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/ail.html](http://www.provost.pitt.edu/info/ail.html)

- Turn in a hardcopy of your solutions at the end of the session on Thursday February 6th, 2014.

- Late submissions will not be accepted.

Problem 1

Suppose an automobile assembly line has 12 stations, each of which requires five minutes to perform its portion of the assembly of a given car.

(a) How long does it take to build a single car?

(b) How long does it take to build 10 cars, 100 cars, and 1000 cars?

Explain your answers.

Problem 2

In a parallel computing system with $n$ processors there are two arrays of length $n$ in shared memory. Array $A$ contains a list of names of projects and array $B$ a list of of binary numbers indicating whether the respective project in array $A$ has been selected. For example, if $B[5] = 1$, then project $A[5]$ has been selected. A third array $C$ of length $s$ in shared memory must store the name of all selected projects. The number of selected projects is exactly $s$. Show a parallel algorithm that runs in $O(\log(n))$ time and stores in array $C$ all selected projects, respecting the order in which those projects appear in array $A$. (Hint: use the parallel prefix sum algorithm)

Problem 3

Consider the summation of the elements of a very large vector on a 2 GHz processor. Each cache block in that processor has 4 words (16 bytes) and the memory latency in the system is equivalent to 80 processor clock cycles.
(a) Assume that the processor can execute one addition every clock cycle, what is the minimum memory bandwidth that will allow the processor to achieve peak performance in the case of perfect operand prefetching or unlimited multithreading?

(b) If no prefetching or multithreading is performed, what is the maximum achievable performance (as a fraction of the peak performance) even if we assume an infinite memory bandwidth?

Problem 4

Compile and execute the matrix/vector multiplication program listed in the Appendix. Repeat the process after interchanging the $i$ and $j$ for loops in the computation of $x$. Compare the execution times in the two cases and explain your results. To observe the intended effect, choose the largest value of $N$ that will not cause a segmentation error on your machine. Run the experiment on two machines with different CPU speeds and comment on your results. On Linux, you can use `more /proc/cpuinfo` to know the speed of the cpu.

Problem 5

Consider the floating point pipelined adder discussed in Section 2.2.5 of the course textbook and assume that each of the fetch and store stages take 2 nanoseconds (if the operands are found in the cache) and each of the remaining stages take 1 nanosecond.

(a) How long will an un-pipelined addition of 1000 pairs of numbers take?

(b) How long will a pipelined addition of 1000 pairs take if all operands are found in the cache?

(c) Assume that a cache line contains 4 operands and that it takes fifty nanoseconds to bring a memory line to the cache. How long will a pipelined addition of 1000 pairs take if the cache is initially empty? Here you may assume that the two operands of each addition are located on the same cache line.
Appendix

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#define N 4000

float a[N][N], b[N], x[N];

main ( int argc, char **argv ) {
    int mid = (N+1)/2;
    int i, j;
    double time_start, time_end;
    struct timeval tv;
    struct timezone tz;

    /* Initialize matrix A and vector B. */
    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            if (j == i) { a[i][j] = 2.0; }
            else if (j == i-1 || j == i+1) { a[i][j] = 1.0; }
            else { a[i][j] = 0.01; }
        }
        b[i] = mid - abs(i-mid+1);
    }
    gettimeofday (&tv, &tz);
    time_start = (double)tv.tv_sec + (double)tv.tv_usec / 1000000.0;

    /* Multiply matrix A and vector B. */
    for (i=0; i<N; i++) x[i] = 0.0;
    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            x[i] += a[i][j] * b[j];
        }
    }
    gettimeofday (&tv, &tz);
    time_end = (double)tv.tv_sec + (double)tv.tv_usec / 1000000.0;
    for (i=0; i<8; i++) { printf(" %10.6f",x[i]); }
    printf("\n");
    printf ("time = %lf\n", time_end - time_start);
}