Spring 2014, CSE 392/CS 395 T, Homework 2
Due 11pm March 10, electronically (ZIP) - no extensions

In this homework, you can work in groups. (No more than three people per group.) Each group member must submit the **same** write-up and source files for the code. (For logistical reasons, every member of the group must submit the homework.) Every group should select a "`group`" name.

Submission: Please submit a single ZIP file named `group_lastname_firstname_hwk2.zip`. Please do not use uppercase and do use the underscores.
The zip file should uncompress to a directory named "`group_lastname`" that contains: (a) your write-up "`group.pdf`"; and (b) the makefile and sources to builds three executables `scan`, `search`, and `sort`.

---

1. Given x_0 and$\{a_i, b_i\}_{i=1}^{n-1}$, we seek to compute

$$x_i = a_i x_{i-1} + b_i, \quad i = 1, \cdots, n-1.$$

   An obvious sequential algorithm for this problem has $\mathcal{O}(n)$ complexity. Propose an algorithm, using the work-depth language model, that has $\mathcal{O}(\log n)$ depth. Your proposed algorithm should be work-optimal.

2. Implement an *in-place* a generic **scan** operation in C/C++ using OpenMP (no MPI). Comments:

   - The signature of your function should be similar to the one defined in "`genscan.c`," which is a sequential implementation of a generic scan operation. You may change the interface to improve performance if you wish.
   - In addition to the example I have provided, give an example in which each element in the input array X is a three-dimensional double precision vector and the binary operator is the vector addition.

   Your code should compile using GNU or Intel compilers on x86 platforms. Please report wall clock times for summing (a) 1D and (b) 4D double-precision vectors. The input array should be 300M keys long and should run on Lonestar.

   Write a driver routine called `scan` that takes one argument, the size of array to be searched (initialized to random integers). On the write-up, give pseudocode for your algorithm and report wall clock time/core/n results for n = 1M, 10M, 100M, 1B of elements, using up to one nodes on longhorn (no MPI). Report weak and strong scaling results using 1 core, 1 socket and 1 node.

3. Implement an MPI+OpenMP parallel **binary search algorithm** that has similar signature to the ANSI C `bsearch()` routine but it supports both OpenMP and MPI parallelism and multiple keys. For this reason, it should have three additional arguments (compared to the standard `bsearch`), the number of keys, the MPI communicator and the number of threads per MPI process. Write a driver routine called `search` that takes one argument, the size of

array to be searched (initialized to random integers). On the write-up, give pseudocode for your algorithm, and report wall clock time/core/n results for n = 1M, 10M, 100M, 1B of elements (for 1 key), using up to two nodes on longhorn with 1 MPI task per socket. Report weak and strong scaling results using 1 core, 1 socket, 1 node, and 2 nodes.