Recitation 15

PASL

15.1 Announcements

- PASLLab is due Friday afternoon.
- We will likely be having a final review sometime on Wednesday, May 4. Keep your ears open for more details.
- The final exam is on Friday, May 6, 1:00-4:00pm.
15.2  map_flatten

Let’s begin by downloading the files rec15.hpp and rec15-bench.cpp. You can put these in the top directory of PASLLab. Then, edit PASLLab’s Makefile to add rec15-bench.cpp to the list of programs, i.e.

```
PROGRAMS=
  sandbox.cpp \
  check.cpp \
  bench.cpp \
  rec15-bench.cpp # add me here.
  # don’t forget the slash on the previous line.
```

**Task 15.1.** Using PASL primitives, implement the function

```
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

\[ \text{flatten} \{ f(x) : x \in xs \} \].

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 \( g(xs[i]) \).

\[ f: \text{value_type} \rightarrow \text{value_type}* \]
\[ g: \text{value_type} \rightarrow \text{long} \]
15.3 inject

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

Task 15.2. Using PASL, implement the function

```cpp
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.)
15.4 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 rec15-bench.opt rec15-bench.baseline

./prun speedup -baseline "./rec15-bench.baseline" \ 
-parallel "./rec15-bench.opt -proc 1,5,10,15,20" \ 
-bench inject -n 100000,1000000 -m 100000000,200000000

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