CSE 6220 Introduction to High Performance Computing
Spring 2015
Programming Assignment 1
Due Tuesday February 24

In *genetics*, a sequence motif is a pattern that occurs in multiple sequences. Motifs are associated with important biological functions and finding "approximate" motifs is an important problem in Computer Science. In this assignment, you will implement one version of motif finding problem across various sequences.

Consider a set of n l-bit integers ($l \leq 64$): $\{S_1, S_2 \dots S_n\}$. Given a number d ($d \leq l$), the goal is to find all integers which are at a Hamming distance at most d from all of the given n integers. Hamming distance between two sequences (in our case, bit representation of integers) of equal lengths is the number of corresponding positions where the sequences differ.

A brute force approach for the problem would be to enumerate all possible 2^l integers and for each of these, check if it is within Hamming distance d from each of the n integers in the input set. For $l = 40$, this search space consists of more than 1 trillion entries, making it computationally infeasible for larger values of l.

Instead consider the set of all integers which are within a Hamming distance d from S_1. The total number of such integers is $\binom{l}{d} \times 2^d$. Note that the solution set is a subset of this set. Thus we can search through this set for the solution, significantly reducing our search space. This search space can be enumerated starting with bit representation of S_1 and inverting bits in selected positions. Specifically, we traverse this l-bit vector from left to right. Suppose we are at position i ($0 \leq i < l - 1$) and we have inverted j positions so far. If $j < d$, we have two possibilities for position $i + 1$, either to invert or keep the bit the same. We consider both possibilities and move forward. If $j = d$, we don't consider the possibility of inverting the bit. Once $i = l - 1$, we check the Hamming distance of the resulting number from each element in the set $\{S_2 \dots S_n\}$. If all the distances are less than or equal to d, the number goes into the solution set. One can come up with more sophisticated strategies to further reduce the search space and you are welcome (but not required) to do so.

The objective of this assignment is to write a parallel program to solve the motif finding problem. Your solution should follow the following outline: We use the master-worker paradigm, where processor with rank 0 will act as the master processor and the rest of the processors act as workers. To begin with, each processor will have a copy of the set of input integers and the parameter d. The master will initiate the search by starting to fill the vector of bits as described before. However, whenever a specific position k is reached, a copy of the vector will be dispatched to one of the worker processors. The worker processor will calculate the number of inversions done by comparing the received sequence with S_1. It will then explore the remaining search space with consideration to the number of further inversions allowed, and report any solution(s) found to the master processor. As soon as the task is dispatched, the master processor will continue by exploring the search space up to position k as described, to generate another task and dispatch it.

One can view the master processor as performing a search limited to the first k positions in the bit vector. Each worker processor will get a task with the first k positions filled. The worker will perform a search for the remaining $l - k$ positions. When a worker is done with its task, it will report the outcome and request for more work. The master processor will continually dispatch work until all the tasks are over. At this stage, it will send a message to all workers to terminate and terminates itself.

In addition, you should overlap computation and communication for best performance. To achieve this, each worker processor starts off with two tasks in hand. It starts processing one and keeps the second one in a buffer. On completion of the task, while the worker reports the solution(s) to the master and waits for next task to be assigned, it starts working on the buffered task in parallel. When the next task arrives for the worker, it goes into the buffer. The worker continues in this fashion for all further tasks, resulting in significant reduction of waiting time by workers.

Submission guidelines

The assignment is due on **Feburary 24, 11:55 PM EST** on T-Square. A framework is provided along with the assignment for you to work with. Please refer to the README file provided in the framework for information on how to use it. You are expected to work in teams of two. One member from each team should submit a zip/tar file containing the following

1. A text file containing the names of all team members and their contribution in terms of percentage of work done.
2. All source files. Your program should be well commented and easy to read.
3. A report in pdf format containing the following:
    - Short design description of your algorithms.
    - Explain the communication protocols, and state how overlapping of computation and communication is achieved.
    - Runtime and speedup plots by varying problem size, parameter k and number of processors (varying one while keeping others fixed). Also provide the input file corresponding to each plot. Give observations on how your algorithm behaves on varying these parameters.
4. Include the graphs generated and any observations that you can make about the performance. Play with the value of k and report the graphs and corresponding value of k that give the highest performance. Report any ideas or optimizations of your own that you might have applied and how that might have affected the performance.

Input and output format

The input to the program is one text file. The first three lines contain the parameters n, l and d in that order. They are followed by n l-bit integers in decimal format. Execution of your program should produce a text file sorted in ascending order, containing m integers, with each number in a separate line, m being the number of solutions. Note that this is already implemented in the framework.