HW3: Array Compaction

Course:	Informal Parallel Programming Course for High School Students, Fall 2007
Title:	Array compaction
Date Assigned:	October 9, 2007
Date Due:	October 16, 2007

1 Problem

Input: An array A of (any kind) elements, and another array B of bits (each valued zero or one).
Task: The compaction problem is to find a one-to-one mapping from the subset of elements of $A[i]$, for which $B[i] = 1$, $0 \leq i \leq n-1$, to the sequence $(0, 1, 2, ..., s-1)$, where s is the number of ones in B. Your program should:

- copy the elements $A[i]$ from array A to array C, only if $B[i] = 1$, so that for some s, $C[0], \ldots C[s-1]$ must be full of elements copied from A, and $C[s], \ldots, C[n-1]$ must be empty (untouched and full of 0s in our case)
- write the index i of the copied element in the original array to D, so that, for every $j < s : C[j] = A[D[j]]$

The mapping does not need to be order preserving.

2 Assignment

1. Parallel algorithm
   (a) Write the pseudo-code of the Parallel Array Compaction algorithm in file algorithm.p.txt
   (b) Write a parallel XMTC program compaction.p.c that implements the Parallel Array Compaction algorithm.
   (c) Run this program using 4 sets of data given in the Input section.
   (d) Collect the number of clock cycles for each run into file table.txt (see Output section).
2. Serial algorithm:
   (a) Write the pseudo-code of the Best Serial Array Compaction algorithm in file algorithm.s.txt
   (b) Write a serial XMTC program compaction.s.c that implements the Serial Array Compaction algorithm.
   (c) Run this program using 4 sets of data given in the Input section.
   (d) 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 $\sim$ *swatson/xmtdata*. To get the data files, log in to your account in the class server and copy the *compaction.tgz* file from directory using the following commands:

```
$ cp ~swatson/xmtdata/compaction.tgz ~/
$ tar xzvf compaction.tgz
```

This will create the directory *compaction* 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

You are given two arrays A and B that contains n integers. Each element of B is either 1 or 0. The C array is empty and it initially contains all 0s. The D array is empty and it initially contains all 0s.

#define n	The size of the arrays
int A[n]	The array A
int B[n]	The array B containing 1s and 0s
int C[n]	The array C initially 0
int D[n]	The array D initially 0

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*. To run the compiled program you will need to specify the binary data with *–data-file* option.

Data Set	**Header File**	**Binary file**	**Number of elements to copy**
$n = 50$	data/50/compaction.h	data/50/compaction.32b	23
$n = 4000$	data/4000/compaction.h	data/4000/compaction.32b	2049
$n = 10000$	data/10000/compaction.h	data/10000/compaction.32b	4896
$n = 50000$	data/50000/compaction.h	data/50000/compaction.32b	24970

You can test your programs by checking the number of elements copied for each dataset. Don't forget to remove the *printf* statements when you run your programs to measure cycle counts.

2.4 Output

Prepare and fill the following table: Create a text file named table.txt in doc. **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.

Input size	**n = 50**	**n = 4000**	**n = 10000**	**n = 50000**
Serial Clock Cycles				
Parallel Clock Cycles				

2.5 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 to submit the assignment:

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

If you have any questions, please send an e-mail to Scott Watson, *swatson@umd.edu*