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 PRAM / shared memory parallel algorithms of reasonable difficulty,
    ii. implement the same using C/C++ and OpenMP,
    iii. deploy/run it on a multi-core system, and
    iv. measure its performance and predict its scalability
- Marks: 15% (of the total weight of the course)
- Due Date: Sun. 1st Apr.
- 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 OpenMP constructs.
  - Insall OpenMP on an individual desktop/laptop and use it for development.
  - Once you have a reasonably stable implementation, test it on a server with more cores.
  - Be prepared for at least two iterations of this cycle of { design --> code --> test --> tune }!
- 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.

==========

Exercises

Exercise 1.

a) Use divide-and-conquer to design a PRAM algorithm for the following problem:

   Given a semi-ordered matrix M of numbers find (the position of) a given number K in M. A semi-ordered matrix is one in which each row is sorted (in increasing order) and each column is sorted (in the same order).

b) Implement your parallel algorithm for a) using OpenMP in C/C++.

c) Measure the performance for different sizes of M in the range 10^3 x 10^3 to 10^5 x 10^5. For each size measure the performance for p = 1, 2, .. 2^q where p is the number of cores used.

Exercise 2.

a) Use divide-and-conquer to design a parallel (shared memory) algorithm for the following problem:

   Given a file system and a root directory on Unix, traverse the tree and (i) *extract words from each text file,* (ii) *compute the* ***document frequency (DF)*** *of each word, and* (iii) *determine the words with the K highest DF.*

   Document frequency of a word is defined as the number of documents (i.e. files) in which that word occurs.

b) Implement your parallel algorithm for a) using OpenMP in C/C++. [**Hint:** Refer to man pages for: *readdir()*, *struct dirent,* and *fstat()* . **End of Hint**.]

c) Measure the performance for different input directories by varying the following parameters:

   - maximum depth (4, 16, 64)
   - average depth (2, 8, 32)
   - average branching factor (1.x, 4, 16, 64, 256)
   - total number of files (10^2, 10^4, 10^6, 10^8)

For each input measure the performance for p = 1, 2, .. 2^q where p is the number of cores used.

Exercise 3. [*required only for three-person teams*]

a) Use divide-and-conquer to design a parallel (shared memory) algorithm for the following problem:

   Given a sparse graph G = (V, $E^{-->}$) *and a weight function* **w** *on the edges, reverse its edges.*

   A graph is sparse if |E| = O(|V|*log|V|).

b) Implement your parallel algorithm for a) using OpenMP in C/C++.

c) Measure the performance for different graph sizes i.e.

   - |V| in { 10^4, 10^6, 10^8, 10^{10} } and
   - |E| in { |V| } * { 1.x, 3, log(log(|V|)), (1/c)*log(|V|), log(|V|) }

For each input measure the performance for p = 1, 2, .. 2^q where p is the number of cores used.