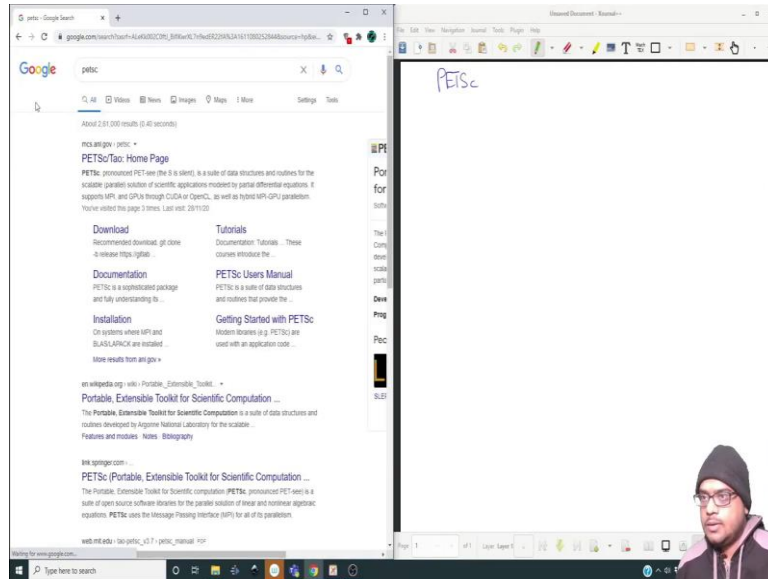


**Tools in Scientific Computing**  
**Prof. Aditya Bandopadhyay**  
**Department of Mechanical Engineering**  
**Indian Institute of Technology, Kharagpur**

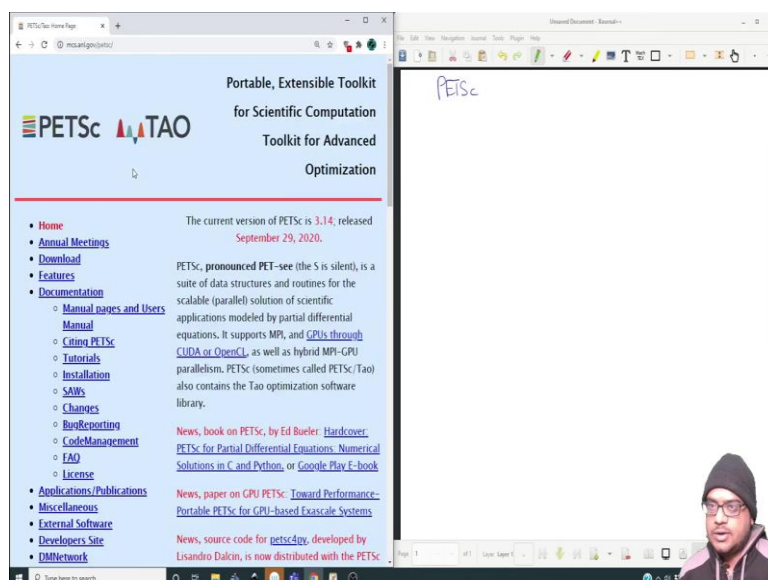
**Lecture - 29**  
**PETSc and MPI basics**

(Refer Slide Time: 00:29)



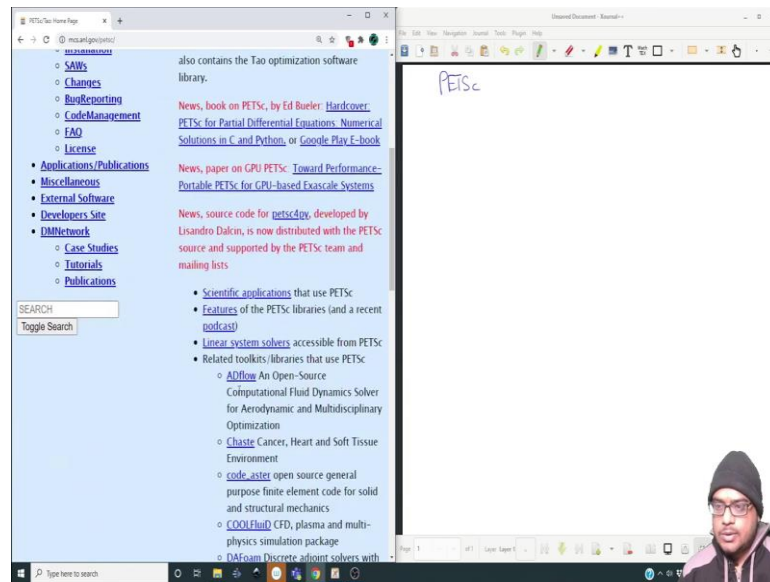
Hello everyone, to this 6th week where we are going to begin our journey with PETSc.  
So, what is PETSc?

(Refer Slide Time: 00:41)

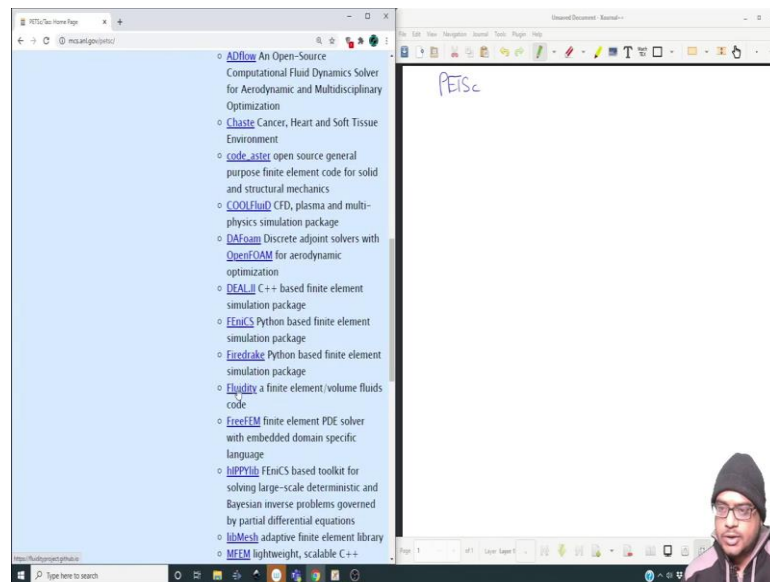


So, PETSc is collection of library, data structures, routines for solving large non-linear algebraic systems and the whole library is sort of built around the MPI interface that is the message passing interface, which enables one to run their programs over multiple processors. And we will see some examples of that.

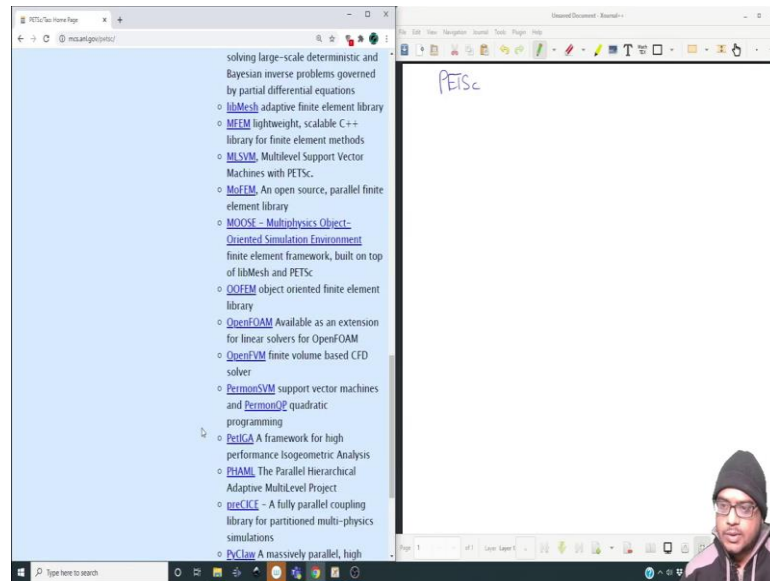
(Refer Slide Time: 01:21)



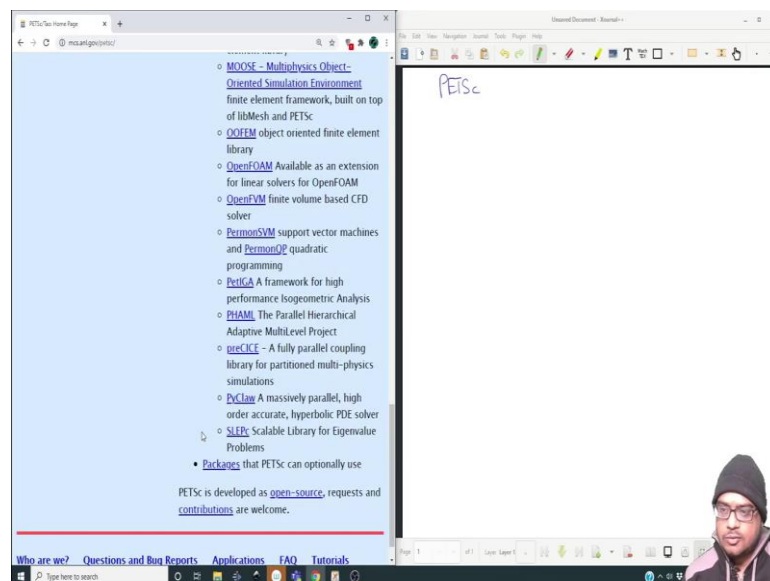
(Refer Slide Time: 01:23)



(Refer Slide Time: 01:40)

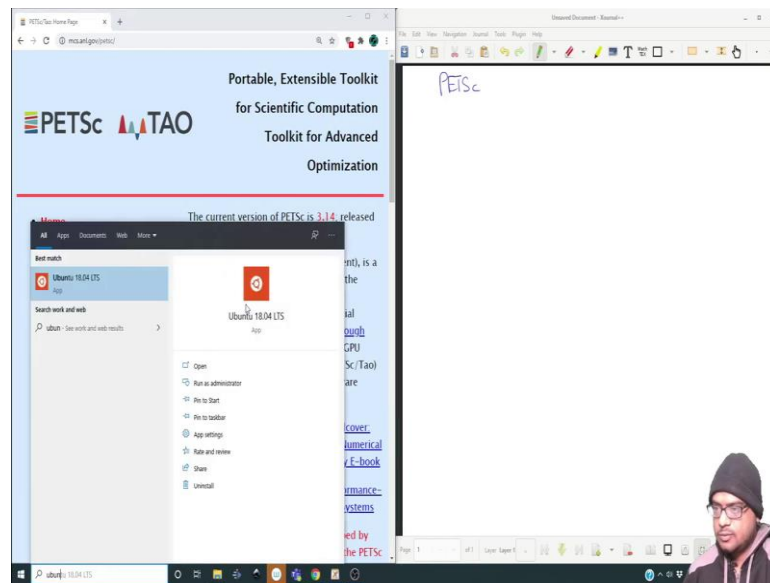


(Refer Slide Time: 01:44)

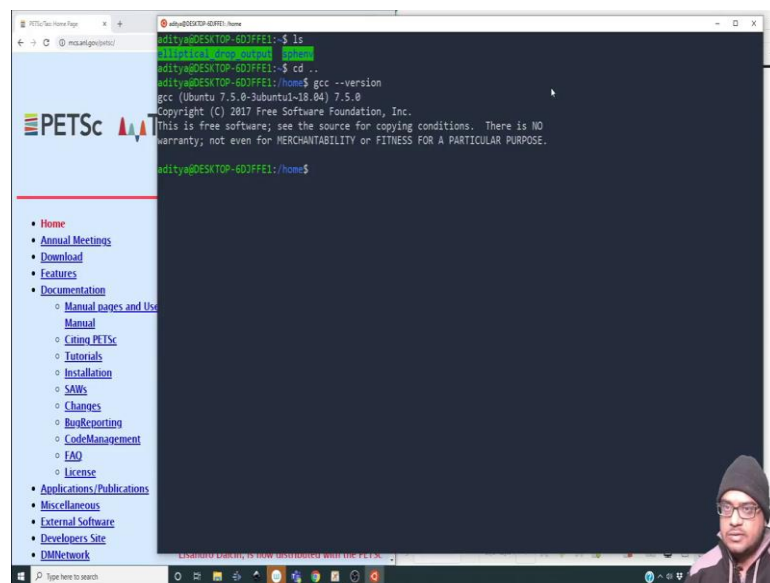


So, yeah, I mean it is being used in a host of different toolkits libraries such as ADflow, code aster, COOLFluid, FEniCS, fluidity, freeFEM, MOOSE, MoFEM, openFOAM and so on. So, this particular library offers you a lot of tools with which you can achieve great computational speed, efficient memory usage all through a very easy interface through the C programming language. So, we are working on Windows, but we can use the subsystem the windows subsystem for Linux to install Ubuntu as an app.

(Refer Slide Time: 02:22)



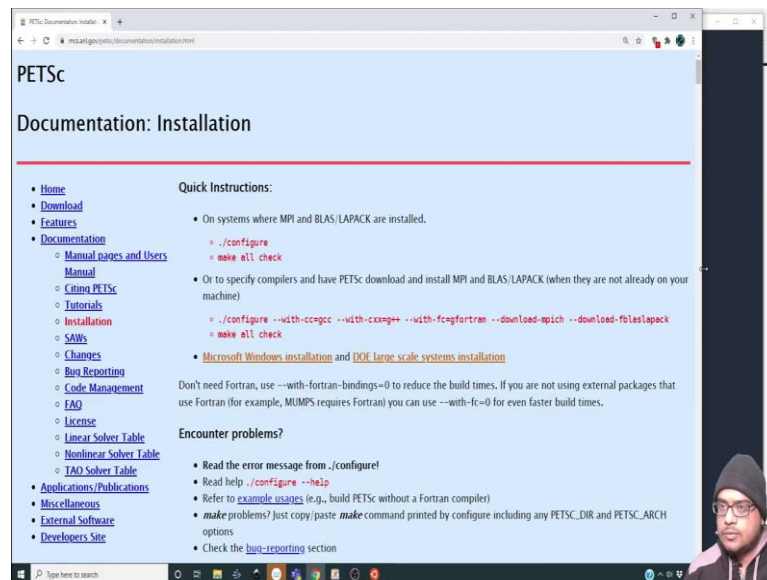
(Refer Slide Time: 02:24)



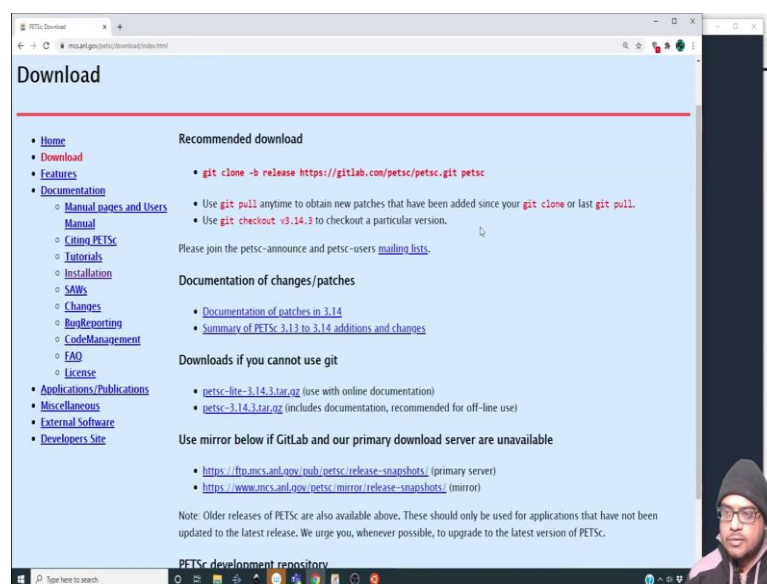
So, I have it installed already. So, when I launch Ubuntu I will be greeted by this. So, this is my desktop on work, I mean this is my Linux sort of virtualization inside windows. All the Linux commands work quite well. So, it is something Linux users do not have to do this. You can directly install PETSc.

So, let me just walk you through on how you would go around installing PETSc. So, first of all I will assume you have GCC installed. So, on my PC I have GCC 7.5.0, and in PETSc on this website you have to go to installation, alright.

(Refer Slide Time: 03:24)



(Refer Slide Time: 03:33)

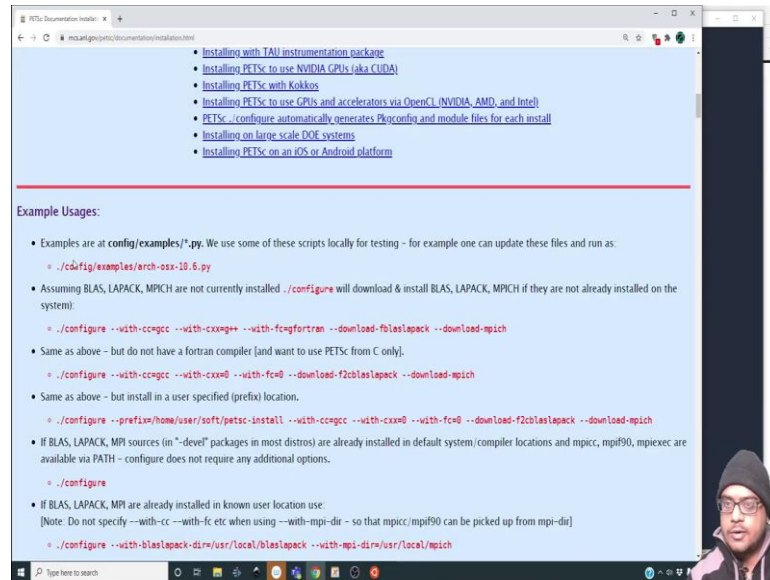


So, how to go about installing this? So, first thing you have to download the thing. So, you have to install git, you have to get the latest release, and once you get the latest release you have to go to the you have to unzip the thing ok, because you will download it as a zip file. Once you unzip it, you can configure it like this you have to sort of build the set of files that you have downloaded, alright.

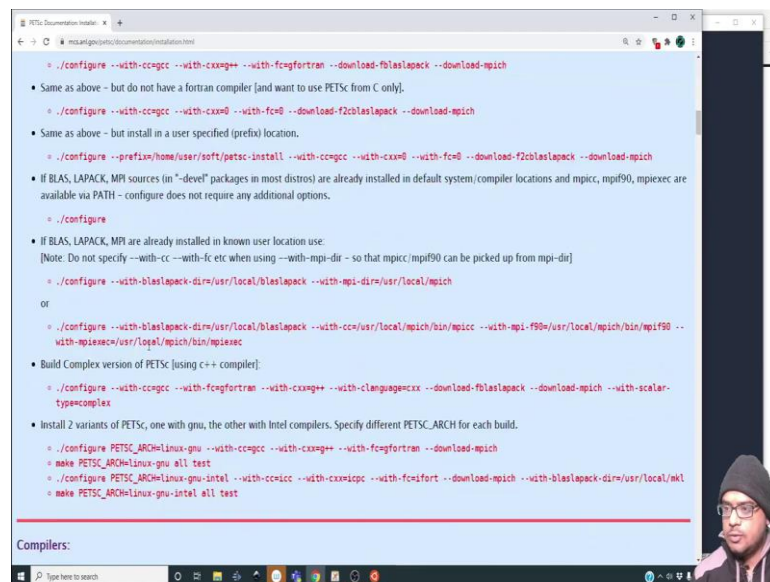
So, the configure statement tells what C compiler it has to use, what C + + compiler it has to use, what Fortran compiler it has to use, whether it has to download MPI, whether

it has to download BLAS and LAPACK. So, I would suggest to keep the configure statement as it is shown, so that the MPI that you may already have installed, does not conflict with the MPI that you will install through this particular build, alright.

(Refer Slide Time: 04:56)



(Refer Slide Time: 05:01)



So, once you do that you do a make all check, and then you follow whatever you see on screen. So, eventually you will end up with lines such as this. You will do this particular thing, you will configure it with PETSc arch and PETSc dir, which specifies the PETSc directory and the PETSc architecture.



(Refer Slide Time: 05:27)

```
aditya@DESKTOP-GDJFFE1: /home$ PETSC_DIR
-bash: /mnt/f/petsc/petsc-3.13.2: Is a directory
aditya@DESKTOP-GDJFFE1: /home$ PETSC_ARCH
arch-linux-c-debug: command not found
aditya@DESKTOP-GDJFFE1: /home$ PETSC_DIR/$PETSC_ARCH/
bin/          include/      obj/          tests/
externalpackages/ lib/          share/
aditya@DESKTOP-GDJFFE1: /home$ PETSC_DIR/$PETSC_ARCH/bin/
-bash: /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/: Is a directory
aditya@DESKTOP-GDJFFE1: /home$ cd $PETSC_DIR/$PETSC_ARCH/bin/
aditya@DESKTOP-GDJFFE1: /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin$ ls
hydra_nameserver  hydra_mpi_proxy  mpicc         mpiexec.hydra  mpiF90  mpirun  parikill
hydra_parallel    mpi4w             mpi4wversion  mpiexec        mpiF77  mpirort  mpivers
aditya@DESKTOP-GDJFFE1: /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin$ PETSC_DIR/$PETSC_ARCH/bin/mpiexec
-bash: /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpiexec: No such file or directory
aditya@DESKTOP-GDJFFE1: /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin$ PETSC_DIR/$PETSC_ARCH/bin/mpiexec
[mpiexec@DESKTOP-GDJFFE1] set_default_values (ui/mpich/utills.c:1578): no executable provided
[mpiexec@DESKTOP-GDJFFE1] HYD_ui_mpi_get_parameters (ui/mpich/utills.c:1783): setting default values failed
[mpiexec@DESKTOP-GDJFFE1] main (ui/mpich/mpiexec.c:148): error parsing parameters
aditya@DESKTOP-GDJFFE1: /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin$ cd ..
aditya@DESKTOP-GDJFFE1: /mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug$ cd ../..
aditya@DESKTOP-GDJFFE1: /mnt/$ cd ..
aditya@DESKTOP-GDJFFE1: /mnt/$ cd /
aditya@DESKTOP-GDJFFE1: /mnt/$ ls
ls: cannot access 'hiberfil.sys': Permission denied
ls: cannot access 'pagefile.sys': Permission denied
ls: cannot access 'swapfile.sys': Permission denied
aditya@DESKTOP-GDJFFE1: /mnt/$ cd /
aditya@DESKTOP-GDJFFE1: /mnt/c/Users/Admin/Dropbox$ cd /
aditya@DESKTOP-GDJFFE1: /mnt/c/Users/Admin/Dropbox$ ls
Documents and Settings*  IntelOptaneData  Program Files (x86)*  System Volume Information*  swapfile.sys
DumpStack.log.tmp       OS                Python39              Users
hiberfil.sys            Secure            Recovery              windows
pagefile.sys            System Volume Information*  swapfile.sys
hiberfil.sys            Users
swapfile.sys            windows
hiberfil.sys            windows
swapfile.sys            hiberfil.sys
hiberfil.sys            swapfile.sys
```

So, once you start installing you will actually be prompted to use those. So, keep a note of those, the variables that you will be requiring are PETSc dir. So, for me it is installed inside this, right. And the other thing will be PETSc arch. So, command not found, but this is the thing. So, in fact, the working directory for PETSc where it is installed its PETSc dir \ PETSc arch \ bin and inside this if I do this I will have all the different.

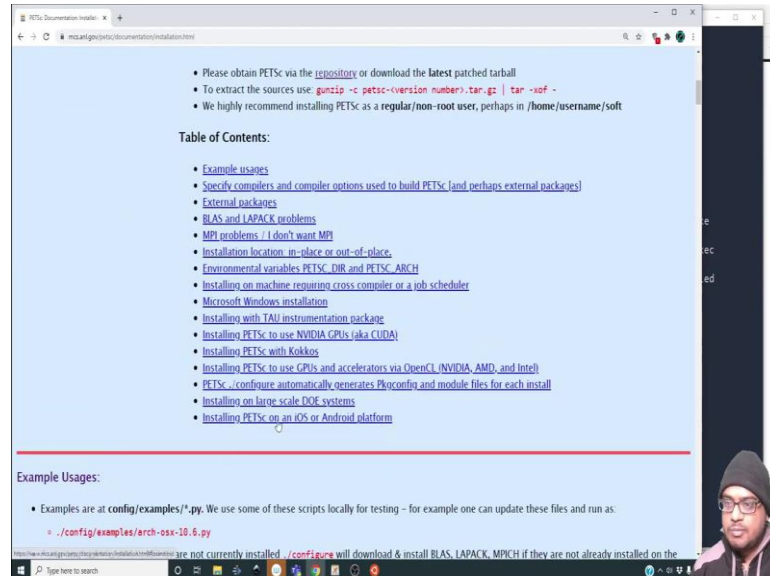
So, inside this I will find MPI exec. So, I must use this particular version of MPI to run the programs over multiple processors. And in fact, if I do not want to do that. You may not wish to run this over multiple processors , but what is the use, I mean the true power of PETSc comes in when you are running it on multiple processors.

So, you must use this particular MPI ch or rather MPI exe. So, you must use this particular MPI exec to run your program on how many processors you want, alright. So, we will look into this a bit later. But this PETSc dir and PETSc arch they will be prompted to you once you start installing, alright.

So, you can fetch which MPI version you are using. But, yeah, I mean if you are building it like this you will have to use this particular MPI, otherwise it will not sort of because it is not come the MPI that is already installed its not compiled with PETSc. Rather PETSc is not compiled with that it will not run it on multiple processors. Well, this is a very easy work around, you can go to your bash hash file and you can declare the MPI exac

version to be this particular version instead of the default version. But anyway all those are minor issues, alright.

(Refer Slide Time: 07:48)



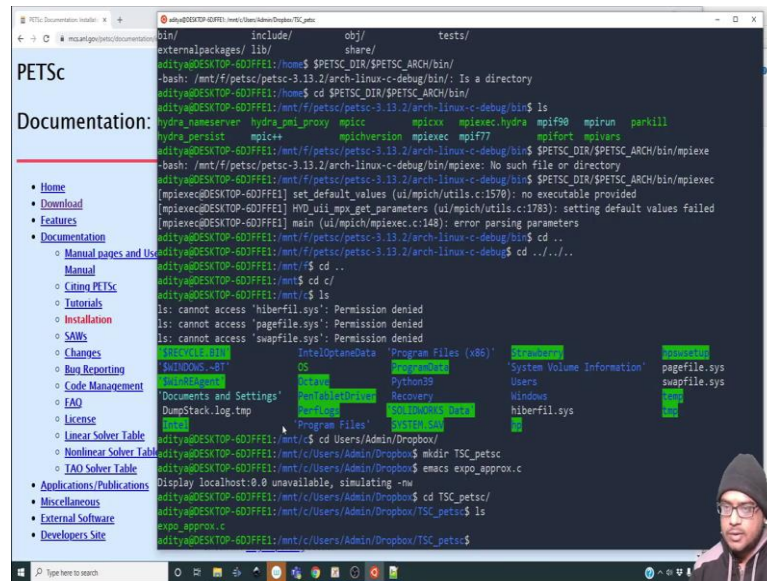
So, nowadays you have PETSc running on iOS or Android. So, it is really going scalable. You can use GPUs and NVIDIA, AMD GPUs. So, they have done a wonderful job of creating a massively scalable library. And it is not a recent library, it is I think the first release came from the year 1994. So, it is something which is quite old and it has been developed over decades, alright. So, yeah. So, let me go back to let me create a new folder in C. So, let me go to my, let me make a new directory, alright.

So, now I have created a new directory, and you can do two things, you can of course, work within this terminal if you are comfortable with it you can use any editor. So, suppose I want to create a file, let us say like 29.C.

So, or let us give it something more descriptive. So, the first example that I want to do is finding out the value of an exponential using a series approximation, a MacLaurin series. So, what I can do is I can use emacs and I can say Taylor, no, not Taylor. exponential approx.C.

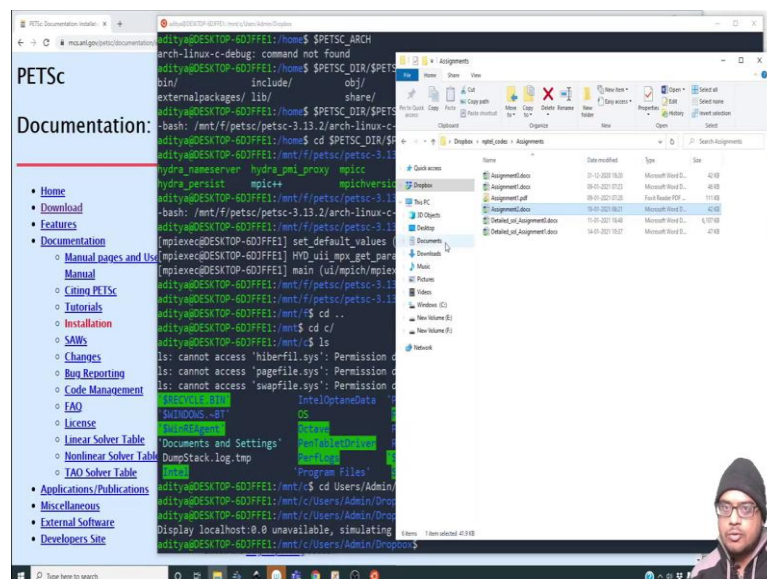


(Refer Slide Time: 09:58)

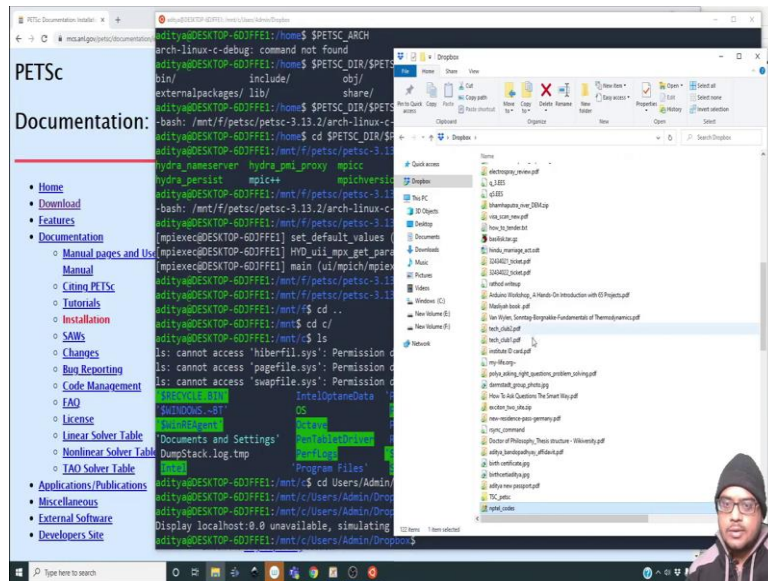


So, what this does is it will open the editor inside the terminal itself and if you are comfortable using emacs. You can do all the programming here itself. But if you are not comfortable doing this you can go to your drop box directory. I mean see not everyone may be comfortably doing things in the terminal, but you can go to the drop box folder, you can create the file over there and you can compile it simply on the terminal.

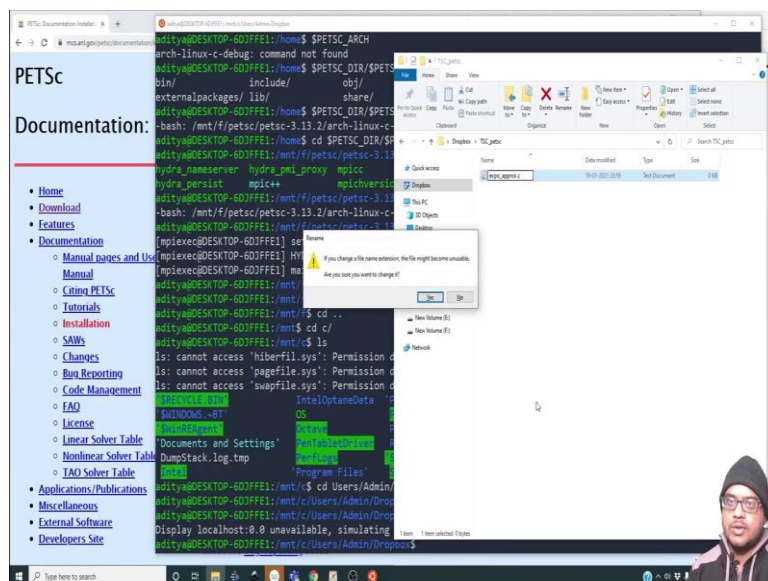
(Refer Slide Time: 10:33)



(Refer Slide Time: 10:35)



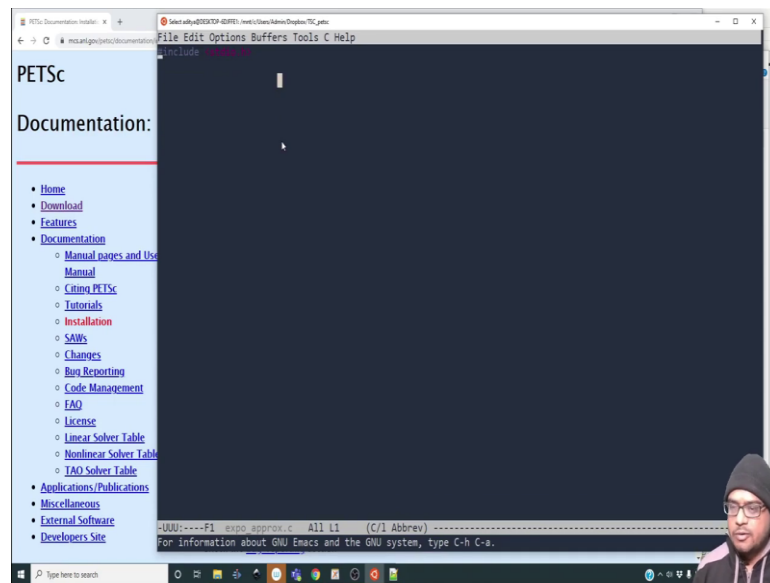
(Refer Slide Time: 10:45)



So, what I mean is now I can go to the terminal, and I can create a new file over here. So, I can call it something like this.

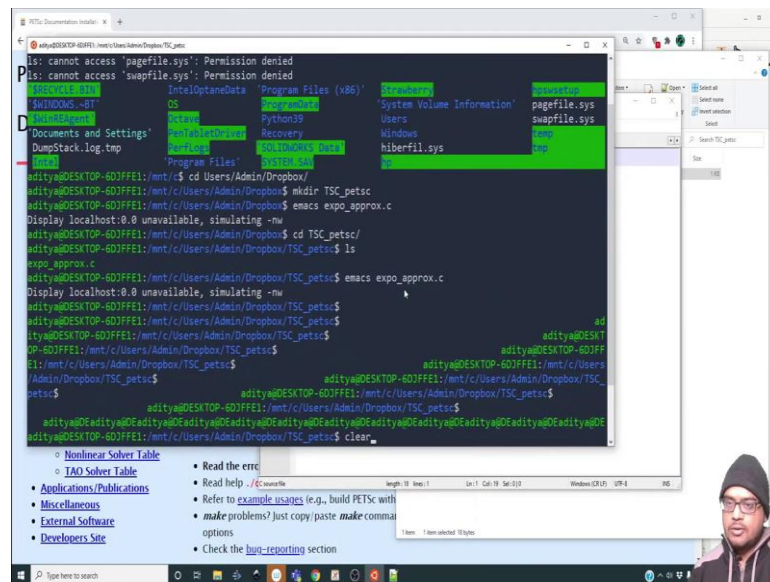


(Refer Slide Time: 11:30)



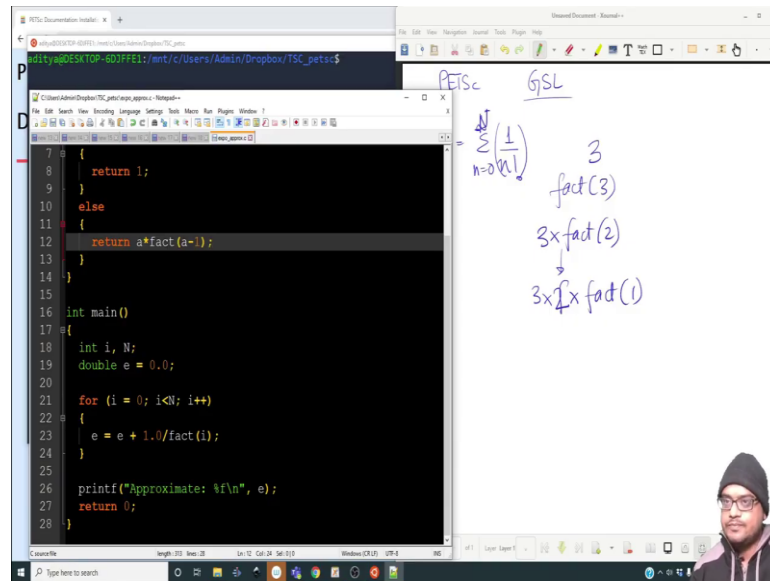
So, if I do emacs expo. So, I have the hash include stdio.h over here, alright. The color contrast is a bit off, but that is ok. So, let me exit. So, I will do it in over here, no problem. And we will just use the terminal to compile our things. So, let me clear all this,.

(Refer Slide Time: 11:44)



So, we want to create a file which will help us approximate an exponential.

(Refer Slide Time: 11:57)



So, the MacLaurin series is as follows. So,  $e = \sum_{n=0}^{\infty} \left(\frac{1}{n!}\right)$ . This is the way you have to define

it. So, let us first create a C program which will help us to do this without using PETSc, and then we will see the PETSc version later. So, let us include math.h. So, int main, return 0. So, now, we need to declare some variables, alright. So, int i, double e.

So, let us store the solution inside e, and yeah, let us declare it as 0, let us initialize it as 0. So, let me change the theme, alright. This is a much better contrast. So, what we can do is inside this program we must do a loop over this. So, we must declare what n is going to be. So, this sum is obviously, we cannot do till infinity. So, let us do the sum till capital N, alright. So, let me declare N as a integer as well, right. So, now, we must do for i = 0, i < N, i + +.

So, then we must do e = e + 1.0 divided by; so, you must do a factorial of i, alright. So, we must not define what the factorial function will be because we do not have those predefined in C as such. I mean of course, we can use the gnu scientific library that is there is something called as the GSL, which also gives you a host of libraries which are useful for computations, but anyway we do not need to do that. So, we need to define what this is, and in the end, let us we have to print.

So, we will do print f, % f \ n and we have to print e. So, so far, we have not defined what factorial will be. So, we must declare it is a long int, and the reason I am using long

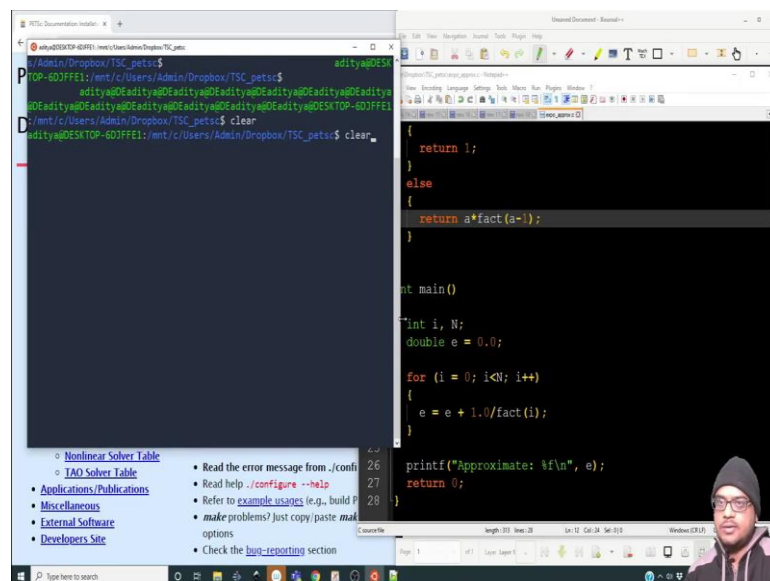


int because factorials tend to become really large after a while the values of factorials it becomes really large. So, we will declare fact and it will take as an input integer, let us say a, alright. So, it must return something, so return r. So, long, ok; so, we do not need to do this. So, the way to do a factorial is if a = 0, right or a = 1, then we must return 1, else we must return a times fact of a - 1.

So, this is a recursive way how to defining the factorial of a number a, alright. So, this is nothing but recursion because once a is large is neither 0 or 1, it will simply suppose a is 3. What happens suppose a is 3? So, it will call 3!. So, when we call 3! it will check whether 3 is 0 or 1, if it is not then it will go to this. So, it will then return  $3 \times 2!$ , but  $3 \times 2!$  will involve  $3 \times 2 \times 1!$ . So, if you unroll the function stack this is what you will see.

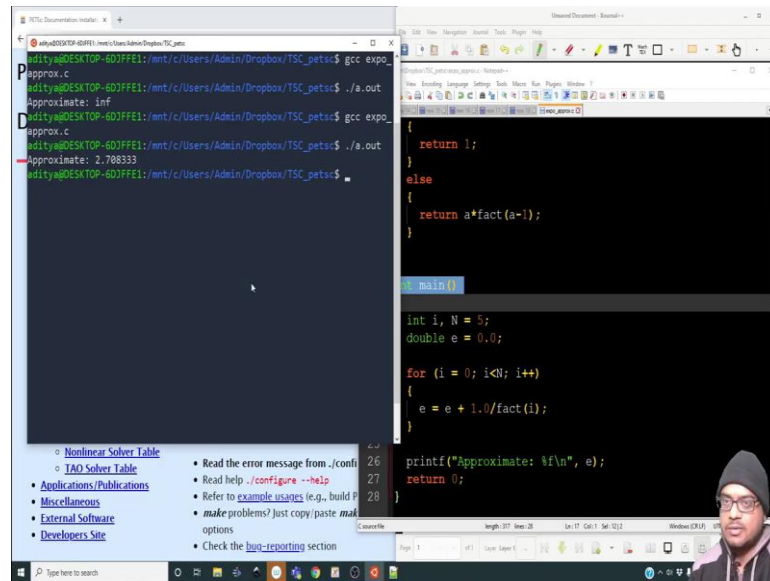
And something like this you might have done in your first year programming or even 11th, 12th. But regardless I am just showing you how one would go about doing this for those of you have not done it. But I highly recommend you grab hold of a C programming book to understand what is really going on. So, this is the factorial way of writing it and I am assuming some working knowledge of C programming in this this is not really a course where you will learn C programming, alright. So, yeah, that is pretty much it.

(Refer Slide Time: 17:24)



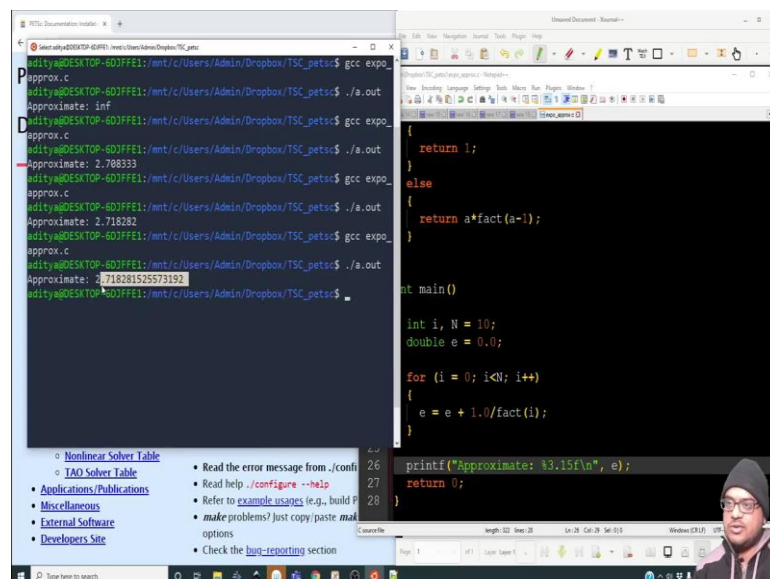


(Refer Slide Time: 17:36)

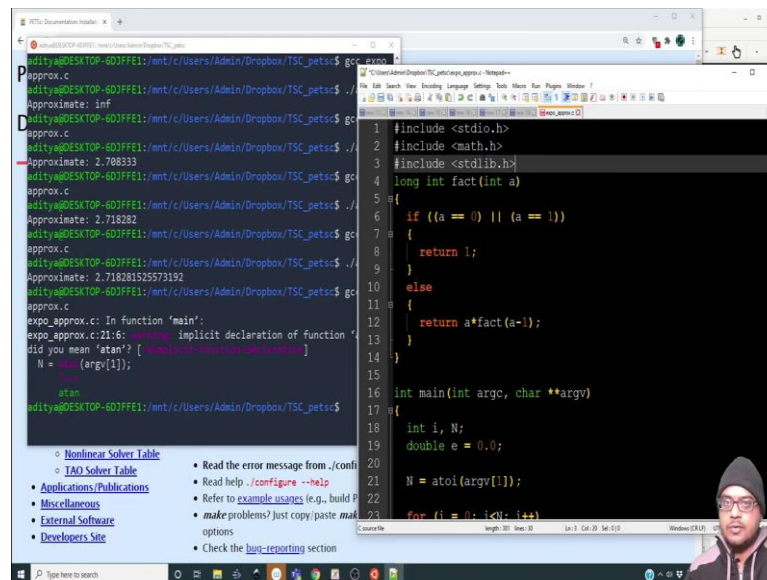


Then we will go to our terminal let us do GCC, and let me clear the screen. So, GCC exp0, alright. Let us simply do this, PETSc if there is some error there is no error. \ a . out. But I have not defined what N is, so there will be an error, ok. So, I have not defined what N is. So, I must define N as a certain number of terms. So, let us say 5, let me save this. So, it gives us an approximation of 2.70833.

(Refer Slide Time: 18:12)



(Refer Slide Time: 18:48)



```
aditya@DESKTOP-6D3FFE1:/mnt/c/Users/Admin/Dropbox/TSC_getsc$ gcc approx.c
aditya@DESKTOP-6D3FFE1:/mnt/c/Users/Admin/Dropbox/TSC_getsc$ ./approx.c
Approximate: inf
aditya@DESKTOP-6D3FFE1:/mnt/c/Users/Admin/Dropbox/TSC_getsc$ gcc approx.c
aditya@DESKTOP-6D3FFE1:/mnt/c/Users/Admin/Dropbox/TSC_getsc$ ./approx.c
Approximate: 2.788333
aditya@DESKTOP-6D3FFE1:/mnt/c/Users/Admin/Dropbox/TSC_getsc$ gcc approx.c
aditya@DESKTOP-6D3FFE1:/mnt/c/Users/Admin/Dropbox/TSC_getsc$ ./approx.c
Approximate: 2.718282
aditya@DESKTOP-6D3FFE1:/mnt/c/Users/Admin/Dropbox/TSC_getsc$ gcc approx.c
aditya@DESKTOP-6D3FFE1:/mnt/c/Users/Admin/Dropbox/TSC_getsc$ ./approx.c
Approximate: 2.718281525573192
aditya@DESKTOP-6D3FFE1:/mnt/c/Users/Admin/Dropbox/TSC_getsc$ gcc approx.c
expp approx.c: In function 'main':
expp approx.c:21:6: warning: implicit declaration of function 'atan' did you mean 'atan'? [-Wimplicit-function-declaration]
   N = atan(argv[1]);
       ^
aditya@DESKTOP-6D3FFE1:/mnt/c/Users/Admin/Dropbox/TSC_getsc$
```

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <stdlib.h>
4 long int fact(int a)
5 {
6     if ((a == 0) || (a == 1))
7     {
8         return 1;
9     }
10    else
11    {
12        return a*fact(a-1);
13    }
14 }
15
16 int main(int argc, char **argv)
17 {
18     int i, N;
19     double e = 0.0;
20
21     N = atoi(argv[1]);
22
23     for (i = 0; i <= N; i++)
```

So, if I increase this let us say to 10, let me recompile and let me run the file, we get a much better approximation. In fact, let us increase the number of decimal places that we can print, so let us say 3.15 f. So, we will get 15 decimal places. So, let me recompile, alright. So, this is what the approximation is. So, this is how you can evaluate an exponential using plane C language.

So, we have not done anything fantastic. In fact, we can use int argc char star star argv. We can pass command line arguments to do certain things. So, through this command line argument we can avoid doing this, and we will simply declare N to be an integer, but what we will do is we will pass how many number of terms we would like.

So, N would be a to i of argv 1. So, argv will be storing as strings the things you will pass. So, argv 0 is always the name of the function, argv 1 is the first argument that you give to the function. So, that is why; so, everything else will be treated as string, so you have to use a to i; so, argument to integer, alright.

So, let us compile this and see how we can do this. So, let me save this. So, let me do GCC, ok. And well, the reason we have this error is we have not included the standard library. So, hash include stdlib . h, alright. So, now we should be fine, excellent.

(Refer Slide Time: 20:15)

The terminal window shows the following commands and output:

```
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out
Approximate: 2.788333
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ gcc expo_approx.c
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out
Approximate: 2.718282
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ gcc expo_approx.c
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out
Approximate: 2.718281525573192
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ gcc expo_approx.c
expo_approx.c: In function 'main':
expo_approx.c:21:6: warning: implicit declaration of function 'atoi'; did you mean 'atan'? [-Wimplicit-function-declaration]
   N = atoi(argv[1]);
       ^
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ gcc expo_approx.c
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out 20
Approximate: 2.718281828459046
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out 2
Approximate: 2.0800000000000000
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out 5
Approximate: 2.7883333333333333
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out 20
Approximate: 2.718281828459046
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out 58
Approximate: 2.718281828459046
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$
```

The whiteboard shows the following mathematical expressions:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$
$$3 \times \text{fact}(3)$$
$$3 \times \text{fact}(2)$$
$$3 \times \text{fact}(1)$$
$$\frac{1}{0!} + \frac{1}{1!} = 1 + 1 = 2$$

So, now, when we do this, we must pass the number of arguments. So, I can pass 20 as the argument. So, if I set 2, I get an approximation of 2, because when we have  $N = 2$ , what we have is  $e = 1$  over  $0$  factorial +  $1$  over  $1$  factorial, which is  $1 + 1$  is  $2$ .

(Refer Slide Time: 20:34)

The terminal window shows the following commands and output:

```
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ clear
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ clear
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ gcc expo_e
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out
Approximate: inf
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ gcc expo_e
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out
Approximate: 2.788333
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ gcc expo_e
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out
Approximate: 2.718282
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ gcc expo_e
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out
Approximate: 2.718281525573192
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ gcc expo_e
expo_approx.c: In function 'main':
expo_approx.c:21:6: warning: implicit declaration of function 'atoi'; c
tan'? [-Wimplicit-function-declaration]
   N = atoi(argv[1]);
       ^
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ gcc expo_e
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out 20
Approximate: 2.718281828459046
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out 2
Approximate: 2.0800000000000000
aditya@DESKTOP-6D3JFF1:~/mt/c/Users/Admin/Dropbox/TSC_petsc$
```

The whiteboard shows the following mathematical expressions:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$
$$3 \times \text{fact}(3)$$
$$3 \times \text{fact}(2)$$
$$3 \times \text{fact}(1)$$
$$e = \frac{1}{0!} + \frac{1}{1!} = 1 + 1 = 2$$

So, as we increase the number of terms. So, if I make it 5, we get this. If I make it 20, we get this. If I make it 50, we get this. And yes, this it does converge to the value of the exponential.

(Refer Slide Time: 21:13)

```

aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out
Approximate: 2.788333
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ gcc exp0_approx.c
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out
Approximate: 2.718282
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ gcc exp0_approx.c
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out
Approximate: 2.718281525573192
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ gcc exp0_approx.c
exp0_approx.c: In function 'main':
exp0_approx.c:21:6: warning: implicit declaration of function 'atan' [-Wimplicit-function-declaration]
   N = atan(argv[1]);
       ^
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ gcc exp0_approx.c
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out
Approximate: 2.718281828459046
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out 2
Approximate: 2.8800000000000000
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out 5
Approximate: 2.7883333333333333
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out 20
Approximate: 2.718281828459046
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out 50
Approximate: 2.718281828459046
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ gcc exp0_approx.c
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out 50
Approximate: 3.15f\n
Approximate: 3.15f\n
Approximate: 3.15f\n
Approximate: 3.15f\n
Approximate: 3.15f\n

```

So, suppose now you want to print out all the variables, not the variables, but the intermediate values as well. So, let us say you want to print the partial sum, alright. So, we go over here we say print f partial sum = % 3.15f \n and we will pass e, alright.

(Refer Slide Time: 21:30)

```

aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ gcc exp0_approx.c
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out
Approximate: 2.788333
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ gcc exp0_approx.c
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out 2
Approximate: 2.8800000000000000
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out 5
Approximate: 2.7883333333333333
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out 20
Approximate: 2.718281828459046
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out 50
Approximate: 2.718281828459046
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ gcc exp0_approx.c
aditya@DESKTOP-6D3FFE1: /mnt/c/Users/Admin/Dropbox/TSC_pets$ ./a.out 50
Approximate: 3.15f\n
Approximate: 3.15f\n
Approximate: 3.15f\n
Approximate: 3.15f\n
Approximate: 3.15f\n

```



(Refer Slide Time: 21:32)

```
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Approximate: 2.718281828459046
aditya@DESKTOP-6D3JFE1:~/nt/c/Users/Admin/Dropbox/TSC_gets$
```

```
9 }
10 else
11 {
12     return a*fact(a-1);
13 }
14 }
15
16 int main(int argc, char **argv)
17 {
18     int i, N;
19     double e = 0.0;
20     N = atoi(argv[1]);
21
22     for (i = 0; i<N; i++)
23     {
24         e = e + 1.0/fact(i);
25         printf("Iter: %d\t Partial sum = %3.15f\n", i, e);
26     }
27
28     printf("Approximate: %3.15f\n", e);
29     return 0;
30 }
31
```

So, let me recompile. So, that is the partial sum. So, the partial sum converges quite quickly. So, let us also print the iteration number, iter % d \ t and I will print the iteration number over here. So, this will allow us to see how those partial sums are evolving.

(Refer Slide Time: 21:58)

```
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Partial sum = 2.718281828459046
Approximate: 2.718281828459046
aditya@DESKTOP-6D3JFE1:~/nt/c/Users/Admin/Dropbox/TSC_gets$ gcc expc_approx.c
aditya@DESKTOP-6D3JFE1:~/nt/c/Users/Admin/Dropbox/TSC_gets$ ./a.out 10
Iter: 0 Partial sum = 1.0000000000000000
Iter: 1 Partial sum = 2.5000000000000000
Iter: 2 Partial sum = 2.6666666666666667
Iter: 3 Partial sum = 2.7083333333333333
Iter: 4 Partial sum = 2.7166666666666666
Iter: 5 Partial sum = 2.7180555555555555
Iter: 6 Partial sum = 2.7182539682539683
Iter: 7 Partial sum = 2.7182787698412780
Iter: 8 Partial sum = 2.7182815257319200
Iter: 9 Partial sum = 2.7182818255731920
Approximate: 2.718281828459046
aditya@DESKTOP-6D3JFE1:~/nt/c/Users/Admin/Dropbox/TSC_gets$
```

```
fact(a-1);
argc, char **argv)
0.0;
v[1]);
i<N; i++)
.0/fact(i);
ter: %d\t Partial sum = %3.15f\n", i, e);
printf("Approximate: %3.15f\n", e);
return 0;

```

So, let me compile it let me take 10 terms. So, first term 1, 2, 2.5, 2.6, so very quickly you see you get  $10^{-5}$  accuracy after just 9 terms, alright. So, what happens when we do not use long int? I want you to find this out. If I do not use long int and let me just give you a quick answer to that, but you should ideally go ahead and print them out.

(Refer Slide Time: 22:32)

The screenshot shows a terminal window on the left and a code editor on the right. The terminal displays the output of a C program that calculates the sum of the reciprocals of the first 10 natural numbers. The output shows the partial sum for each iteration and the final approximate value of 2.718281525573192. The code editor shows the source code of the program, which includes a factorial function and a main function that iterates from 1 to 10, calculating the partial sum and printing it for each iteration. The code is as follows:

```
10 else
11 {
12     return a*fact(a-1);
13 }
14 }
15
16 int main(int argc, char **argv)
17 {
18     int i, N;
19     double e = 0.0;
20
21     N = atoi(argv[1]);
22
23     for (i = 0; i<N; i++)
24     {
25         e = e + 1.0/fact(i);
26         printf("%d\t %ld\n", i, fact(i));
27     }
28
29     printf("Approximate: %3.15f\n", e);
30     return 0;
31 }
32
```

So, let me just print out percentage  $ld \setminus n$ , and I will set  $\% d \setminus t$  and print  $i$  and I will print  $i!$ , alright.

(Refer Slide Time: 23:01)

The screenshot shows a terminal window on the left and a code editor on the right. The terminal displays the output of a C program that calculates the sum of the reciprocals of the first 10 natural numbers. The output shows the partial sum for each iteration and the final approximate value of 2.718281525573192. The code editor shows the source code of the program, which includes a factorial function and a main function that iterates from 1 to 10, calculating the partial sum and printing it for each iteration. The code is as follows:

```
fact(a-1);
argc, char **argv)
0.0;
gv[1]);
i<N; i++)
.0/fact(i);
t %ld\n", i, fact(i));
printf("Approximate: %3.15f\n", e);
return 0;
```



(Refer Slide Time: 23:07)

```
5040
48320
362880
Approximate: 2.718281525573192
D
1
1
2
6
24
120
720
5040
48320
362880
39916800
479001600
6227020800
87178291200
1307674368000
20922789888000
355687428096000
6402373705728000
121645100488832000
Approximate: 2.718281828459045
aditya@DESKTOP-6D3FFE1:~/mnt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out 20
Nonlinear Solver Table
IAO Solver Table
Applications/Publications
Miscellaneous
External Software
Developers Site
• Read the error message from ./confi
• Read help ./configure --help
• Refer to example usages (e.g., build P
• make problems? Just copy/paste mak
options
• Check the bug-reporting section
```

So, let me comment out this line, we do not need the partial sum. I am just trying to show you how large those integers are. Still, it is not that large. So, maybe for 20, ok. So, integers quickly become quite large.

If you do not use the long int you will run into floating or rather overflow exceptions because you are overflowing the range that you can represent using the number of bytes GCC allocates to integers. So, that is why you need to use a long int, great.

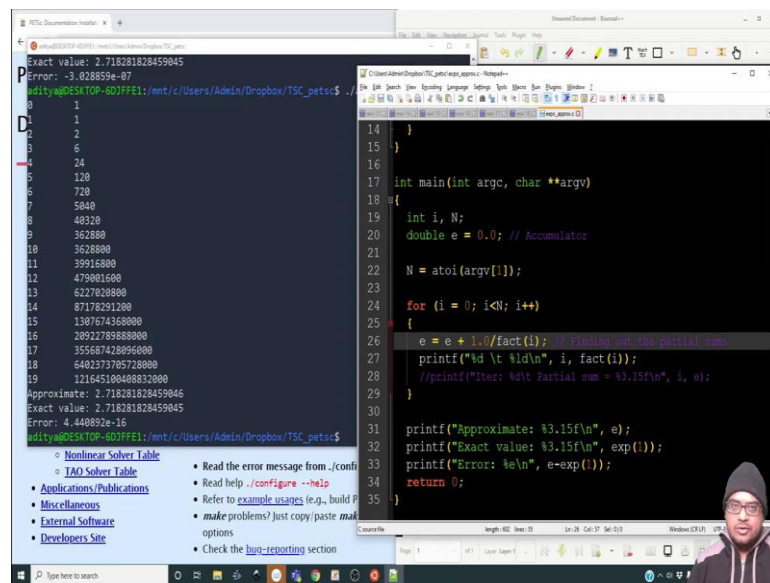
(Refer Slide Time: 23:32)

```
24
120
720
5040
48320
362880
Approximate: 2.718281525573192
Exact value: 2.718281828459045
Error: -3.028859e-07
aditya@DESKTOP-6D3FFE1:~/mnt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out 10
1
1
2
6
24
120
720
5040
48320
362880
Approximate: 2.718281525573192
Exact value: 2.718281828459045
Error: -3.028859e-07
aditya@DESKTOP-6D3FFE1:~/mnt/c/Users/Admin/Dropbox/TSC_petsc$ ./a.out 20
Nonlinear Solver Table
IAO Solver Table
Applications/Publications
Miscellaneous
External Software
Developers Site
• Read the error message from ./confi
• Read help ./configure --help
• Refer to example usages (e.g., build P
• make problems? Just copy/paste mak
options
• Check the bug-reporting section
```

So, one thing which I forgot to show we can find out the actual error. What we can do is we can write down the exact value as returned by the math library as  $\exp(1)$  and we can print out the error the absolute error as  $\%e - \exp(1)$ .

So, let us save this, let me compile. So, the error is  $10^{-7}$  forgot to back \ n the (Refer Time: 24:20) carriage (Refer Time: 24:22) compile. So, the error is  $10^{-7}$ , ok.

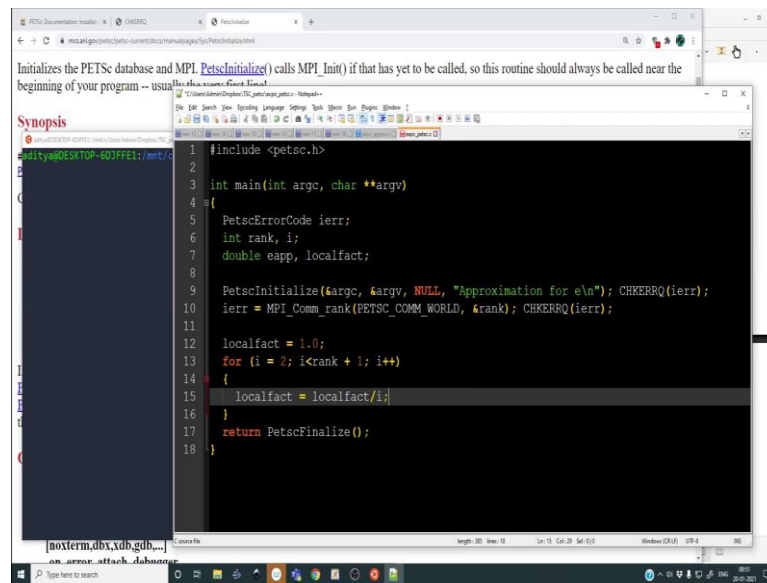
(Refer Slide Time: 24:35)



So, for 10 terms that is. So, that means, 20 terms  $10^{-16}$  is machine precision. So, this small program helps us get the job done. We have not made use of any multi-processor things we just created a function and it uses recursion to find out factorials and you used a very simple accumulator.

So, this e is actually an accumulator e because it is accumulating the partial sum ok. This is finding out the partial sum. This is how we have to work. So, now we will move on to the parallel implementation using PETSc. Let us create a new file and we call it expo PETSc . c. So, we are going to use the inbuilt PETSc data types and commands to find out the approximation for exponential.

(Refer Slide Time: 25:36)



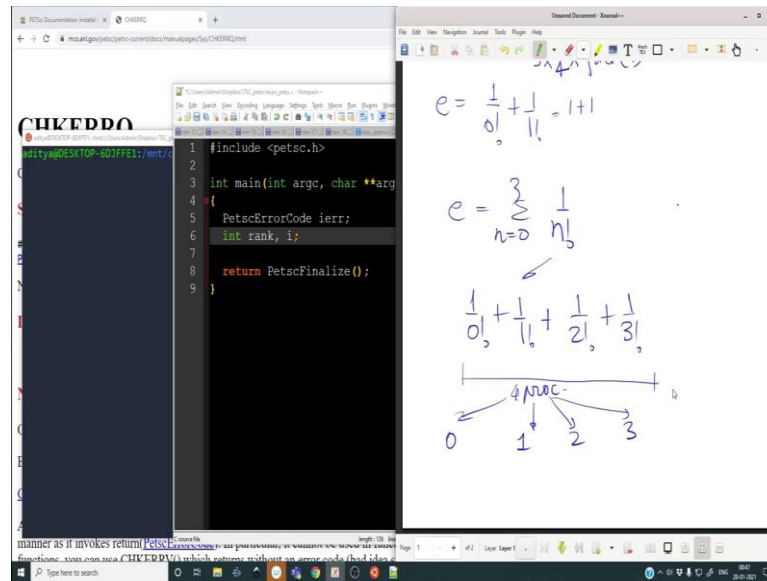
```
1 #include <petsc.h>
2
3 int main(int argc, char **argv)
4 {
5     PetscErrorCode ierr;
6     int rank, i;
7     double eapp, localfact;
8
9     PetscInitialize(&argc, &argv, NULL, "Approximation for e\w"); CHKERRQ(ierr);
10    ierr = MPI_Comm_rank(PETSC_COMM_WORLD, &rank); CHKERRQ(ierr);
11
12    localfact = 1.0;
13    for (i = 2; i < rank + 1; i++)
14    {
15        localfact = localfact/i;
16    }
17    return PetscFinalize();
18 }
```

So, first things first hash include PETSc . h int main and yeah int argc char star star argv, for that we sort of PETSc code that you write has to eventually finished with return PETSc finalize. So, this command helps you to gather all the threads. So, because you will be using the MPI either explicitly or implicitly PETSc, you have to do this PETSc finalize.

If you do not do this you will get very weird behavior now and do not forget to do this, and it is sort of almost magnetic. But then, the next thing we will have will be PETSc initialize rather before initializing let us declare some variables. The first variable will be the inbuilt data type PETSc error code ierr, and the PETSc error code is something which is 0. If everything is working properly if it does not work properly then it will return 1, 2, depending on the error type that you encounter.

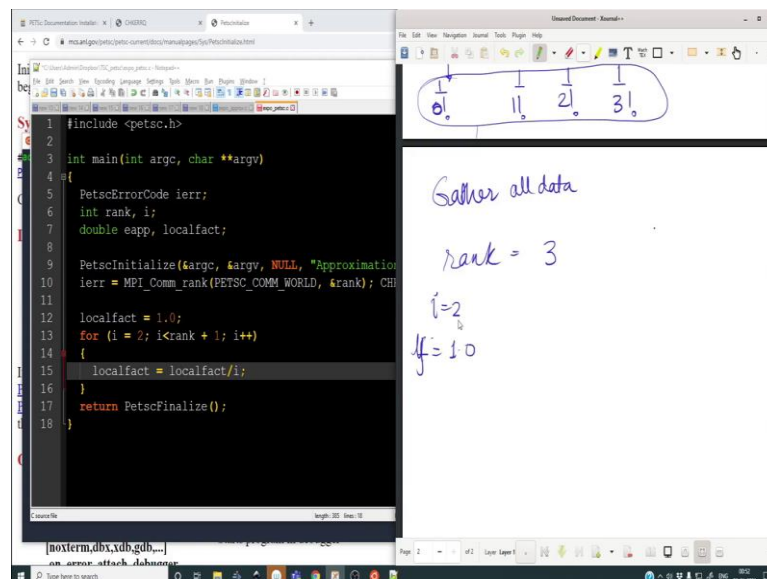
So, it is ordinarily of type int, but there is some additional data structures associated with the data type PETSc error code. So, this is something which you will be using as well. Then we need int rank to hold the rank of the processor that we are working with. Additionally, we will need i for running a loop. So, why do we bother with the rank?

(Refer Slide Time: 27:33)



So, suppose you want to find  $e = \text{summation } 1 \text{ by } n \text{ factorial for } n = 0 \text{ to } 3$ , so essentially what we will have  $e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!}$ . So, what our strategy will be is to declare 4 processors or 4 processes and each of these 4 processes will have a certain rank because there will be some process with rank 0, rank 1, rank 2, rank 3, right.

(Refer Slide Time: 28:08)



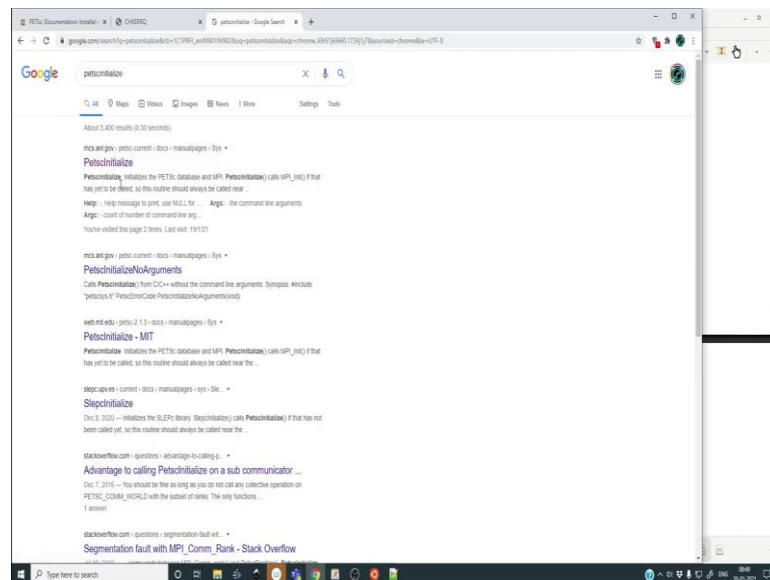
So, once you do have these ranks then what will happen? We would like to find the factorial of this one or rather 1 by factorial of this one, 1 by factorial of this one, 1 by

factorial of this one and 1 by factorial of this one. When we have; once we have these 4 factorials and then gather all the data that the different processes have, so, it will simply sum over all these. But essentially it is a reduction operation over going from 4 processes to one central process, ok. This is what the strategy is. So, hence we need to hold the rank of the process inside this, alright.

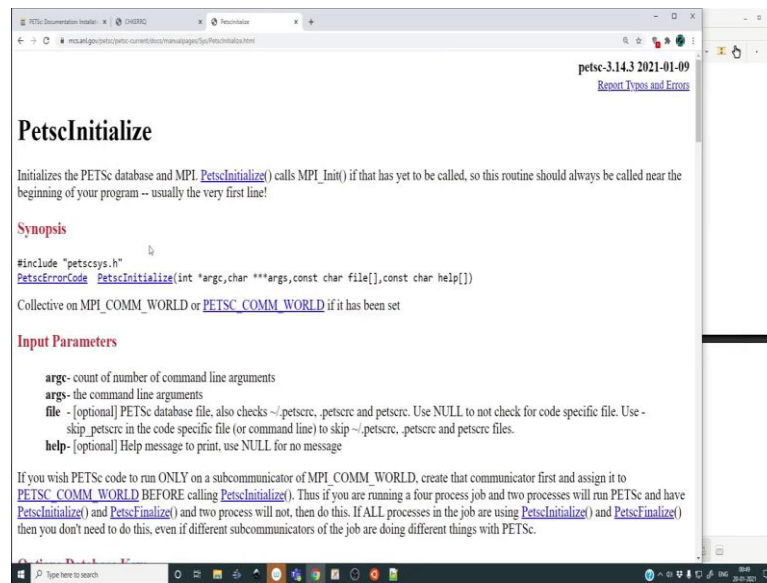
Then what we will do is ok we need a double data type for the e approx and the local factorial, ok. So, we will see what it means just in a moment. Then, what we will do is initialize the; so, we will initialize PETSc meaning it will create all the MPI back it will run background on the MPI initialization steps PETSc initialize the arguments we initialize will be and.

So, the address of argc, the address of argv then the PETSc database in this case it is null, then the help text, alright. So, this is how you initialize. So, most of the initializations which will look like this, nothing much will change, ok. And you can look in the function reference to see exactly what it does, ok.

(Refer Slide Time: 30:01)

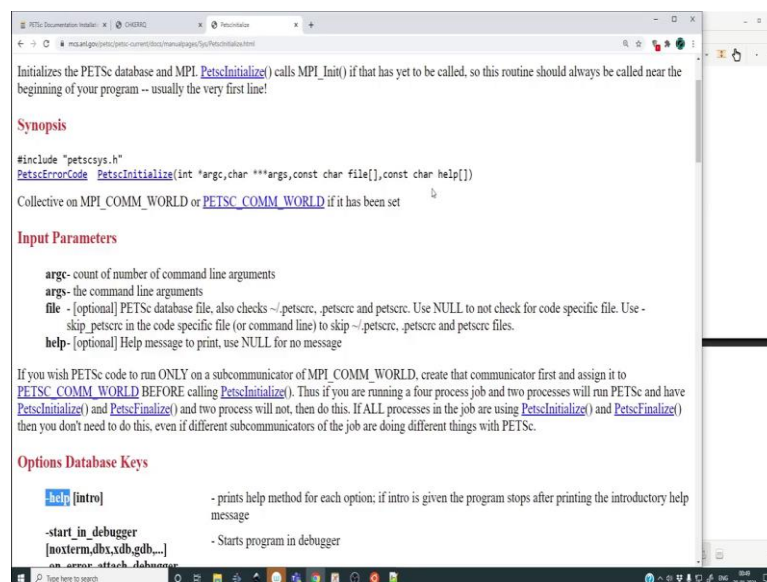


(Refer Slide Time: 30:03)



It initialize the database and MPI, and it has to be the very first line. Takes in inputs as `argc` `argv`, and the point is to run the address of those, then the database file and the help file, ok.

(Refer Slide Time: 30:25)



You can also pass it as null, but; so, if you type down the executable and write `- help` it will push this (Refer Time: 30:35) whatever you would, right ok what we do is we have initialized this created of the MPI things in the background. So, we do not need to bother about all these. So, after declaring this we have to usually check for an error. If there is



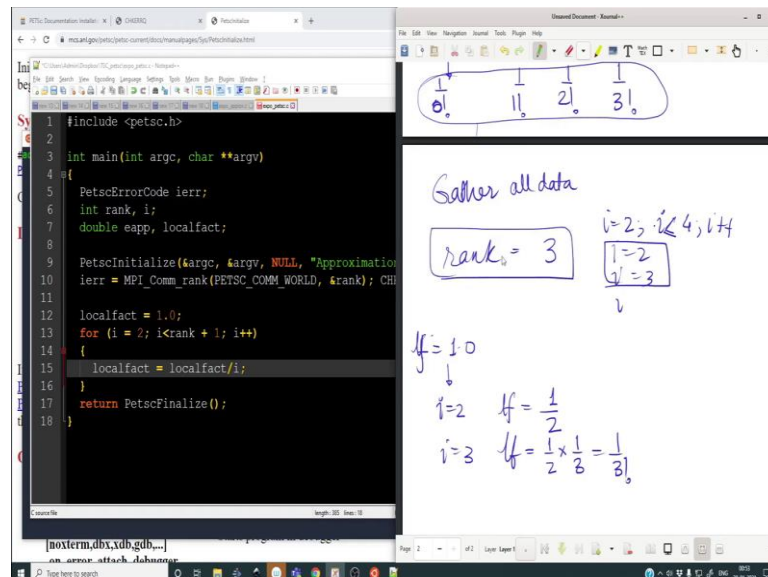
an error, so the way to do it is `CHKERRQ ierr`. So, it checks whether there is an error, ok.

So, now what we will do is we will create the ranks. So, we will create the ranks `ierr = MPI_Comm_rank`. So, it will create the different thread, ok. And the way to declare this PETSc comm world, then will pass the address of rank then, we will as usual check for error, alright.

Now, this particular line is important. It will create all the threads that we need depending on how many threads we ask we go as to allocate to this particular program. So, it will store all the different ranks of the thread inside the address of rank. So, after this point we will have n number of threads working together, not together but in parallel, ok.

Now, we will do `localfact = 1`. So, for each thread the factorial has to initialize with 1 or  $i = 2, i < rank + 1, i++$  then what we will do is `localfact = localfact / i`, essentially what we are achieving through this. So, let us look at this. So, suppose the rank is 3, (Refer Time: 32:55) So, for  $i = 2$ ; so, let us see local fact is 1 initially.

(Refer Slide Time: 33:10)



The image shows a code editor on the left and a handwritten slide on the right. The code editor displays the following C code:

```

1 #include <petsc.h>
2
3 int main(int argc, char **argv)
4 {
5     PetscErrorCode ierr;
6     int rank, i;
7     double eapp, localfact;
8
9     PetscInitialize(&argc, &argv, NULL, "Approximation");
10    ierr = MPI_Comm_rank(PETSC_COMM_WORLD, &rank); CHKERRQ(ierr);
11
12    localfact = 1.0;
13    for (i = 2; i < rank + 1; i++)
14    {
15        localfact = localfact / i;
16    }
17    return PetscFinalize();
18 }

```

The handwritten slide on the right contains the following content:

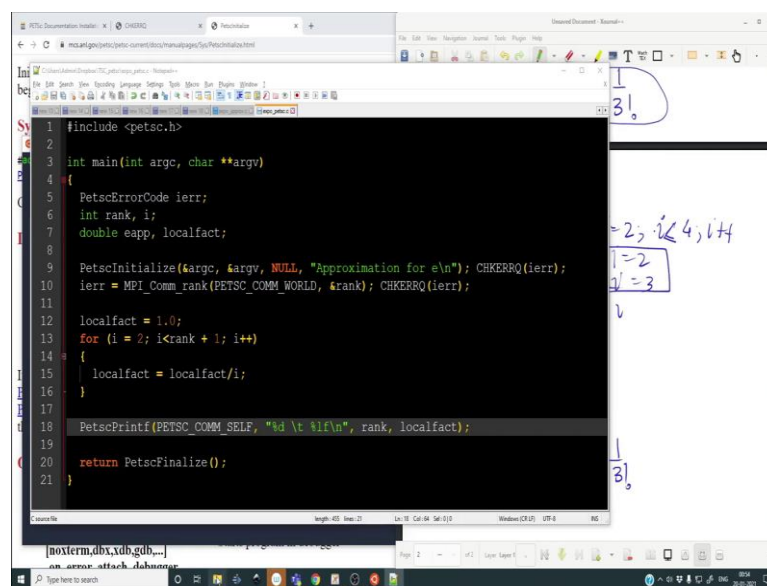
- A diagram showing ranks 0, 1, 2, 3 with arrows pointing to them.
- The text "Gather all data".
- A box containing  $rank_3 = 3$ .
- Handwritten notes:  $i=2; i \leq 4; i++$  and a box containing  $\frac{1}{2}$  and  $\frac{1}{3}$ .
- Calculations for  $if = 1.0$ :
  - $i=2 \quad if = \frac{1}{2}$
  - $i=3 \quad if = \frac{1}{2} \times \frac{1}{3} = \frac{1}{3}!$

So, then, so it enters the loop with  $i = 1$ , and `lf = 1`, local fact = 1; then  $i = 2$ , it tells local fact = local fact by  $i$ . So, local fact is 1,  $i$  is 2 let us the loop will execute from 2 to rank +

$i - 1$ . So,  $i = 2$   $i = \text{rank} + 1$ ,  $i$  less than 4,  $i + 4$ . So, it will execute a loop going from  $i = 2$ ,  $i = 3$ , that is it, because the loop will execute till  $i$  less than 4 that is still  $i = 3$ .

After this  $i = 3$ , the local factorial will become  $1f$  by  $i$ . So,  $1f$  is already  $1$  by  $2$ , so  $1$  by  $2$  divided by  $3$ . So, it is  $1$  by  $3$  factorial. So, this is how we can achieve a local factorial using example 1. And this all will execute for all the different parallel threads that you will create it is not running for a single processor. All this will execute for all the multiple processes that you will create.

(Refer Slide Time: 34:38)



```
1 #include <petsc.h>
2
3 int main(int argc, char **argv)
4 {
5     PetscErrorCode ierr;
6     int rank, i;
7     double eapp, localfact;
8
9     PetscInitialize(&argc, &argv, NULL, "Approximation for e\n"); CHKERRQ(ierr);
10    ierr = MPI_Comm_rank(PETSC_COMM_WORLD, &rank); CHKERRQ(ierr);
11
12    localfact = 1.0;
13    for (i = 2; i < rank + 1; i++)
14    {
15        localfact = localfact/i;
16    }
17
18    PetscPrintf(PETSC_COMM_SELF, "%d \t %f\n", rank, localfact);
19
20    return PetscFinalize();
21 }
```

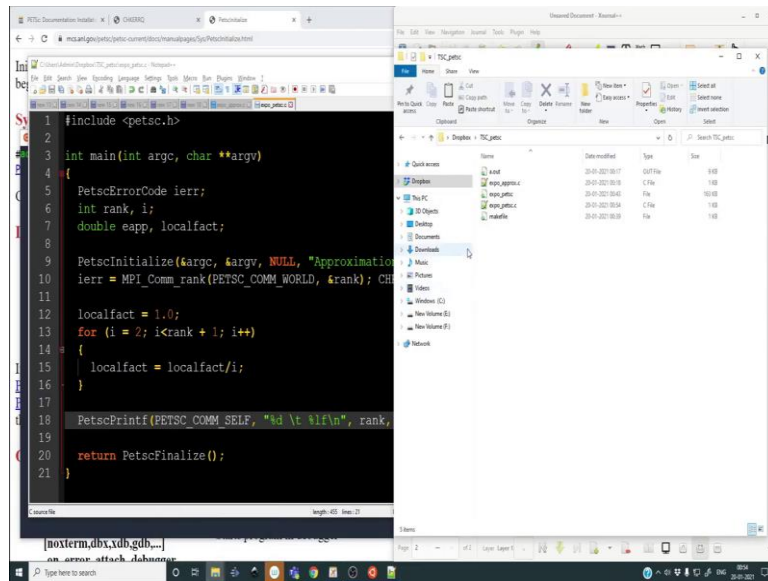
Handwritten notes on the right side of the code editor:

- $i = 2; i < 4; i++$
- $1 = 2$
- $1 = 3$
- $3!$

So, we have local fact. And so, what we can do now? We can print out what rank of the thread is and what the factorial it has written. Let us do that. We will use PETSc print f, PETSc comm self because for each thread that has to execute.

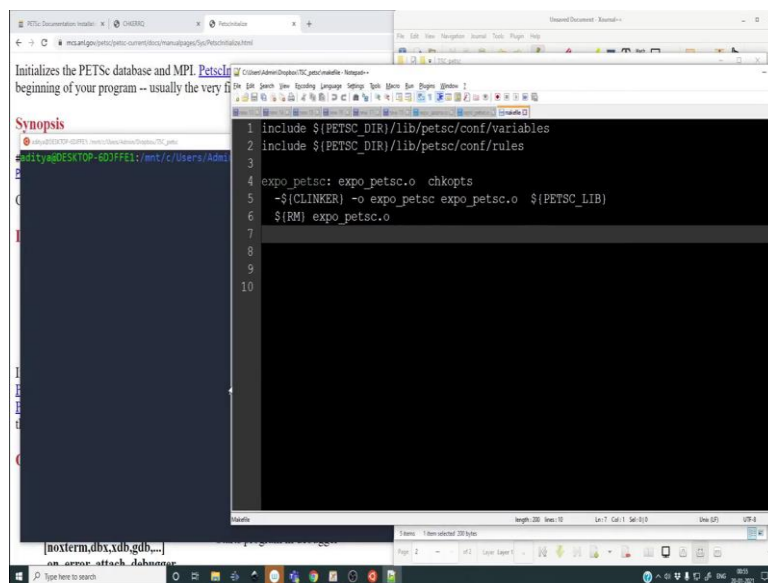
So, PETSc print f has to execute for each step, otherwise it will show some behavior which is not controllable. We need to pass this definite. Then, as usual  $\% d \backslash t$  percentage  $1f$  because we directly finding out  $1$  by factorial, alright. So,  $\backslash n$  then we will say rank and local fact, yeah. So, that is pretty much it.

(Refer Slide Time: 35:25)



```
1 #include <petsc.h>
2
3 int main(int argc, char **argv)
4 {
5     PetscErrorCode ierr;
6     int rank, i;
7     double eapp, localfact;
8
9     PetscInitialize(&argc, &argv, NULL, "Approximation
10     ierr = MPI_Comm_rank(PETSC_COMM_WORLD, &rank); CH
11
12     localfact = 1.0;
13     for (i = 2; i < rank + 1; i++)
14     {
15         localfact = localfact/i;
16     }
17
18     PetscPrintf(PETSC_COMM_SELF, "%d \t %f\n", rank,
19
20     return PetscFinalize();
21 }
```

(Refer Slide Time: 35:29)



```
1 include $(PETSC_DIR)/lib/petsc/conf/variables
2 include $(PETSC_DIR)/lib/petsc/conf/rules
3
4 expo_petsc: expo_petsc.o chkopts
5     -$(LINKER) -o expo_petsc expo_petsc.o $(PETSC_LIB)
6     $[RM] expo_petsc.o
7
8
9
10
```

Now, before compiling this, we need to create a make file. I have already created make file. (Refer Time: 35:28) with the notepad. The make file contains these lines, it will always contain these lines, it will contain the variables or rules, and the target will be first to create the object file and it will link the object file to the executable with the help of this then it will make use of the libraries of PETSc.

And finally, it will remove the object file we do not because we do not need the object file, we just care about the exponent exponential about the executive ok.



(Refer Slide Time: 37:01)

```
Initializes the PETSc database and
beginning of your program -- usual

Synopsis
1 #include <petsc.h>

In file included from /mnt/f/petsc/petsc-3.13.2/include/petscsys.h:1247:0,
             from /mnt/f/petsc/petsc-3.13.2/include/petsc.h:4,
             from /mnt/f/petsc/petsc-3.13.2/include/petsc.h:5,
             from /mnt/c/Users/Admin/Dropbox/TSC_petsc/expo_petsc.c:1:
/mnt/f/petsc/petsc-3.13.2/include/petscerror.h:463:51: warning: 'ierr' is used uninitialized in this function [-Wuninitialized]
#define CHKERRQ(ierr) do {PetscErrorCode ierr = (ierr); if (PetscUnlikely
(ierr)) return PetscError(PETSC_COMM_SELF, __LINE__, PETSC_FUNCTION_NAME, FILE, ierr
c__PETSC_ERROR_REPEAT, " ");} while (0)

/mnt/c/Users/Admin/Dropbox/TSC_petsc/expo_petsc.c:5:17: note: 'ierr' was declared here
     PetscErrorCode ierr;

Warning: chkopts target is deprecated and can be removed from user makefiles
/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -Wall -Wwrite-strings -Wno-str
ict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility-hidden -g3 -o expo
_petsc.o -Wl,-rpath,/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl,-rpath,/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl,-rpath,/usr/lib/x86_64-linux-gnu -L/usr/lib/x86_64-linux-gnu -Wl,-rpath,/lib/x86_64-linux-gnu -L/lib/x86_64-linux-gnu -L/usr/lib/x86_64-linux-gnu -Wl,-rpath,/lib/x86_64-linux-gnu -L/lib/x86_64-linux-gnu -lpetsc -lflpack -lfolas -lpthread -lX11 -lm -lstdc++ -ldl -lmpifort
-lmpi -lgfortran -lm -lgfortran -lm -lgcc_s -lquadmath -lstdc++ -ldl
/bin/rm -f expo_petsc.o
[mnterm.dbx.xdb.gob...]$ make expo_petsc.o
on_serve_attach_dabunotes

length 401 lines 2  Ln:3 Col:65 541:013  Windows (X11)  UTF-8
```

(Refer Slide Time: 37:09)

```
Initializes the PETSc database and
beginning of your program -- usual

Synopsis
1 #include <petsc.h>

/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -o expo_petsc.o -c -Wall -Wwri
te-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility-h
idden -g3 -I/mnt/f/petsc/petsc-3.13.2/include -I/mnt/f/petsc/petsc-3.13.2/arch-linu
x-c-debug/include -pwd /expo_petsc.c
/mnt/c/Users/Admin/Dropbox/TSC_petsc/expo_petsc.c: In function 'main':
/mnt/c/Users/Admin/Dropbox/TSC_petsc/expo_petsc.c:7:9: warning: unused variable 'eap
' [-Wunused-variable]
     double eap;
           ^

Warning: chkopts target is deprecated and can be removed from user makefiles
/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -Wall -Wwrite-strings -Wno-str
ict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility-hidden -g3 -o expo
_petsc.o -Wl,-rpath,/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl,-rpath,/usr/lib/x86_64-linux-gnu -L/usr/lib/x86_64-linux-gnu -Wl,-rpath,/lib/x86_64-linux-gnu -L/lib/x86_64-linux-gnu -L/usr/lib/x86_64-linux-gnu -lpetsc -lflpack -lfolas -lpthread -lX11 -lm -lstdc++ -ldl -lmpifort
-lmpi -lgfortran -lm -lgfortran -lm -lgcc_s -lquadmath -lstdc++ -ldl
/bin/rm -f expo_petsc.o
[mnterm.dbx.xdb.gob...]$ make expo_petsc
on_serve_attach_dabunotes

length 401 lines 2  Ln:3 Col:65 541:013  Windows (X11)  UTF-8
```



(Refer Slide Time: 37:10)

```
Initializes the PETSc database and
beginning of your program -- usual

Synopsis
1 #include <petsc.h>

/mnt/ff/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -o expo_petsc.o -c -Wall -Werror -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -g3 -I/mnt/ff/petsc/petsc-3.13.2/include -I/mnt/ff/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/mnt/ff/petsc/petsc-3.13.2/arch-linux-c-debug/lib -lmpi -lmpifort -lmpi -lgfortran -lm -lgcc_s -lquadmath -lstdc++ -ldl
/mnt/c/Users/Admin/Dropbox/TSC_petsc/expo_petsc.c: In function 'main':
/mnt/c/Users/Admin/Dropbox/TSC_petsc/expo_petsc.c:7:9: warning: unused variable 'eap'
1
1
double rank, localfact;
Warning: chkopts target is deprecated and can be removed from user makefiles
/mnt/ff/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -g3 -o expo_petsc expo_petsc.o -Wl,-rpath,/mnt/ff/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/mnt/ff/petsc/petsc-3.13.2/arch-linux-c-debug/lib -lmpi -lmpifort -lmpi -lgfortran -lm -lgcc_s -lquadmath -lstdc++ -ldl
/bin/rm -f expo_petsc.o
aditya@DESKTOP-60JFFE1:/mnt/c/Users/Admin/Dropbox/TSC_petsc:$ SPETSC_DIR=/SPETSC_ARCH/B
in/mplicxec -n 4 ./expo_petsc
0 1.000000
1 0.500000
2 1.000000
3 0.166667
aditya@DESKTOP-60JFFE1:/mnt/c/Users/Admin/Dropbox/TSC_petsc:$
[no term.abx.xdb.gob...]
```

This should work, great. So, it is just showing a warning because e approximate we have not yet called, but that is ok. So, now what we can do is we can use MPI to launch those many number of threads and find out the factorial for each thread.

So, the way to do it, I have shown this earlier as well, PETSc dir we have to use the appropriate MPI ch, PETSc arch MPI exac - n, let us say 4 threads and the executable is; so, approx.c PETSc so we have created 4 threads. So, 0 thread and 1 thread return 1 obviously, 2 returns half, returns 1 by 6.

(Refer Slide Time: 38:07)

```
Initializes the PETSc database and
beginning of your program -- usual

Synopsis
1 #include <petsc.h>

/mnt/ff/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -g3 -o expo_petsc expo_petsc.o -Wl,-rpath,/mnt/ff/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/mnt/ff/petsc/petsc-3.13.2/arch-linux-c-debug/lib -lmpi -lmpifort -lmpi -lgfortran -lm -lgcc_s -lquadmath -lstdc++ -ldl
/mnt/c/Users/Admin/Dropbox/TSC_petsc/expo_petsc.c: In function 'main':
/mnt/c/Users/Admin/Dropbox/TSC_petsc/expo_petsc.c:7:9: warning: unused variable 'eap'
1
1
double rank, localfact;
Warning: chkopts target is deprecated and can be removed from user makefiles
/mnt/ff/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -g3 -o expo_petsc expo_petsc.o -Wl,-rpath,/mnt/ff/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/mnt/ff/petsc/petsc-3.13.2/arch-linux-c-debug/lib -lmpi -lmpifort -lmpi -lgfortran -lm -lgcc_s -lquadmath -lstdc++ -ldl
/bin/rm -f expo_petsc.o
aditya@DESKTOP-60JFFE1:/mnt/c/Users/Admin/Dropbox/TSC_petsc:$ SPETSC_DIR=/SPETSC_ARCH/B
in/mplicxec -n 8 ./expo_petsc
0 1.000000
1 0.500000
2 1.000000
3 0.166667
4 1.000000
5 0.008333
6 0.001309
7 0.000198
aditya@DESKTOP-60JFFE1:/mnt/c/Users/Admin/Dropbox/TSC_petsc:$
[no term.abx.xdb.gob...]
```

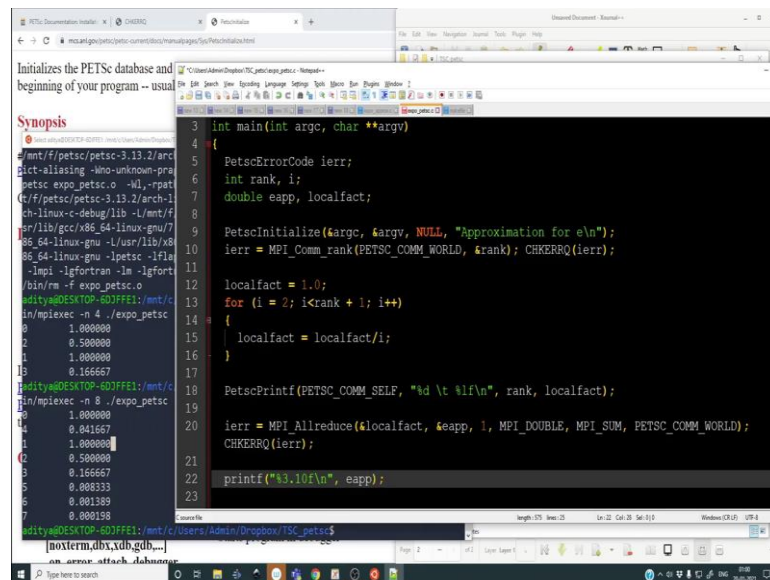


So, everything looks fine. You can create more threads despite my computer being having 4 cores, you can create 8 and create more than that as well, and you can verify all these are correct. You will notice that the ranks are not printed in sequence. And that is why that is because once you call a print f. So, all this is remember, after initializing or declaring the number of threads all this runs in parallel for all threads.

So, whenever a certain thread is able to do this computation first I will go to this print , it will execute this print state. For threads which will take more time it will appear after a while, ok. There is no synchronization among the thread. And that is something which parallel programming you have to always account ok.

There is no synchronization. Once you start doing these kind of things you will quickly realize you have to do a lot of hard work to synchronize all these things. So, you have to do lot of thinking, deep learning and all, but anyway. So, for now we have all the different threads giving us all the different factorials. So, now, we must accumulate all the different factorials.

(Refer Slide Time: 39:23)



```
3 int main(int argc, char **argv)
4 {
5     PetscErrorCode ierr;
6     int rank, i;
7     double eapp, localfact;
8
9     PetscInitialize(&argc, &argv, NULL, "Approximation for e\n");
10    ierr = MPI_Comm_rank(PETSC_COMM_WORLD, &rank); CHKERRQ(ierr);
11
12    localfact = 1.0;
13    for (i = 2; i <= rank + 1; i++)
14    {
15        localfact = localfact * i;
16    }
17
18    PetscPrintf(PETSC_COMM_SELF, "%d\t %lf\n", rank, localfact);
19
20    ierr = MPI_Allreduce(&localfact, &eapp, 1, MPI_DOUBLE, MPI_SUM, PETSC_COMM_WORLD);
21    CHKERRQ(ierr);
22    printf("%3.10f\n", eapp);
23
```

So, the way to do that in parallel environments to reduce all the threads to a single thread, `ierr = MPI all reduce`. So, now, to all reduce function we must pass the local facts, the addresses of all the local facts, it is just a local fact. We must pass the address of the variable where we want to store all the reduction variables. So, after reducing all the local factorials, we want only one sum and that is `e app`. So, we must pass the address

of the approximate, then we must pass how many what we call what each local factor, how many elements each local factor that is 1, then MPI double that is the data type, this is the data type of; ok it has to be in capitals.

Finally, what kind of reduction we must do, there are various kind of reduction, there are products that are sums we must say that you must sum, finally, we must pass comm world, alright. Once we have done this, we can check for error, alright. So, with the help of this function, function call, it will take all the local factorials, do a sum and put it into the app, alright. So, finally, we can check with print, yeah. So, once we gone obtained a single thread, because we want to do print f percentage 3.10 f \n PETSc.

(Refer Slide Time: 41:13)

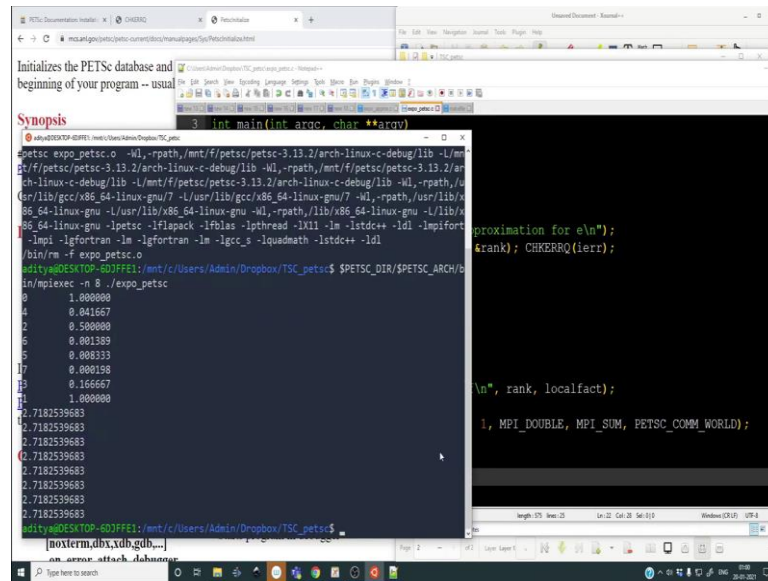
```
Initializes the PETSc database and beginning of your program -- usual
```

```
Synopsis
3 int main(int argc, char **argv)
```

```
aditya@DESKTOP-6D3FFE1:~/mnt/c/Users/Adain/Dropbox/TSC_petsc$ ./expo_petsc
0 1.000000
1 0.041667
2 1.000000
3 0.000000
4 0.166667
5 0.008333
6 0.001389
7 0.000198
```

```
aditya@DESKTOP-6D3FFE1:~/mnt/c/Users/Adain/Dropbox/TSC_petsc$ make expo_petsc
/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -o expo_petsc.o -c -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -g3 -I/mnt/f/petsc/petsc-3.13.2/include -I/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/include -Dmpiexec -Dmpifile
Warning: mpiexec target is deprecated and can be removed from user makefiles
/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/bin/mpicc -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -g3 -o expo_petsc expo_petsc.o -Wl,-rpath,/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl,-rpath,/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -L/mnt/f/petsc/petsc-3.13.2/arch-linux-c-debug/lib -Wl,-rpath,/usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/gcc/x86_64-linux-gnu/7 -Wl,-rpath,/usr/lib/x86_64-linux-gnu -L/usr/lib/x86_64-linux-gnu -Wl,-rpath,/lib/x86_64-linux-gnu -L/lib/x86_64-linux-gnu -lpetsc -lfblas -lfblas -lflapack -lflblas -lflapack -lX11 -lm -lstdc++ -ldl -lmpifort -lmpi -lf90fortran -lm -lgfortran -lm -lgcc_s -lquadmath -lstdc++ -ldl
/bin/rm -f expo_petsc.o
aditya@DESKTOP-6D3FFE1:~/mnt/c/Users/Adain/Dropbox/TSC_petsc$ make expo_petsc
/bin/rm -f expo_petsc.o
```

(Refer Slide Time: 41:17)



```
3 int main(int argc, char **argv)
{
    PetscErrorCode ierr;
    int rank, i;
    double eapp, localfact;

    PetscInitialize(&argc, &argv, NULL, "Approximation for e\n");
    ierr = MPI_Comm_rank(PETSC_COMM_WORLD, &rank); CHKERRQ(ierr);

    localfact = 1.0;
    for (i = 2; i < rank + 1; i++)
    {
        localfact = localfact/i;
    }

    PetscPrintf(PETSC_COMM_SELF, "%d \t %lf\n", rank, localfact);

    ierr = MPI_Allreduce(&localfact, &eapp, 1, MPI_DOUBLE, MPI_SUM, PETSC_COMM_WORLD);
    CHKERRQ(ierr);

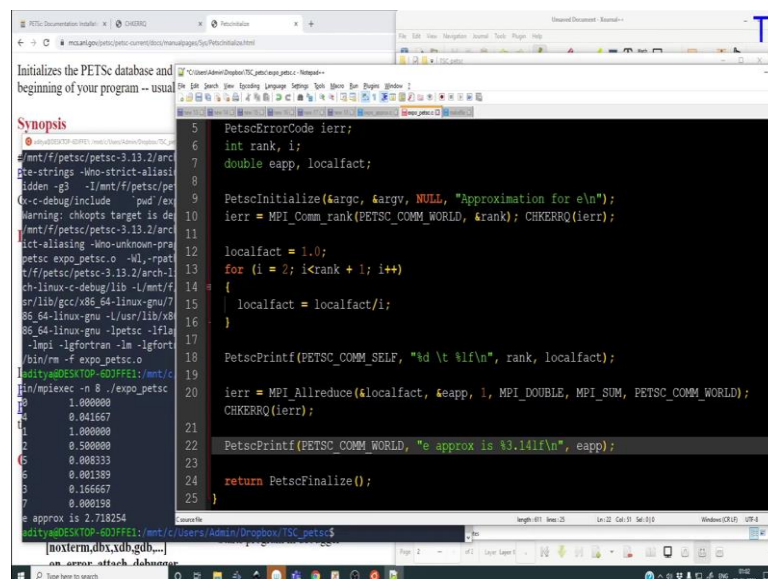
    PetscPrintf(PETSC_COMM_WORLD, "e approx is %3.14lf\n", eapp);

    return PetscFinalize();
}
```

```
1.000000
0 0.041667
2 0.500000
4 0.001389
6 0.000333
8 0.000159
10 0.166667
11 1.000000
2.7182539683
2.7182539683
2.7182539683
2.7182539683
2.7182539683
2.7182539683
2.7182539683
2.7182539683
e approx is 2.718254
```

Ok we will print it again, so ok. So, it is outputting the final print statement a bunch of times you must do something about it. What can we do about? So, what has happened is this has it has executed for all the number of threads that we instead of this we will use PETSc print f, but we will use comm world. So, comm self is using all the different threads. So, self implies all the different threads that you have done.

(Refer Slide Time: 42:02)



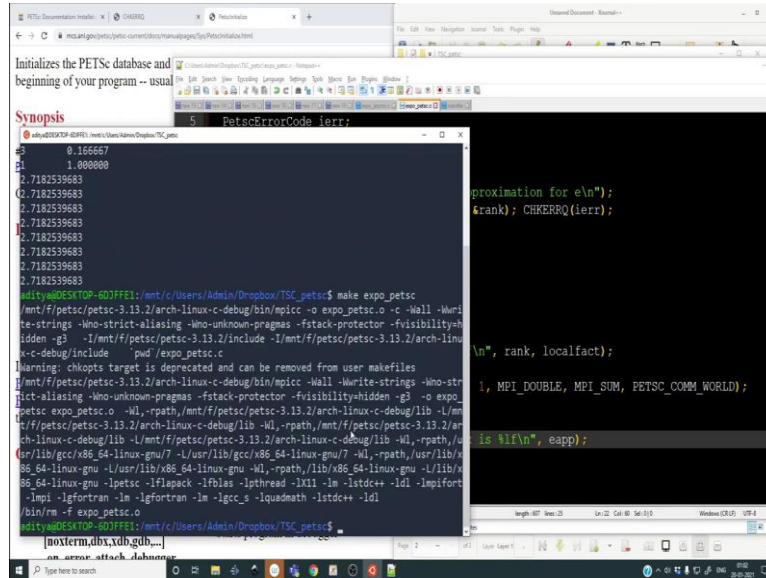
```
5 PetscErrorCode ierr;
6 int rank, i;
7 double eapp, localfact;
8
9 PetscInitialize(&argc, &argv, NULL, "Approximation for e\n");
10 ierr = MPI_Comm_rank(PETSC_COMM_WORLD, &rank); CHKERRQ(ierr);
11
12 localfact = 1.0;
13 for (i = 2; i < rank + 1; i++)
14 {
15     localfact = localfact/i;
16 }
17
18 PetscPrintf(PETSC_COMM_SELF, "%d \t %lf\n", rank, localfact);
19
20 ierr = MPI_Allreduce(&localfact, &eapp, 1, MPI_DOUBLE, MPI_SUM, PETSC_COMM_WORLD);
21 CHKERRQ(ierr);
22
23 PetscPrintf(PETSC_COMM_WORLD, "e approx is %3.14lf\n", eapp);
24
25 return PetscFinalize();
}
```

```
1.000000
0 0.041667
2 0.500000
4 0.001389
6 0.000333
8 0.000159
10 0.166667
11 1.000198
e approx is 2.718254
e approx is 3.14159
```

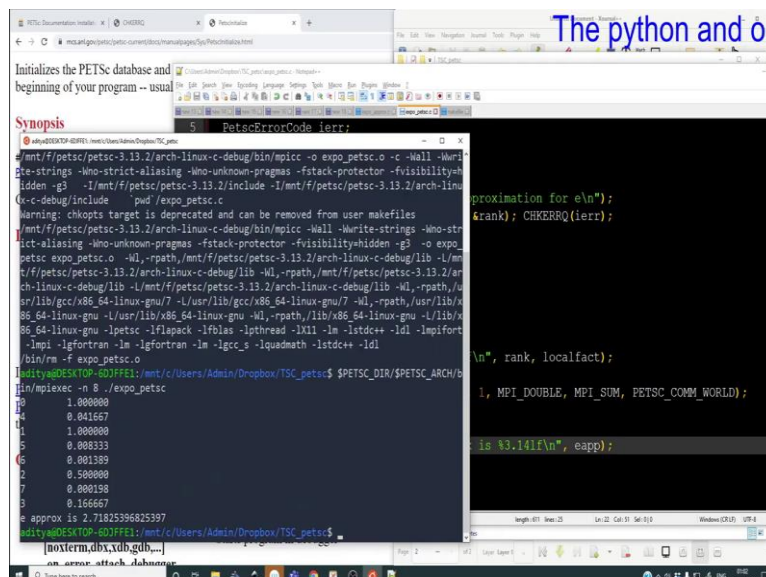
But we will do this, PETSc print f PETSc comm world. So, once we use world it will use the merge thread to find to only print it 1, and using this we going to say e approx is

percentage of  $\frac{1}{n}$  and we are going to print out (Refer Time: 42:38). So, PETSc print f by passing comm world we are going to print out only one the output for only one thread, alright.

(Refer Slide Time: 42:53)

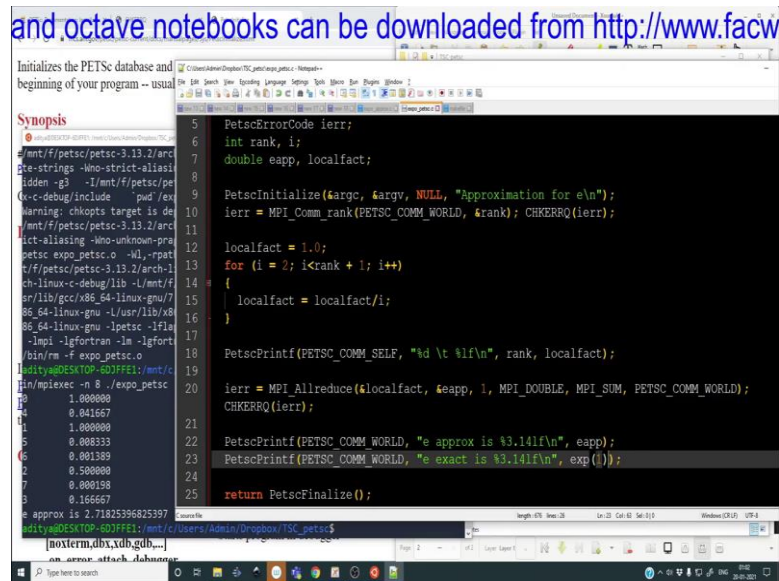


(Refer Slide Time: 43:06)



(Refer Slide Time: 43:10)

and octave notebooks can be downloaded from <http://www.facw>



```
Initializes the PETSc database and beginning of your program -- usual

Synopsis
5 PetscErrorCode ierr;
6 int rank, i;
7 double eapp, localfact;
8
9 PetscInitialize(&argc, &argv, NULL, "Approximation for e\n");
10 ierr = MPI_Comm_rank(PETSC_COMM_WORLD, &rank); CHKERRQ(ierr);
11
12 localfact = 1.0;
13 for (i = 2; i < rank + 1; i++)
14 {
15     localfact = localfact/i;
16 }
17
18 PetscPrintf(PETSC_COMM_SELF, "%d %t %f\n", rank, localfact);
19
20 ierr = MPI_Allreduce(&localfact, &eapp, 1, MPI_DOUBLE, MPI_SUM, PETSC_COMM_WORLD);
21 CHKERRQ(ierr);
22 PetscPrintf(PETSC_COMM_WORLD, "e approx is %3.14f\n", eapp);
23 PetscPrintf(PETSC_COMM_WORLD, "e exact is %3.14f\n", exp(1));
24
25 return PetscFinalize();

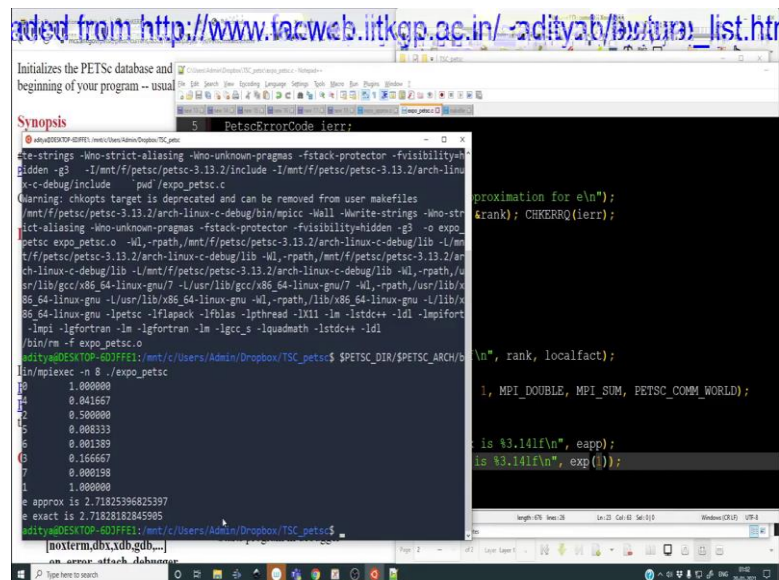
e approx is 2.71828182845905
e exact is 2.71828182845905

length: 678, lines: 38, Ln: 23, Col: 65, Sel: 0:0, Windows (X11): 0/4
```

Let us see ok we can get out of this 2.1 let me (Refer Time: 42:59) more decimals, ok. Let us see what the, let us copy this, e exact let us print out e exact, right.

(Refer Slide Time: 43:30)

added from [http://www.facweb.iitkgp.ac.in/~aditya/bw/turn\\_list.htm](http://www.facweb.iitkgp.ac.in/~aditya/bw/turn_list.htm)



```
Initializes the PETSc database and beginning of your program -- usual

Synopsis
5 PetscErrorCode ierr;
6 int rank, i;
7 double eapp, localfact;
8
9 PetscInitialize(&argc, &argv, NULL, "Approximation for e\n");
10 ierr = MPI_Comm_rank(PETSC_COMM_WORLD, &rank); CHKERRQ(ierr);
11
12 localfact = 1.0;
13 for (i = 2; i < rank + 1; i++)
14 {
15     localfact = localfact/i;
16 }
17
18 PetscPrintf(PETSC_COMM_SELF, "%d %t %f\n", rank, localfact);
19
20 ierr = MPI_Allreduce(&localfact, &eapp, 1, MPI_DOUBLE, MPI_SUM, PETSC_COMM_WORLD);
21 CHKERRQ(ierr);
22 PetscPrintf(PETSC_COMM_WORLD, "e approx is %3.14f\n", eapp);
23 PetscPrintf(PETSC_COMM_WORLD, "e exact is %3.14f\n", exp(1));
24
25 return PetscFinalize();

e approx is 2.71828182845905
e exact is 2.71828182845905

length: 678, lines: 38, Ln: 23, Col: 65, Sel: 0:0, Windows (X11): 0/4
```

So, if you use more number of threads, obviously the error will reduce, but, yeah. So, through this small program I have shown you how to sort of use some simple aspects of MPI and in case you are not so interested in all this do not worry because once we start using PETSc, you will quickly see that even running programs using a single thread they are very fast and very efficient.



So, in the later lectures, maybe we will just focus on one thread. But if you are interested in speed then always do this MPI exactly and allocate a large number of threads. But you know later on because you will be using a bunch of inbuilt solvers, you can pass only one thread you can get the solution, no problem. In the later programs, we will not be using this kind of reductions and all this. Typically, we will avoid doing all these reductions.

So, whatever MPI business needs to be done will be handled by PETSc in an inbuilt fashion, we will not be exposed to all this kind of declarations, then merging and all. So, but anyway this lecture, I hope to have shown you how to include PETSc. This includes all the libraries, inside this header file.

I have shown you how to initialize, how to declare error codes, how to print using different threads, how to reduce, and how to make a make file. So, make file always will look something like this; and how to execute; how to compile it using a make file, how to execute a file.

So, it is a very sort of simple lecture. It may be confusing in the beginning, but once you start using it, once you get the hang of what MPIs, this particular program will seem very easy. Even if you have not understood all of this completely; in the next class we will not be exposing ourselves to all this thread, splitting, thread merging and all.

Well, but all this will still exist PETSc comm world and all, because if you do use MPI, PETSc will internally allocate to the solver whatever number of threads it has to be, ok. So, with this I end this particular lecture. And I will see you next time with a new lecture and we will actually solve a one-dimensional problem using PETSc. Bye.