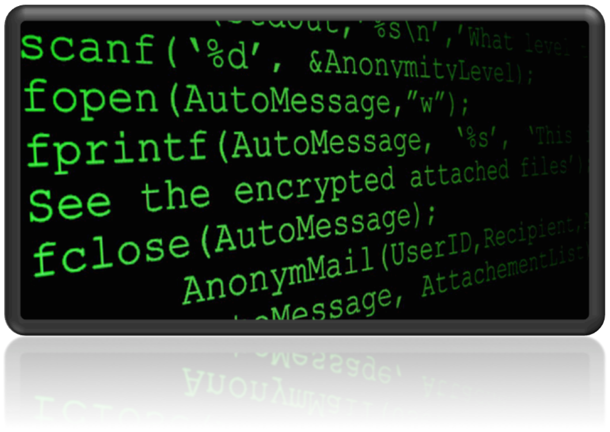
Comp 428
Parallel Computing

Assignment Two



```
                              '%s\n','What level
scanf('%d', &AnonymityLevel);
fopen(AutoMessage,"w");
fprintf(AutoMessage, '%s', 'This
See the encrypted attached files')
fclose(AutoMessage);
          AnonymMail(UserID,Recipient,
          Message, AttachementList
```

Due date: 11:55 PM, Tuesday, October 20.

Note: All submissions are made via Moodle (see "deliverables" section).
Late penalties (< 1 hour = 10%, < 24 hours = 25%) will be applied automatically.

After you upload the assignment, please verify that your submission is recorded as submitted.

Note: Graduate students will be required to solve additional problems and/or provide additional software components. These additional components will be clearly indicated in the description below.

Part A: Concept questions

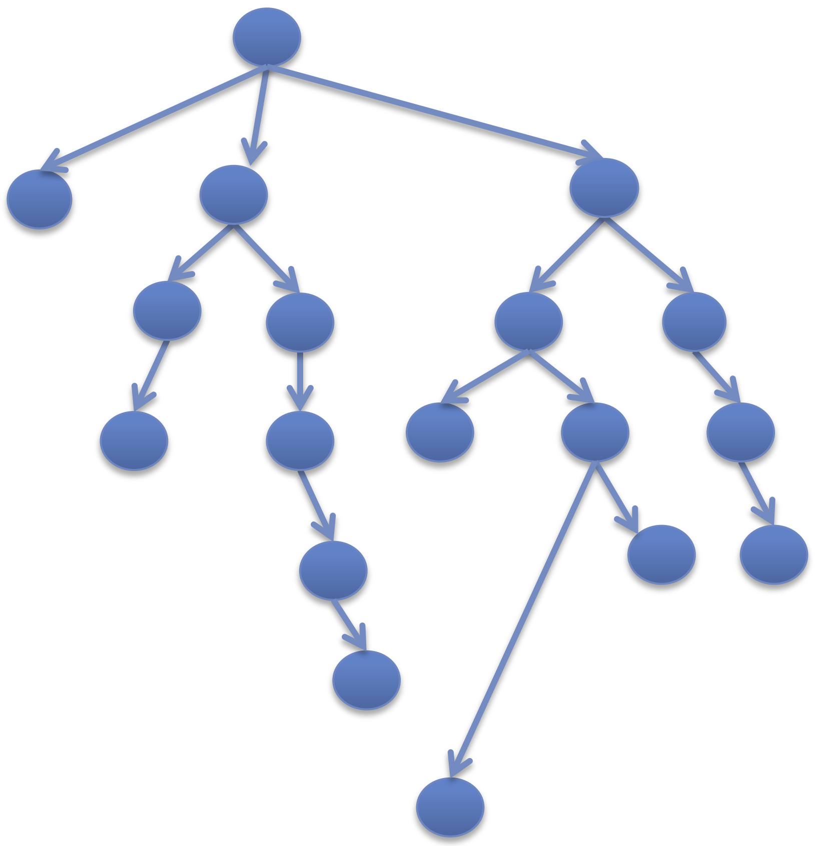
Total Points for Undergads: 30
Total Points for Grad Students: 40

1] Assume that you have the task dependency graph illustrated below. Each node in the graph represents a task to be executed. Recall that these tasks will then be mapped to multiple processes in a parallel implementation.

You will note that there are no weights associated with the nodes. Consequently, when answering the questions, you can make no assumptions about the weights. In other words, they could be large or small – you have to think in terms of best case and worst case scenarios.

You MUST justify your answer in each case. Simply listing a value for your answer will count for no points, without an explanation.

a) (2 points.) What is the maximum degree of concurrency that would be possible in this graph? In other words, how many tasks could potentially be executing concurrently?
b) (2 points) What is the critical path length?
c) (2 points) Assuming that you can use as many processes as you like, what is the maximum possible speedup relative to a single process.
d) (2 points) What is the minimum number of processes needed to achieve the maximum possible speedup?
e) **Grad students.** (2 points) What is the maximum possible speedup if we limit the number of processes to exactly 2?



2] Communication operations:

a) (5 points) In class, we described the process of performing a scatter operation on a 2D hypercube. Repeat this process for a $\sqrt{p}\ x\ \sqrt{p}$ mesh. Draw a series of diagrams, one for each step, labeling the nodes appropriately. For the diagram, assume p = 4.
b) (5 points) What is the cost of this communication algorithm? Provide a detailed explanation to support your claim.
c) **Grad students.** (1 point) Let's say that we want to compute a prefix sum but we do not have a prefix sum communication operation. Explain how you could compute a prefix sum using nothing other than a one-to-all broadcast? You can represent your answer in the form of simple pseudocode.
d) **Grad students.** (1 point) What would be the cost of this method?
e) **Grad students.** (2 points) Could you do this more efficiently using one of the other collective operations that we discussed? Explain your answer.

3] Let us assume that we have a sequential algorithm whose complexity is $\Theta(n^3)$. The parallel version of the algorithm is bounded as $\theta\left(\frac{n^3}{p}\right) + \theta(n^2\ (\log p)^2)$. In this case, the first expression refers to the work per processor, while the second refers to the number of communication rounds.

Because, we will ultimately be charting the efficiency values in this question, we will work with the actual costs instead of the asymptotic notation. As such, we will use the following representation of the cost

$T_S = 8n^3$

and

$T_P = \frac{8n^3}{p} + n^2(\log p)^2$

Given this as a starting point, answer the following questions:

a) (2 point) Provide a concise expression for **Speedup**
b) (1 point) Provide a concise expression for **Parallel Cost**
c) (1 point) Provide a concise expression for **Total Overhead**
d) (3 points) Provide a concise expression for **Efficiency**

e) (5 points) Draw a small table that plots the efficiency for the values of n = (2^5, 2^{10}, 2^{20}, 2^{32}) and p = (4, 32, 128, 1024). Use the style in the class notes (n on rows, and p on the columns)

f) **Grad students**: (4 points) Let's say that we have a second algorithm. Again, using the non-asymptotic form of cost, its sequential time is $4n^2$ and its parallel time is $\frac{4n^2}{p} + 2n$ Compare this to the previous algorithm in terms of the relative efficiency for a fixed n and fixed p. In other words, for a given problem, which of the two algorithms appears to make better use of available resources.

Part B: MPI program

Total Points for Undergads: 30
Total Points for Grad Students: 40

In this first assignment, you will be writing a small MPI program. It is a fairly simple problem to describe. However, bear in mind that if you have never written an MPI application, it can be more work than you first expect to get everything working.

Your primary task will be to process a large text file and count the number of occurrences of each letter of the (English) alphabet. There are 26 letters in the alphabet (52 symbols if you count both lower case and upper case). Letters occur in various frequencies in common text (e.g., **e** is the most common, **z** the least common). Your job is to process the text file to determine the frequencies in this particular case.

The process will be as follows:

1. You will run your parallel program with a user-defined number of processes, between 1 and 5. The exact number will be provided as part of the mpi invocation (e.g., `mpirun -np 3` …). You should provide basic error checking to make sure that the argument is in the appropriate range.

2. The first process (the root) will read the text file from disk and load it into memory. The text file will be called **input.txt** and will be found in the directory from which the executable is run (this makes the IO much easier to manage).

3. The root process will now distribute the text more or less equally to all processes (including the root). In other words, each process will get approximately 1/p of the text. It doesn't matter how you do the division (e.g., round robin distribution or block distribution), as long as each process gets about an equal share of the text.

4. Each process will calculate the frequency of the 26 letters in its partition (count both upper and lower case of each letter). Each process should now display its partial totals for each node. This would look like:

   ```
   Partial Totals:

   Process 0:
    A: 183
    B: 78
    …
    Z: 6

   Process 1:
    A: 118
    …
   and so on
   ```

   There is one very important thing to keep in mind here. You should not just use a print statement on the remote nodes to display this info. If you do this, the output from the various processes can be arbitrarily mixed together if each process tries to print at the same time. This just creates a mess. Instead, each process should send their results back to the root node as a single message. The root node will then print the result, in order (P0, P1,…P5).

5. The data must now be combined into totals. This could be done is several ways but we will do it as follows. Each process will be sent the data corresponding to some of the letters. So, for example, if there were 5 processes, all processes would send their totals for the letters {a,b,c,d,e} to P0. All processes would thent send their totals for {f,g,h,i,j} to P1. Ultimately, each process would get the totals for 5 distinct letters (the last process will actually get 6 letters since 26 does not divide

5 evenly). Note that the number of letters sent to a given process depends on the number of processes being used (e.g., if p = 2, each will get 13 letters).

6. Each process combines its totals into a sum for each of its letters. It then reports its own totals as follows (again printing from Process 0):

```
Process 0 Partial totals
A: 428
B: 237
C: 93
D: 78
E: 647
…
```

7. The root node (Process 0) will now produce a final graphical result in the form of a histogram. For this, you will use "*" characters to create a simple horizontal bar char. It will look like this:

```
A: **************
B: *******
C: ****
D: ***
E: *********************
…
Z: *
```

There is no "rule" on how many "*" characters to use in each bar, but the bar chart must capture the relative size of each frequency.

So that's it. The program is not that large or that complex but it will certainly allow you to get used to the flavor of message passing applications.

Grad version. Graduate students will follow the same application logic. The idea is to count letter frequencies and produce a final histogram of the results. However, in this case, each process will read a separate input file (input0.txt, input1.txt, … inputn.txt, etc.). Each process will then act as the root process for the purpose of calculating its frequencies. It will distribute slices of its file to each of the other process, which will prepare partial totals for the some of the letters of that file.

There are a few other things to keep in mind. First, the frequency calculations for the group of files must be done concurrently. In other words, you cannot serialize the process by simply having Process 0 prepare all of its totals and prints its histograms, then Process 1 does its totals and prints its histogram. You are implementing more of an all-to-all process.

Second, printing should always be done form the process 0. So after each process computes its final totals, it must send these results to the process 0 for printing. There will, of course, be *p* times as much output as there is in the basic version of the program.

Finally, the number of input files used will be between 1 and 5. So you must have 5 files prepared so that the appropriate number can be read, as per the command line argument.

Final Note. It is important that your program actually runs in parallel. In other words, if you just run the program as "mpirun – np 5 my_program", this will actually just run 5 processes on the root node of the cluster.

If you are using PTP, then you will use a Run Configuration that lists the 5 cluster nodes (ap01.apini.private…). If you run the program from the command line, you must specify this same list.

```
mpirun -np 2 --host ap0.apini.private,ap1.apini.private
```

Deliverables:

Part A: The concept questions should be prepared in a Word Processor and then saved in a pdf file named ***A2_concepts.pdf***

Part B: All source code is to be written in C. Multiple source files can used. At the very least, you will have a main file called `frequency.c`, which will contain the `main` method. The final executable will be called `frequency`. You are free to use other source files to structure your code.

You will also need a `makefile` to compile your code. If you are using PTP, then you can use the Managed Makefile approach and allow the PTP plugin to produce the `makefile` for you.

You should also include a README text file to indicate what parts of the programming assignment work or don't work. This may make it easier for the grader to give you points if there are problems/limitations with your solution.

Final Submission: All files (concept questions and source code) should then be combined into a single zip file as indicated below and submitted to Moodle.

LastName_FirstName_A2.zip **(e.g., Smith_John_A2.zip)**