Recitation 16

Hashing and PASL

16.1 Announcements

- PASLLab is due on Friday at midnight. Note that you cannot use late days on this lab.
16.2 Removing Duplicates

Removing duplicates is a crucial substep of many interesting algorithms. For example, in BFS, consider the step where we construct a new frontier. One viable method would be to generate the sequence of all out-neighbors, and then remove duplicates:

$$F' = \text{removeDuplicates}\langle v : u \in F, v \in N^+_G(u)\rangle$$

So, how fast is it to remove duplicates? Can we do it in parallel?

16.2.1 Sequential

Before we think about parallelism, we should acquaint ourselves with a good sequential algorithm solving the same problem. This way, we know what to shoot for in terms of work bounds, since we want our parallel algorithm to be asymptotically work-efficient.

**Task 16.1.** Describe a sequential algorithm which performs expected $O(n)$ work to remove duplicates from a sequence of length $n$. Also argue that $\Omega(n)$ work is necessary in order to solve this problem, and conclude that your algorithm is asymptotically optimal.

*Hint: try hashing elements one at a time.*

16.2.2 Parallel

**Task 16.2.** Implement a function

```plaintext
val removeDuplicates : (α × int → int) → α Seq.t → α Seq.t
where (removeDuplicates h S) returns a sequence of all unique elements of S, given that $h(e, m)$ hashes the element $e$ to a uniform random integer in the range $[0, m)$ (thus the probability of collision for any two distinct elements is $1/m$).

*Hint: as a first attempt, try simultaneously hashing as many elements as possible all at the same time. What do you do when elements collide?*
16.3 **PASL: map_flatten**

If you would like to see the code run on your computer, begin by downloading the files `rec14.hpp` and `rec14-bench.cpp`. You can put these in the top directory of PASLLab once it is released. Then, edit PASLLab’s Makefile to add: `rec14-bench.cpp` to the list of programs, i.e.

```
PROGRAMS=\ 
sandbox.cpp \ 
check.cpp \ 
bench.cpp \ 
rec14-bench.cpp # add me here. 
# don’t forget the slash on the previous line.
```

**Task 16.3. Using PASL primitives, implement the function**

```cpp
template <class Map_func, class Size_func>
sparray map_flatten(const Map_func& f, 
                   const Size_func& g,
                   const sparray& xs);
```

where, at a high-level, the goal is to compute

$$\text{flatten}\left( f(x) : x \in xs \right).$$

Begin by thinking of a sequential implementation and then parallelizing it. You should assume that the function arguments are typed as follows, where $f(xs[i])$ is a pointer to the front of an array of length $g(xs[i])$.

- $f: \text{value\_type} \rightarrow \text{value\_type*}$
- $g: \text{value\_type} \rightarrow \text{long}$
16.4 inject

Throughout the semester, we’ve largely kept the sequence function `inject` shrouded in mystery. Let’s see how the magic works!

**Task 16.4. Using PASL, implement the function**

```pasl
sparray inject(const sparray& xs,
               const sparray& indices,
               const sparray& updates);
```

which returns the result of injecting into `xs`. We require that `indices` and `updates` be the same length, such that for each `i`, we attempt to write `updates[i]` at position `indices[i]` in `xs`. Note that you should not destructively modify `xs`. If there are multiple updates specified at the same position, then all except the last should be ignored. (We want to match the behavior of `inject` as specified in the 15210 Library.)
16.5 Benchmarking

Try running some speedup experiments! The two bench arguments are `map_flatten` and `inject`, respectively. For example, the following injects $m$ randomly placed updates into an array length $n$. In the `map_flatten` benchmark, $n$ is the initial array size, and $m$ is the size of each subarray (so the output is length $nm$).

```
make rec14-bench.opt rec14-bench.baseline

./prun speedup -baseline "./rec14-bench.baseline" \  
-parallel "./rec14-bench.opt -proc 1,5,10,15,20" \  
-bench inject -n 100000,1000000 -m 100000000,200000000

./pplot speedup -series n,m
```