CS 525: Parallel Computing
Spring 2001
Problem Set 2: Performance Metrics/Programming Shared Address Space Machines
Due Date: Mar 6, 2001 in class
No late submissions will be accepted

1. If a problem has size W and a serial component of W_s, then show that the peak speedup is bounded by W/W_s.

2. Scaled speedup is defined as the speedup obtained when the problem size is increased linearly with the number of processors. Compute the scaled speedup for the problem of adding n numbers using the communication cost as $t_s + t_w m$.

3. Compute the isoefficiency of a program with

$$W = n^3 t_c$$

and

$$T_p = \frac{n^3}{p} t_c + \log p t_s + \frac{n^2}{\sqrt{p}} t_w$$

4. Plot the speedup of the above program for $t_s = 20\mu s$, $t_w = 0.5\mu s$ for varying values of p. Repeat for the following cases:

   - The value of n is fixed at 100.
   - The value of n is 100 at $p = 1$ and is increased linearly with p.
   - The value of n is 100 and increased as p^2, i.e., for values of n for $p = 1, 2, ...$ are given by $100, 400, 900, ...$ respectively.

   Plot these for p in the range of 1 through 16. Note that you might need some ingenuity in handling large numbers while plotting these.

   (The objective of this exercise is to see how speedup goes down when number of processors goes up for a fixed problem size, and how this can be compensated by increasing the problem size accordingly).

5. Memory constrained scaling: Assume that memory scales up linearly with the number of processors. What is the speedup as a function of p so as to always operate at memory capacity for the problem of adding n numbers. What is the situation in which you expect to see a linear speedup in the memory constrained case in terms of the isoefficiency and serial runtime of the algorithm? (This is a realistic case that you will encounter most of the time).

6. Superlinearity: Consider the parallel formulation of a hash-based join. The serial runtime of joining n records is $n \log n t_c$. The parallel formulation first hashes the records to the right processors and then computes a sort-based join. The parallel runtime of this formulation is given by:

$$T_p = t_{c1}\frac{n}{p} + t_s \log p + t_w \frac{n}{p} + t_c \frac{n}{p} \log(\frac{n}{p})$$

   Using $t_s = 20\mu s$, $t_w = 0.05\mu s$ and $t_{c1} = t_c = 1$, can the above formulation display superlinearity effects? What is the cause of this superlinearity? Plot a speedup curve ($p = 1..128$) for a selected value of n to demonstrate superlinearity.

7. Write a threaded database join. Assume a quick-sort based serial join and hashing across threads. Assume that databases are in the form of 2-D arrays (tables). Each tuple has 5 fields, which are all integers. Assume that you are performing a join on the first field. Show speedup curves on mentor as you increase the number of threads from 1 to 64.

8. Write a threaded program for solving a 15 puzzle. The program takes an initial position and keeps an open list of outstanding positions. This list is sorted on the "goodness" measure of the boards. A simple goodness measure is the Manhattan distance (i.e., the sum of x-displacement and y-displacement of every tile from where it needs to be). This open list is a work queue. Each thread extracts work (a board) from the work queue, expands it to all possible successors, and inserts the successors into the work queue. Make sure you have appropriate locking. Implement the work queue as a heap. Plot the speedup of your program with the number of threads. You can compute the speedups for some reference board that is the same for various thread counts.

9. Modify the above program so that you now have multiple open lists (say k). Now each thread picks a random open list and tries to pick a board from the random list and expands and inserts it back into another, randomly selected list. Plot the speedup of your program with the number of threads. Compare your performance with the previous case. Make sure you use your locks and trylocks carefully to minimize serialization overheads.