HW2: Randomized Selection

Course:	ENEE759K/CMSC751, Spring 2010
Title:	Randomized selection
Date Assigned:	February 23th, 2010
Date Due:	Friday March 12th, 2010, **11:59pm**
Contact:	Alex Tzannes – tzannes@cs.umd.edu
	Please Include [ParAlg] in the subjectline for better email labeling

1 Problem

The objective of this homework is to use the XMT paradigm in order to program a parallel variant of the serial randomized algorithm for selection in expected linear time. The serial algorithm appears in chapter 9.2 of the book *Introduction to Algorithms*, by Cormen, Leiserson, Rivest and Stein.

The project requires writing both a serial and a parallel version, run both on the XMT FPGA and compare running times. Before starting programming derive an iterative variant of the serial algorithm in a text form. Both serial and parallel algorithms must be *ITERATIVE*, not *RECURSIVE*.

For the parallel algorithm:

- The expected number of iterations should be $O(\log n)$
- The expected amount of work should be linear.
- The expected parallel time should be $O(\log^2 n)$

2 Assignment

Your program will take a constant value as a *#define* statement in the header file (see Input section), and display the element ranked at that position. *POSITION=0* means the minimum element, *POSITION=10* means **11th** smallest element etc.

The program must not destroy the initial data array, therefore you may want to start by copying the initial array to a temporary location.

1. Parallel implementation
   (a) Describe the parallel algorithm in file algorithm.p.txt
   (b) Provide a brief work and time complexity analysis of this algorithm. Append this analysis to the file algorithm.p.txt
   (c) Write an XMTC program (*XMTPAR*) that executes this algorithm. Edit the provided code skeleton file selection.p.c

   (d) Run this program using the data sets given in the Input section.
   (e) Collect the number of clock cycles for each run into file table.txt (see Output section).

2. Serial implementation
   (a) Describe the serial algorithm in file algorithm.s.txt
   (b) Provide a brief time complexity analysis of this algorithm. Append this analysis to the file algorithm.s.txt
   (c) Write an XMTC program (*XMTSER*) that executes this algorithm. Edit the provided code skeleton file selection.s.c
   (d) Run this program using the data sets given in the Input section.
   (e) Collect the number of clock cycles for each run into file table.txt (see Output section).

2.1 Setting up the environment

The header files and the binary files can be downloaded from $\sim$ *george/xmtdata*. To get the data files, log in to your account in the class server and copy the *hw2selection.tgz* file from directory using the following commands:

```
$ cp /opt/xmt/class10/xmtdata/selection.tgz ~/
$ tar xzvf selection.tgz
```

This will create the directory *selection* with following folders: *data, src*, and *doc*. Data files are available in data directory. Put your *c* files to *src*, and *txt* files to *doc*.

2.2 Input format

Obtaining random numbers: We do not have a library random number generator at this time. Instead, we have provided a list of pre-generated random numbers in the Input data. The numbers are positive integers in the range $0..1,000,000$. You need to normalize these values to the range that you need in your program.

You should use the random values in the order they are in the array, keeping track of the last used one by using a global variable. In case you need more values than provided, re-use them in a round-robin fashion. The total number of random values available is stored in the *random_numbers_dim0_size* variable in the input data.

#define N	The number of elements in the data array
int array[N]	This array contains N integers for you to work on. You will apply "randomized selection" to this array
#define POSITION 0	The position in the array that you are searching for. (Zero-based counting) #define POSITION 0 means "search for minimum" #define POSITION 4 means "search for the fifth smallest element" #define POSITION 255 means "search for the 256th smallest element". In N=256 dataset, this means "search for the maximum". In datasets with N < 256 this is an ERROR!
int random_numbers[500]	This array contains 500 random numbers that you can use instead of a random number generator

Temporary and auxiliary arrays: You can declare any number of global arrays and variables in your program as needed. The number of elements in the arrays (*n*) is declared as a constant in each dataset, and you can use it to declare auxiliary arrays. For example, this is valid XMTC code:

```
#define N 16384

int temp1[16384];
int temp2[2*N];
int pointer;

int main() {
  //...
}
```

2.3 Data sets

Run all your programs (serial and parallel) using the data files given in the following table. You can directly include the header file into your XMTC code with *#include* or you can include the header file with the compile option *-include*. Remember to also provide the compiler with the *.xbo* file corresponding to the header file you included, e.g.:

```
$> xmtcc -D PRINT_RESULT selection.p.c -include ../data/xsmall/selection.h
../data/xsmall/selection.xbo
```

The X-Small data set is provided for easier tracing/debugging. It will not be included in grading.

Data Set	**N**	**Header File**	**Binary File**	**POSITION** $(logN)*(logN)-9$ **log is based 2 zero-based counting**
X-Small	N=16	data/xsmall/selection.h	data/xsmall/selection.32b	6
Small	N=256	data/small/selection.h	data/small/selection.32b	54
Medium	N=65536	data/medium/selection.h	data/medium/selection.32b	246
Large	N=1048576	data/large/selection.h	data/large/selection.32b	390

2.4 Testing the program

Some results are provided for each dataset.

Data Set	**N**	**POSITION and corresponding result**
X-Small	N=16	POSITION=10 : 56
Small	N=256	POSITION=100 : 348877
Medium	N=65536	POSITION=100 : 1545
Large	N=1048576	POSITION=1000 : 942

To print the result of selection compile using `-D PRINT_RESULT`, e.g.:

```
$> xmtcc -D PRINT_RESULT selection.p.c -include ../data/xsmall/selection.h
../data/xsmall/selection.xbo
```

You can override the POSITION constant by using `-D POSITION=x` when compiling, instead of editing the `#define POSITION` in the dataset files.

Finally, remember to store your result in the `solution` variable declared in the provided files.

2.5 Output

Prepare and fill the following table: Create a text file named table.txt in doc and put the these tables in it.

Remove any *printf* statements from your code while taking these measurements. Printf statements increase the clock count. Therefore the measurements with printf statements may not reflect the actual time and work done. So remember **not** to use the `-D PRINT_RESULT` flag when compiling to get measurements.

XMTPAR Clock Cycles

Input size	**Small**	**Medium**	**Large**
Position=0			
Position=N-1			
Position=(log n)*(log n)-9			

XMTSER Clock Cycles

Input size	**Small**	**Medium**	**Large**
Position=0			
Position=N-1			
Position=(log n)*(log n)-9			

2.6 Submission

The use of the make utility for submission *make submit* is required. Make sure that you have the correct files at correct locations (*src/* and *doc/* directories) using the make submitcheck command. Run following commands in the *src/* subdirectory to submit the assignment:

```
$ make submitcheck
$ make submit
```