2nd Sem. 2017-18

CS F422 Parallel Computing

Assignment 1

==

- Mode:
  - Take-home, Programming, In Teams of two or three students
- Learning Outcome:
  - The student should be able to
    - i. design distributed programs of reasonable complexity,
    - ii. implement the same using C/C++ and MPI,
    - iii. deploy/run them on a (Beowulf-style) cluster, and
    - iv. measure their performance and predict the scalability
- Marks: 15% (of the total weight of the course)
- Due Date: Wed. 21st Feb. (11pm).
- Deliverables and Evaluation:
  - Design and brief Analysis (via a design document)
  - Code and
  - Performance Results (Actual Measurements of Time, Plots demonstrating scalability)

==

General Notes

- Suggested Model of Development:
  - Have a clear design and analyze the expected performance and scalability on paper before coding.
  - Code using C/C++ and MPI calls. Use OpenMPI as the implementation choice for MPI.
  - Insall OpenMPI on an individual desktop/laptop and use it for development.
  - Once you have a reasonably stable implementation, deploy it on a cluster (in CSIS lab) and test it, measure the performance, and tune the performance (which may require significant change in code, and possibly a little in the design as well).
  - Be prepared for at least two to four iterations of this cycle i.e. last minute efforts are doomed to fail!
- Design and Analysis:
  - You may use an existing / known sequential algorithm to begin with. Your design focus should be on parallelizing the algorithm.
  - Your analysis should focus on minimizing communication complexity (i.e. number and size of messages exchanged) and synchronization cost (i.e. number of synchronization points and potential delays caused).
- Performance Results:
  - This is a critical component of the exercises (and will carry a significant weight in evaluation). Demonstrating scalability is a must!

    - This in turn requires testing on multiple inputs and varying sizes as applicable. For each input / size, measure the time taken on multiple runs to average out variations and eliminate anomalous readings.
- Evaluation:
    - Evaluation will be based on the design, code, and performance results.
    - A demo may be required to complete the evaluation. If so, it will be scheduled after the due date.
    - A team may consist of two students or three students. No more than three students will be permitted in a team. Singleton teams are discouraged.
    - Three-person teams will be assigned additional work marked below.

- Fairness Policy:
    - Each team is expected to work independently. If at all teams collaborate they are expected to report such collaboration specifying the form and details of collaboration at the time of uploading the assignment.
    - Any use of unfair means including copying from other sources such as the Web will warrant a sanction that is equal to or more than the weight of the assignment. Further you may be reported to the UnFair Means Committee of the Institute and be subject to regulations stipulating penalty for such cases.

==

Exercises

Exercise 1.

a) Design a parallel version of the Sieve of Eratosthenes algorithm given below to find all primes less than N (an input integer):

   i. Create the list Nums of numbers from 2 to N;
   ii. Initially no number is crossed out.
   iii. Let pr=2;
   iv. Scan Nums from the left and cross out multiples of pr (except pr itself).
   v. Update pr to be the next number not crossed out (to the right of current pr).
   vi. Repeat steps iv) and v) until pr > sqrt(N)

b) Implement this parallel algorithm using MPI in C. Note that your program should be parameterized by N, the input, and p, the number of nodes available.
c) Measure the performance for different values of N in the range 10^5 to 10^10. For each value of N, measure the performance for p = 1, 2, 4, and 8 where p is the number of (physical) nodes used and plot a curve indicating how performance (of a single input) varies as p increases.

Exercise 2. Consider a large collection of documents distributed in the secondary memories (i.e. hard disks) of multiple nodes in a cluster:

a) Design a program that extracts words and their frequency of occurrence from each document and create a word-document index in each node ranked on the frequency: i.e. for each word, a list of documents is associated with it and the list is ordered by decreasing frequency (of occurrence of that word in the document). Each index is local to a node a word is associated with only those documents in that node. Stop words i.e. frequently occurring words (e.g. a, an, the, for, if, to, then, on, etc.) need not be indexed. Assume that a list of stop words is available. Implement this program using MPI in C.
b) Design a program to merge all indices into one large word-document index ranked by decreasing frequency to be stored in node. Implement this program using MPI in C.
c) Measure the time taken for a) and b) for different numbers and sizes of documents and independently varying the number of processors used. For each input (of a given number of documents of certain sizes), plot a curve of time taken against number of processors used.

Exercise 3. [*Required only for teams of size three*] Provide an alternative extension to 2.a) - instead of the single large word-document index - where queries can be sent to any one of the nodes in which documents are stored and the responses are aggregated on demand into one result:

a) A *query* will constitute a word and a *response* will be the collection of all documents associated with it (aggregated across all nodes) in decreasing order of frequency. This response is to be obtained on demand.
b) In order to improve the performance of the query-response model in a), design a Bloom Filter corresponding to each local word-document index and broadcast it to all nodes. Thus each node has one Bloom Filter corresponding to each node. When a query hits a node, it in turn queries all the Bloom Filters to decide whether to send a query to each of the corresponding nodes or not i.e. if a word w is queried in node i, and w is not found in the Bloom Filter j (kept in node i), then the query need not be forwarded to node j. Thus node i will have to aggregate fewer responses.
c) Measure the saved communication cost by using Bloom Filters instead of sending the entire word-index. Measure the communication cost for different numbers of documents and document sizes. For each such input collection of documents, plot the cost curve against the number of processors.