Please read this:
   1) Homework assignments are to be completed individually, no teamwork.
   2) Total points possible: 100 pts.
   3) Please add the following statement in the beginning of your submission.
      I have neither given or received unauthorized assistance on this work
      Signed: Date:

Problem 1 (10 pts): Suppose you are given a program that does a fixed amount of work, and some fraction $s$ of that work must be done sequentially. The remaining portion of the work is perfectly parallelizable on $P$ processors. Assuming $T_1$ is the time taken on one processor, derive formula for $T_p$, the time taken on $P$ processors. Use this to get a formula giving an upper bound on the potential speedup on $P$ processors. Explain why it is an upper bound.

Problem 2 (10 pts): Describe an example where a good parallel algorithm must be based on a serial algorithm that is different from the best serial algorithm since the later cannot be parallelized.

Problem 3 (10 pts): In the shared address space parallel equation solver, why do we need all the three barriers in a while loop iteration? What problems will we have if we eliminate them?

Problem 4 (10 pts): What are the advantages and disadvantages of using distributed task queues (as opposed to a global task queue) to implement load balancing? Do small tasks inherently increase communication, contention, and task management overhead in each case?

Problem 5 (10 pts): What are the advantages and disadvantages of static task assignment and dynamic task assignment? In what scenarios is dynamic assignment more desirable than static assignment?

Problem 6 (20 pts): Using data parallel programming mode to write a parallel program that adds two vectors (Hint: use the DECOMP and for_all statements to write the pseudo code):

```c
float v1[100], v2[100];
int nprocs; /* number of processes */
for (i = 0; i < 100; i++)
   v1[i] = v1[i] + v2[i];
```
Problem 7 (30 pts): Using shared memory model and message passing model to write parallel programs that add up the elements of a float-point array (A), whose size (n) and data are input by users. ‘nprocs’ processes are used. You can assume that $n \mod nprocs = 0$. (pseudo code only)

```c
float *A;
int n; /* size of A */
int nprocs; /* number of processes */
float total = 0;

for (i = 0; i < n; i++)
    total += A[i];
```