Instructions

- The solutions to all the problems in this homework should be the result of the student’s individual effort. Presenting the words or ideas of somebody else under your name hinders your skills in academic research and violates the university’s policy on academic integrity: http://www.provost.pitt.edu/info/ai1.html
- Turn in a hardcopy of your solutions at the end of the session on Tuesday February 25th, 2014.
- Late submissions will not be accepted.

Problem 1

Construction of the Empire State Building required about two person-millennia of effort, yet the project was completed in only about one year.

(a) How was this possible?

(b) What must the average workforce have been?

(c) The peak workforce was about 3500. What efficiency does this imply for the overall project?

Problem 2

(a) Suppose the runtime of a serial program is given by $T_1 = n^2$, where the units of the runtime are in microseconds. Suppose that a parallelization of this program has runtime $T_p = n^2/p + \log_2(p)$. Write a program that finds the speedups and efficiencies of this program for various values of $n$ and $p$. Run your program with $n = 10, 20, 40, 80, 160, 320$ and $p = 1, 2, 4, 8, 16, 32, 64, 128$. Show a plot of your results. What happens to the speedups and efficiencies as $p$ is increased and $n$ is held fixed? What happens when $p$ is fixed and $n$ is increased?

(b) Suppose that $T_p = T_1/p + T_O$, where $T_O$ stands for the parallel overhead. Also suppose that we fix $p$ and increase the problem size. Show that if $T_O$ grows more slowly than $T_1$, the parallel efficiency will increase as we increase the problem size. Show that if, on the other hand, $T_O$ grows faster that $T_1$, the parallel efficiency will decrease as we increase the problem size.
Problem 3

Consider a program that consists of a large number of iterations, where the time to execute one iteration on a single processor is 1000 nanoseconds. If this program is parallelized on two processors, each iteration would require \((500 + M)\) nanoseconds, where \(M\) is the time to exchange an \(x\)-byte message between the two processors. Assume that \(M = (100 + 10x)\) nanoseconds.

(a) For what value of \(x\) (amount of communication per iteration) would executing the program on two processors result in a speedup larger than 1?

(b) For what value of \(x\) would executing the program on two processors result in an efficiency larger than 0.7?

(c) What is the speedup and efficiency if \(x = 20\) bytes?

(d) Assume that it is possible to buffer data locally and group the communication such that only one message of \(2x\) bytes is exchanged every two iterations, rather than a message of \(x\) bytes every iteration. How would this affect the speedup? Compare the speedup when \(x = 20\) bytes (and messages exchanged every two iterations) with the answer of part (c).

Problem 4

Parallel algorithms can often be represented by dependency graphs. Four such dependency graphs are shown in Figure 1. If a program can be broken into several tasks, then each node of the graph represents one task. The directed edges of the graph represent the dependencies between the tasks of the order in which they must be performed to yield correct results. A node of the dependency graph can be scheduled for execution as soon as the tasks at all the nodes that have incoming edges to that node have finished execution. For example, in Figure 1(B), the nodes on the second level from the root can begin execution only after the task at the root is finished. Any deadlock-free dependency graph must be a directed acyclic graph (DAG); that is, it is devoid of any cycles. All the nodes that are scheduled for execution can be worked on in parallel provided enough processing elements are available. If \(N\) is the number of nodes in a graph, and \(n\) is an integer, then \(N = 2^n - 1\) for graphs (A) and (B), \(N = n^2\) for (C), and \(N = n(n + 1)/2\) for (D). Graphs (A), (B), (C), and (D) are drawn for \(n = 3\) in Figure 1. Assuming each task takes one unit of time and that interprocessor communication is zero, for the algorithms represented in the graphs:

(a) Compute the degree of concurrency (the maximum parallelism).

(b) Compute the maximum possible speedup if an unlimited number of processing elements is available.
(c) Compute the values of speedup, efficiency, and the overhead function if the number of processing elements is (i) the same as the degree of concurrency, and (ii) equal to half of the degree of concurrency.

Problem 5

The original Dijsktra’s algorithm finds the shortest path from one single source node $s$ to all other nodes in a directed graph. If this algorithm is implemented using regular arrays has a complexity of $O(n^2)$, where $n$ is the number of nodes in the graph. The sequential version of the algorithm is given by:

Create a cluster $Q$
Given a source vertex $s$
while (there exist a vertex that is not in the cluster $Q$){
    for (all the vertices outside the cluster $Q$)
        Calculate the distance from non-member vertex to $s$ through the cluster $Q$
        Select the vertex with the shortest path and add it to the cluster $Q$
    endfor
}  

Now consider the following parallelization of Dijsktra’s algorithm:

Create a cluster $Q$
Given a source vertex $s$
Each processor handles a subgroup of $n/p$ vertices
while (there exist a vertex that is not in the cluster $Q$) {
    for (vertices in my subgroup but outside the cluster)
        Calculate the distance from non-member vertex to $s$ through the cluster;
        Select the vertex with the shortest path as the local closest vertex;
    endfor
    Use parallel reduction to find the global closest vertex among all the local closest vertices from each processor.
    Add the closest vertex to the cluster $Q$
} 

(a) Compute the values of $M, W, T, C$ for both the sequential and the parallel version. Assume the graph is stored as a weighted adjacency matrix. Do not include that space in your estimate for $M$.

(b) What are the values of $E$ and $S$ for both the sequential and the parallel version.

(c) Find the isoeficiency function for the parallel algorithm.