 8 ./gsa -log_view

⇒ Following Screenshot, I solved for $n = k = 4$ for the orthogonality check. As we can see, we obtain Identity Matrix, which followed rule of $\begin{cases} \delta_{ij} = 1 & (\text{if } i=j) \\ 0 & (\text{if } i \neq j) \end{cases}$

⇒ Thus we can verify our code is right.

⇒ Also you can use '-help' flag on terminal to receive information provided by author followed by default help dialogue written by Petsc developers.

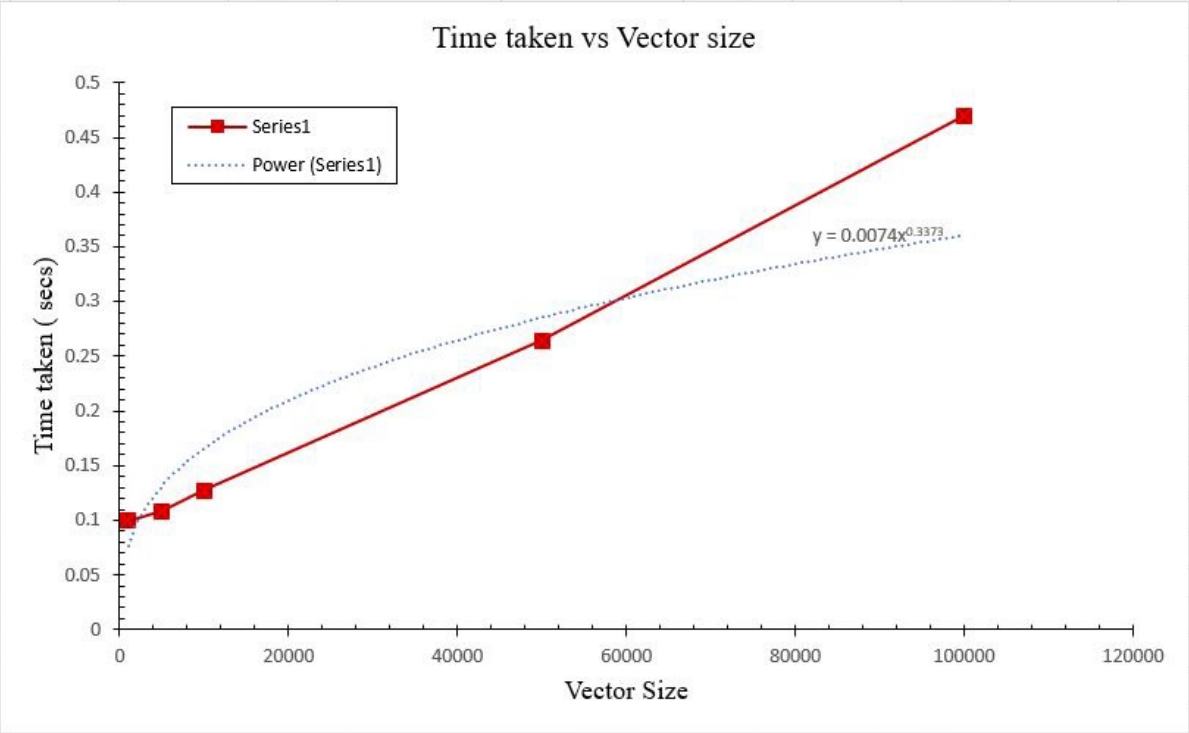
```
raziel66612@Raziel_Uchiha: /mnt/c/Users/anike/VSC_Cpp_code/petsc/hpc_hw_3_GS_algorithm/trial_3_final
raziel66612@Raziel_Uchiha:/mnt/c/Users/anike/VSC_Cpp_code/petsc/hpc_hw_3_GS_algorithm/trial_3_final$ ls
gsa.c  makefile
raziel66612@Raziel_Uchiha:/mnt/c/Users/anike/VSC_Cpp_code/petsc/hpc_hw_3_GS_algorithm/trial_3_final$ make gsa
/petsc/arch-linux-cxx-debug/bin/mpicxx -o gsa.o -c -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -Wno-lto-type-
mismatch -fstack-protector -fvisibility=hidden -g -O0  -std=gnu++17  -I/petsc/include -I/petsc/arch-linux-cxx-debug/include    `p
wd`/gsa.c
/petsc/arch-linux-cxx-debug/bin/mpicxx -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -Wno-lto-type-mismatch -fs
tack-protector -fvisibility=hidden -g -O0  -o gsa gsa.o -Wl,-rpath,/petsc/arch-linux-cxx-debug/lib -L/petsc/arch-linux-cxx-debug/
lib -Wl,-rpath,/petsc/arch-linux-cxx-debug/lib -L/petsc/arch-linux-cxx-debug/lib -Wl,-rpath,/usr/lib/gcc/x86_64-linux-gnu/9 -L/us
r/lib/gcc/x86_64-linux-gnu/9 -lpetsc -lflapack -lfblas -lpthread -lm -lstdc++ -ldl -lmpifort -lmpi -lgfortran -lm -lgfortran -lm
-lgcc_s -lquadmath -lstdc++ -ldl
raziel66612@Raziel_Uchiha:/mnt/c/Users/anike/VSC_Cpp_code/petsc/hpc_hw_3_GS_algorithm/trial_3_final$ ./gsa -k 4 -n 4
1  0  0  0
0  1  0  0
0  0  1  0
0  0  0  1
raziel66612@Raziel_Uchiha:/mnt/c/Users/anike/VSC_Cpp_code/petsc/hpc_hw_3_GS_algorithm/trial_3_final$ ./gsa -help


 This code take k vectors of size N and check their orthogonality by gram schmidt algorithm

----------------------------------------
Petsc Release Version 3.18.1, unknown
       The PETSc Team
    petsc-maint@mcs.anl.gov
 https://petsc.org/
See docs/changes/index.html for recent updates.
See docs/faq.html for problems.
See docs/manualpages/index.html for help.
Libraries linked from /petsc/arch-linux-cxx-debug/lib
----------------------------------------
Options for all PETSc programs:
 -version: prints PETSc version
 -help intro: prints example description and PETSc version, and exits
 -help: prints example description, PETSc version, and available options for used routines
 -on_error_abort: cause an abort when an error is detected. Useful
       only when run in the debugger
 -on_error_attach_debugger [gdb,dbx,xxgdb,ups,noxterm]
      start the debugger in new xterm
      unless noxterm is given
 -start_in_debugger [gdb,dbx,xxgdb,ups,noxterm]
      start all processes in the debugger
 -on_error_emacs <machinename>
    emacs jumps to error file
 -debugger_ranks [n1,n2,..] Ranks to start in debugger
 -debugger_pause [m] : delay (in seconds) to attach debugger
 -stop_for_debugger : prints message on how to attach debugger manually
                      waits the delay for you to attach
 -display display: Location where X window graphics and debuggers are displayed
 -no_signal_handler: do not trap error signals
```

Problem 2 : Go to directory "Q2" to find Code & results in .txt file's
Also you can open the excel Spreadsheet to view the Summarized result

PROBLEM 2:	
k= 20 vectors	
N = vector size	Time taken
1000	0.099809
5000	0.10823
10000	0.12712
50000	0.26417
100000	0.46984



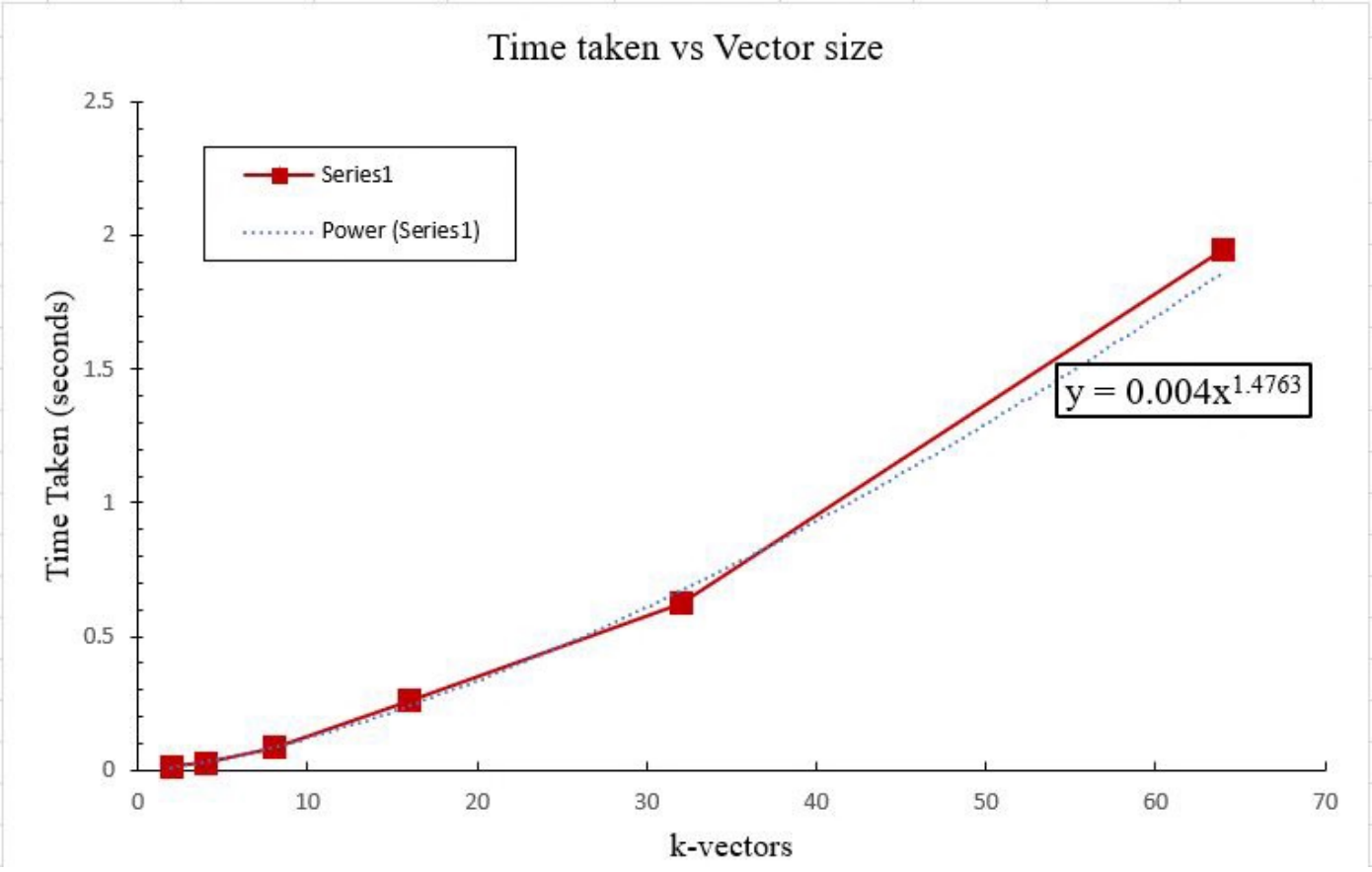
→ As observed above, on regressing the result using Power law ($T = CN^{\alpha}$) we can see that Time does have a polynomial dependency on N.

$T = CN^{\alpha} = 0.0074 \; x^{0.3373}$

Hence $\alpha = 0.3373$

Problem 3 : Go to directory "Q3" to find Code & results in .txt file's
Also you can open the excel Spreadsheet to view the Summarized result

PROBLEM 3:	
N=50000 vect size	
k	Time taken
2	0.012206
4	0.02786
8	0.084397
16	0.25969
32	0.62495
64	1.9465



As observed, on regarding the results by using Power law ($T = D K^{\beta}$) we can see that Time does have a polynomial dependency on K.

$T = D K^{\beta} = 0.004 \times^{1.4763}$

Hence $\beta = 1.4763$