1 Introduction

Course content

- Introduction to HPSC and scientific computing
- Processor performance, memory hierarchy, multi-core computing and vector computing
- Introduction to parallel programming concepts and parallel algorithms
- Effective use of Bash scripting
- Effective use of tools like git, SVN, Mercurial etc
- OpenMP, MPI, GPGPU, Vector programming
- debuggers
- Performance analysis
- Numerical methods, applications of

This course is all about making you a better researcher. It is very hands on. The processors keep changing every two years. How to exploit hardware-level optimization techniques will also change every two years. This course is targeted at those who are pursuing a project - help them improve their code over there. So, include this in your SoPs and course description.

The biggest component of this course the project. All the best. This project is a distillation of the professor's experience over B.Tech. This man wants to *teach*. So you better *imbibe*.

The traditional scientific method paradigms have been about observation $\rightarrow$ hypothesis $\rightarrow$ testing by experimentation and going back to refine the hypothesis. Think Archimedes' eureka moment and the fable of an apple falling on Newton's head.

A more concrete example would be the Higgs Boson discovery; the existence of the Higgs Boson was hypothesized much earlier. The experiments at LHC set out to test this hypothesis and the veracity of the standard model in general.

The computational modelling paradigm is much recent compared to the traditional paradigms. It picked up during the second world war, the Los Alamos project, for example

Alexei Krylov - general in the imperial soviet army (remember Krylov subspaces and all that), his work led to important iterative methods in linear algebra (typically systems with millions to billions of students). Professor gave an example of computational fluid dynamics (CFD). I imagine that a lot of examples will be from mechanical engineering.

Consider trying to solve for $\vec{x}$ in a linear system of size n

$$A\vec{x} = B. \tag{1}$$

If you do Gaussian elimination, you'll have an algo that goes as $\mathcal{O}(n^3)$. Imagine if $n = 1M$. Unscalable. Practical applications using this method would be intractable even on the largest supercomputers. Using iterative solvers would reduce the complexity to something like $\mathcal{O}(n^2)$. If you're doing any serious computational work, you will be using these methods instead of naïve Gaussian elimination kind of methods.

Floating Point Operations (FLOPs)

A FLOP is a binary operation between two floating point operands. ENIAC was able to achieve 300 FLOPs per second (FLOPS, note the capitalisation). The ENIAC was used for writing artillery firing tables. Some factors that would come in

- Elevation of target

- Horizontal distance

- Aerodynamics of the shell

    - wind direction

    - geometry and weight of the shell

    - density of air

    - altitude

- . . .

ENIAC was insanely slow by modern computing standards. Head over to `https://top500.org`. (AMD would have us believe that exascale begins with their `Instinct MI100` GPUs). We are on the cusp of the first exascale machine. Thanks to Alexei Krylov's work.

The fastest supercomputer (SC) is Fugaku at RIKEN (Kobe, Japan). It clocks a peak 537 Peta FLOPS, and consumes about 30 MegaWatts (as much as a small town/city). IITB's SC, SpaceTime2, costs a few crores in electricity alone. Fugaku is about 4 times more power efficient than SpaceTime2. SpaceTime2 is also some four years old, and newer machines are becoming more and more power efficient.

Application Areas

- CFD - sneeze and cough droplet simulations in these trying times

- Structural Mechanics - stresses, strains, deformations, fractures and all that

- Computational Electromagnetics - scattering of EM waves off e.g. an aircraft body (RADAR signature detection for military aircraft). Stealth

- Magnetohydrodynamics - e.g. molten ferromagnetic material flowing. Nuclear fusion reactors use a blanket of Lithium-Lead liquid

- Computational astrophysics - mostly exclusively computational, i.e. no experimental counterpart. You cannot really sit around to observe the evolution of a galaxy over a gajillion years

- Computational Finance - models made using partial differential equations (PDEs), usually involve all time scales (microtrading to long term investments et cetera)

- Molecular Dynamics (MD)

Computational science cannot completely replace experiments, of course. You can explore the parameter space freely in computational models, and then cherry pick promising results to be tested by experimental rigs. "Computational science is the partial truth fully revealed. Experimental science is the full truth partially revealed". When you make a computational model, you have to start off with some simplifications.

What makes a good computational scientist

- Domain specific knowledge - obviously

- Programming proficiency - you should be able to write your models and math in code. You need to be able to use these large machines *efficiently*.

- Excellent background in applied mathematics and numerical analysis

Your model on a machine should be able to procure a three-day prediction in a few hours. It should also not run at some 30 megawatts just for a three-day prediction for one city. We will use parallel computing and serial computing optimization techniques.

Related to course logistics - the domain specific knowledge should come from your end. This course is targeted at senior undergrads and postgrads. All the best.

Standard benchmarking programs do exist for supercomputers. E.g. LIN-PACK.