Scientific Examples on a Single Core

Outline

Scientific example #1: *Drug design*

Scientific example #2: *Pandemic*

Scientific Example #1
Drug Design

Drug Design

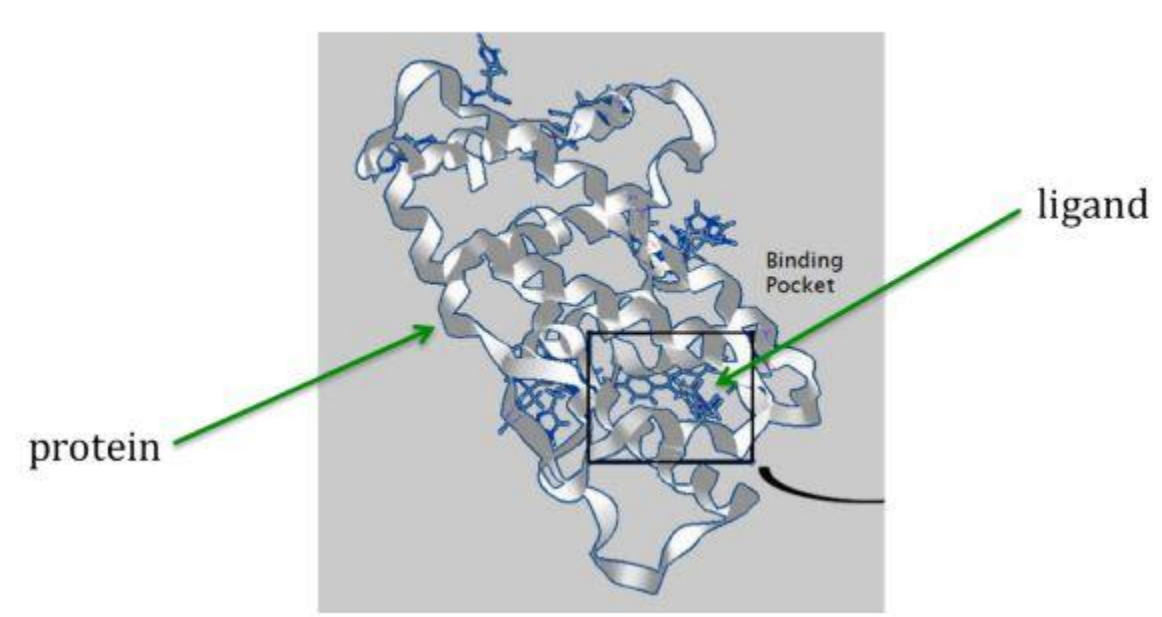
An extremely important problem in the biological sciences is **drug design**

Goal: Find small molecules (**ligands**) that are good candidates for use as drugs

General approach to the problem:

- A protein associated with the disease of interest is identified, and its 3D structure is found either *experimentally* or *through a molecular modeling computation*
- A **collection of ligands** is tested against the protein (that is, for every orientation of the ligand relative to the protein, computation is done to test whether the ligand binds with the protein in useful ways)
- A **score** is set based on these binding properties, and the best scores are flagged, identifying ligands that would make good drug candidates

Drug Design



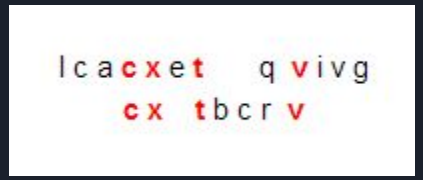
Drug Design

Working with actual ligand and protein data is beyond the scope of this example, so we will represent the computation by a simpler *string-based comparison*

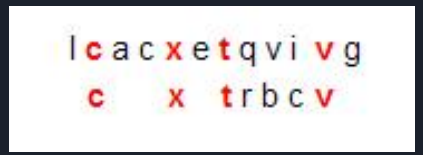
- Proteins and ligands will be represented as randomly-generated characters strings
- The computation will be represented by comparing a **ligand string (L)** to a **protein string (P)**
- The score for a pair [L, P] will be the maximum number of matching characters among all possibilities when L is compared to P, moving from left to right, allowing possible insertions and deletions

Drug Design

For example, if **L** is the string "`cxtbcrv`" and **P** is the string "`lcacxetqvivg`", then the score is **4**, arising from this comparison of **L** to a segment **P**:



Another possible match would be the following (also yielding a score of **4**):



NOTE: there is *no comparison* that matches *5 or more characters* while moving left to right, so the score is **4**

Drug Design

The point of this module is *not* to write the code yourself, but to use a working example to explore the problem and the performance of the solution

Included with this module is the file `dd_serial.cpp`, a simple sequential (serial) solution to this problem

You can compile this **C++** file as follows:

```
g++ -o dd_serial dd_serial.cpp
```

You can then run the program as follows:

```
./dd_serial
```

Drug Design

The `dd_serial` program optionally accepts up to *three command-line arguments*:

1) maximum length of the randomly generated ligand strings

2) number of ligands generated

3) protein string to which ligands will be compared

Try opening the source code file to see what the default values are for these!

Scientific Example #2

Pandemic

Pandemic

Another extremely important problem is **epidemiology** (the study of infectious disease)

Diseases are said to be ***contagious*** among people if they are transmittable from one person to another

Epidemiologists can use models to assist them in predicting the behavior of infectious diseases

We now look at a program that implements a simple agent-based model toward the study of infectious disease

Pandemic

Our model makes certain assumptions about the spread of the disease:

1) The disease spreads from person to person with some ***contagiousness factor*** (percent chance that the disease will be transmitted)
2) Diseases can only be spread from an ***infected*** person (carrying the disease) to a ***susceptible*** person (able to be infected)
3) The disease has an incubation period (***duration***) in which the disease remains in the person
4) The disease is only transmittable within a distance (***infection radius***) from an infected person
5) Each person ***moves randomly at most 1 unit*** in a given direction each day
6) After the incubation period, the person is either ***immune*** to the disease (moves around but incapable of being further infected or of infecting other people), or ***dead*** (incapable of being further infected, infecting other people, or moving)

Pandemic

The components of the model:

- Environment
  - The bound in which people move (a fixed ***width*** and ***height***)
- People
  - Each person has ***location (x,y)*** in the environment
  - Each person has one ***state***: [*susceptible*, *infected*, *immune*, *dead*]
- Disease
  - duration = number of days a person is infected
  - contagiousness factor = likelihood of the disease spreading
  - deadliness factor = likelihood that a person will die from the disease
- Timer
  - Counts the number of days that have elapsed in the simulation

Pandemic

Again, the point of this module is ***not*** to write the code yourself, but to use a working example to explore the problem and the performance of the solution

Included with this module is the directory `pandemic`, which contains a simple sequential (serial) solution to this problem with visualization

You can easily compile this **C** program by running:

```
make clean
make
```

You can then run the program as follows:

```
./pandemic_serial
```

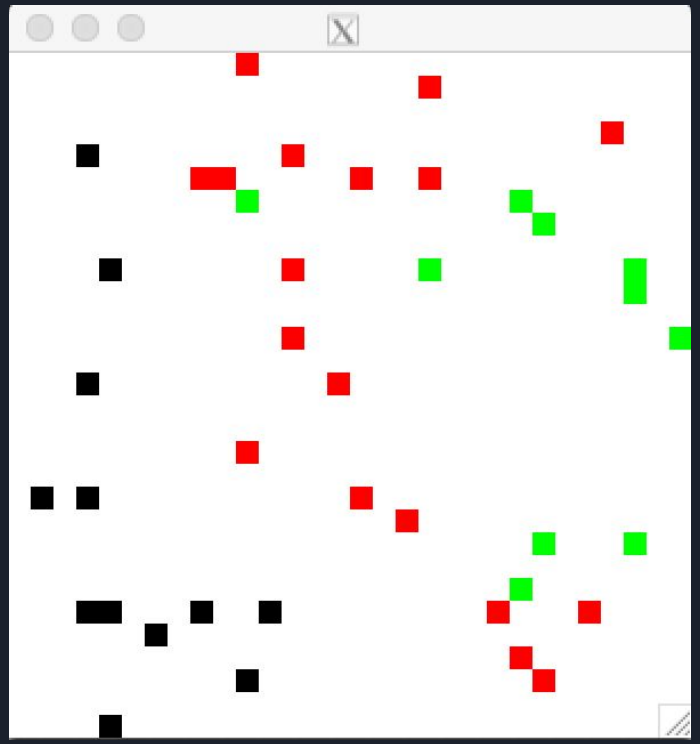
Pandemic

The `pandemic_serial` program optionally accepts the following *command-line arguments*:

- `-n` the number of people in the model
- `-i` the number of initially infected people
- `-w` the width of the environment
- `-h` the height of the environment
- `-t` the number of time days in the model
- `-T` the duration of the disease (in days)
- `-c` the contagiousness factor of the disease
- `-d` the infection radius of the disease
- `-D` the deadliness factor of the disease
- `-m` the number of microseconds in between days in the model (used to speed up or slow down the animation)

Try opening the source code files to see what the default values are for these!

Pandemic



What is the problem with these?

Both programs work fine, but if you start playing around with the parameters for the models, you will quickly see the performance take a major hit

For both problems, think about:

Why do they slow down based on different parameter values?

How much do they slow down?

Would a faster processor help?

Are we able to run parts of the problem in parallel if we had more than one core?

How would you go about parallelizing these programs? No need for specifics, as you will learn how to later.